

# Distributed and Cloud Computing

From Parallel Processing to the Internet of Things

Kai Hwang · Geoffrey C. Fox · Jack J. Dongarra



# Distributed and Cloud Computing

---

## From Parallel Processing to the Internet of Things

Kai Hwang

Geoffrey C. Fox

Jack J. Dongarra



AMSTERDAM • BOSTON • HEIDELBERG • LONDON  
NEW YORK • OXFORD • PARIS • SAN DIEGO  
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann Publishers is an imprint of Elsevier

# Table of Contents

---

Cover image

Title page

Copyright

Dedication

Preface

    About the Book

    A Glance at the Contents

    Key Features

    Readership and Suggestions to Instructors/Students

    Invited Contributions

    Permissions and Acknowledgements

About the Authors

Foreword

**Part 1: Systems Modeling, Clustering, and Virtualization**

Introduction

    Chapter 1: Distributed System Models and Enabling Technologies

    Chapter 2: Computer Clusters for Scalable Parallel Computing

    Chapter 3: Virtual Machines and Virtualization of Clusters and Data Centers

# Chapter 1. Distributed System Models and Enabling Technologies

Summary

1.1 Scalable Computing Over the Internet

1.2 Technologies for Network-Based Systems

1.3 System Models for Distributed and Cloud Computing

1.4 Software Environments for Distributed Systems and Clouds

1.5 Performance, Security, and Energy Efficiency

1.6 Bibliographic Notes and Homework Problems

Acknowledgments

References

Homework Problems

# Chapter 2. Computer Clusters for Scalable Parallel Computing

Summary

2.1 Clustering for Massive Parallelism

2.2 Computer Clusters and MPP Architectures

2.3 Design Principles of Computer Clusters

2.4 Cluster Job and Resource Management

2.5 Case Studies of Top Supercomputer Systems

2.6 Bibliographic Notes and Homework Problems

Acknowledgments

References

Homework Problems

# Chapter 3. Virtual Machines and Virtualization of Clusters and Data Centers

Summary

3.1 Implementation Levels of Virtualization

3.2 Virtualization Structures/Tools and Mechanisms

3.3 Virtualization of CPU, Memory, and I/O Devices

3.4 Virtual Clusters and Resource Management

3.5 Virtualization for Data-Center Automation

3.6 Bibliographic Notes and Homework Problems

Acknowledgments

References

Homework Problems

## Part 2: Computing Clouds, Service-Oriented Architecture, and Programming

### Introduction

Chapter 4: Cloud Platform Architecture over Virtualized Data Centers

Chapter 5: Service-Oriented Architectures for Distributed Computing

Chapter 6: Cloud Programming and Software Environments

### Chapter 4. Cloud Platform Architecture over Virtualized Data Centers

Summary

4.1 Cloud Computing and Service Models

4.2 Data-Center Design and Interconnection Networks

4.3 Architectural Design of Compute and Storage Clouds

4.4 Public Cloud Platforms: GAE, AWS, and Azure

4.5 Inter-cloud Resource Management

4.6 Cloud Security and Trust Management

4.7 Bibliographic Notes and Homework Problems

Acknowledgements

References

Homework Problems

### Chapter 5. Service-Oriented Architectures for Distributed Computing

Summary

5.1 Services and Service-Oriented Architecture

5.2 Message-Oriented Middleware

5.3 Portals and Science Gateways

5.4 Discovery, Registries, Metadata, and Databases

5.5 Workflow in Service-Oriented Architectures

5.6 Bibliographic Notes and Homework Problems

Acknowledgements

References

Homework Problems

## Chapter 6. Cloud Programming and Software Environments

Summary

6.1 Features of Cloud and Grid Platforms

6.2 Parallel and Distributed Programming Paradigms

6.3 Programming Support of Google App Engine

6.4 Programming on Amazon AWS and Microsoft Azure

6.5 Emerging Cloud Software Environments

6.6 Bibliographic Notes and Homework Problems

Acknowledgement

References

Homework Problems

## Part 3: Grids, P2P, and the Future Internet

### Introduction

Chapter 7: Grid Computing Systems and Resource Management

Chapter 8: Peer-to-Peer Computing and Overlay Networks

Chapter 9: Ubiquitous Clouds and the Internet of Things

## Chapter 7. Grid Computing Systems and Resource Management

Summary

7.1 Grid Architecture and Service Modeling

7.2 Grid Projects and Grid Systems Built

7.3 Grid Resource Management and Brokering

7.4 Software and Middleware for Grid Computing

7.5 Grid Application Trends and Security Measures

7.6 Bibliographic Notes and Homework Problems

Acknowledgments

References

## Chapter 8. Peer-to-Peer Computing and Overlay Networks

Summary

8.1 Peer-to-Peer Computing Systems

8.2 P2P Overlay Networks and Properties

8.3 Routing, Proximity, and Fault Tolerance

8.4 Trust, Reputation, and Security Management

8.5 P2P File Sharing and Copyright Protection

8.6 Bibliographic Notes and Homework Problems

Acknowledgements

References

Homework Problems

## Chapter 9. Ubiquitous Clouds and the Internet of Things

Summary

9.1 Cloud Trends in Supporting Ubiquitous Computing

9.2 Performance of Distributed Systems and the Cloud

9.3 Enabling Technologies for the Internet of Things

9.4 Innovative Applications of the Internet of Things

9.5 Online Social and Professional Networking

9.6 Bibliographic Notes and Homework Problems

Acknowledgements

References

Homework Problems

## Index

# Copyright

**Acquiring Editor: Todd Green**

**Development Editor: Robyn Day**

**Project Manager: Danielle S. Miller**

**Designer: Eric DeCicco**

*Morgan Kaufmann* is an imprint of Elsevier  
225 Wyman Street, Waltham, MA 02451, USA

© 2012 Elsevier, Inc. All rights reserved.

AMD, the AMD logo, AMD Opteron, and combinations thereof are trademarks of, and used with permission from, Advanced Micro Devices, Inc.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. Details on how to seek permission, further information about the Publisher's permissions policies and our arrangements with organizations such as the Copyright Clearance Center and the Copyright Licensing Agency, can be found at our website: [www.elsevier.com/permissions](http://www.elsevier.com/permissions).

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

## Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods or professional practices, may become necessary. Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information or methods described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

**Library of Congress Cataloging-in-Publication Data**

Application submitted

**British Library Cataloguing-in-Publication Data**

A catalogue record for this book is available from the British Library.

ISBN: 978-0-12-385880-1

For information on all MK publications visit our website at [www.mkp.com](http://www.mkp.com)

*Typeset by:* diacriTech, Chennai, India

Printed in the United States of America

13 14 15 10 9 8 7 6 5 4 3 2

Working together to grow  
libraries in developing countries

[www.elsevier.com](http://www.elsevier.com) | [www.bookaid.org](http://www.bookaid.org) | [www.sabre.org](http://www.sabre.org)

ELSEVIER

BOOK AID  
International

Sabre Foundation

# Dedication

---

*This book is dedicated to our wives: Jennifer, Judy, and Sue; and to our children: Tony, Andrew, Katherine, Annie, Alexis, James, Heather, Pamela, Oliver, Nick, Ben, and Katie.*

*KH, GF, and JD*

# Preface

---

Kai Hwang, Geoffrey C. Fox and Jack J. Dongarra

## About the Book

Over the past three decades, parallel processing and distributed computing have emerged as a well-developed field in computer science and information technology. Many universities and colleges are now offering standard courses in this field. However, the instructors and students are still in search of a comprehensive textbook that integrates computing theories and information technologies with the design, programming, and application of distributed systems. This book is designed to meet these demands. The book can be also used as a major reference for professionals working in this field.

The book addresses the latest advances in hardware and software, system architecture, new programming paradigms, and ecosystems emphasizing both speed performance and energy efficiency. These latest developments explain how to create high-performance clusters, scalable networks, automated data centers, and high-throughput cloud/grid systems. We also cover programming, and the use of distributed or cloud systems in innovative Internet applications. The book aims to transform traditional multiprocessors and multi-computer clusters into web-scale grids, clouds, and P2P networks for ubiquitous use in the future Internet, including large-scale social networks and the Internet of things that are emerging rapidly in recent years.

## A Glance at the Contents

We have included many milestone developments in a single volume. We present the contributions not only from our own research groups but also from leading research peers in the U.S., China, and Australia. Collectively, this group of authors and contributors summarize the progress that has been made in recent years, ranging from parallel processing to distributed computing and the future Internet.

Starting with an overview of modern distributed models, the text exposes the design principles, system architectures and innovative applications of parallel, distributed, and cloud computing systems. This book attempts to integrate parallel processing technologies with the network-based distributed systems. The book emphasizes scalable physical systems and virtualized data centers and cloud systems for research, e-commerce, social networking, supercomputing, and more applications, using concrete examples from open-source and commercial vendors.

The nine chapters are divided into three Parts: Part 1 covers system models and enabling technologies, including clustering and virtualization. Part 2 presents data center design, cloud computing platforms, service-oriented architectures, and distributed programming paradigms and software support. In Part 3, we study computational/data grids, peer-to-peer networks, ubiquitous clouds, the Internet of Things, and social networks.

Cloud computing material is addressed in six chapters (1, 3, 4, 5, 6, 9). Cloud systems presented include the public clouds: Google AppEngine, Amazon Web Service, Facebook, SalesForce.com, and many others. These cloud systems play an increasingly important role in upgrading the web services and Internet applications. Computer architects, software engineers, and system designers may want to explore the cloud technology to build the future computers and Internet-based systems.

## Key Features

- Coverage of modern distributed computing technology including computer clusters, virtualization, service-oriented architecture, massively parallel processors, peer-to-peer systems, cloud computing, social networks, and the Internet of Things.
- Major emphases of the book lie in exploiting the ubiquity, agility, efficiency, scalability, availability, and programmability of parallel, distributed, and cloud computing systems.
- Latest developments in Hardware, Networks, and System Architecture:
  - Multi-core CPUs and Many-Core GPUs (Intel, Nvidia, AMD)
  - Virtual Machines and Virtual Clusters (CoD, Violin, Amazon VPC)
  - Top-500 Architectures (Tianhe-1A, Jaguar, Roadrunner, etc.)
  - Google AppEngine, Amazon AWS, Microsoft Azure, IBM BlueCloud
  - TeraGrid, DataGrid, ChinaGrid, BOINC, Grid5000 and FutureGrid
  - Chord, Napster, BitTorrent, KaZaA, PPLive, JXTA, and .NET
  - RFID, Sensor Networks, GPS, CPS, and the Internet of Things
  - Facebook, Force.Com, Twitter, SGI Cylone, Nebula, and GoGrid
- Recent advances in paradigms, programming, software and ecosystems:
  - MapReduce, Dryad, Hadoop, MPI, Twister, BigTable, DISC, etc
  - Cloud Service and Trust Models (SaaS, IaaS, PaaS, and PowerTrust)
  - Programming Languages and Protocols (Python, SOAP, UDDI, Pig Latin)
  - Virtualization Software (XEN, KVM, VMWare ESX, etc.)
  - Cloud OS and Meshups (Eucalyptus, Nimbus, OpenNebula, vSphere/4, etc.)
  - Service-Oriented Architecture (REST, WS, Web 2.0, OGSA, etc.)
  - Distributed Operating Systems (DCE, Amoeba, and MOSIX)
  - Middleware and Software Libraries (LSF, Globus, Hadoop, Aneka)
- Over 100 examples are illustrated with 300 figures, designed to meet the need of students taking a distributed system course. Each chapter includes exercises and further reading.
- Included are case studies from the leading distributed computing vendors: Amazon,

Google, Microsoft, IBM, HP, Sun, Silicon Graphics, Rackspace, SalesForce.com, netSuite, Enomaly, and many more.

## Readership and Suggestions to Instructors/Students

The readership includes students taking a distributed systems or distributed computing class. Professional system designers and engineers may find this book useful as a reference to the latest distributed system technologies including clusters, grids, clouds, and the Internet of Things. The book gives a balanced coverage of all of these topics, looking into the future of Internet and IT evolutions.

The nine chapters are logically sequenced for use in an one-semester (45-hour lectures) course for seniors and graduate-level students. For use in a tri-semester system, [Chapters 1, 2, 3, 4, 6, and 9](#) are suitable for a 10-week course (30-hour lectures). In addition to solving homework problems, the students are advised to conduct some parallel and distributed programming experiments on available clusters, grids, P2P, and cloud platforms. Sample projects and a solutions manual will be made available to proven instructors from Morgan Kaufmann, Publishers.

## Invited Contributions

The book was jointly planned, authored, edited, and proofread by all three lead authors in four years (2007–2011). Over this period, we have invited and received partial contributions and technical assistance from the following scientists, researchers, instructors, and doctoral students from 10 top Universities in the U.S., China, and Australia.

Listed below are the invited contributors to this book. The authorship, contributed sections, and editorship of individual chapters are explicitly identified at the end of each chapter, separately. We want to thank their dedicated work and valuable contributions throughout the courses of repeated writing and revision process. The comments by the anonymous reviewers are also useful to improve the final contents.

**Albert Zomaya, Nikzad Babaii Rizvandi, Young-Choon Lee, Ali Boloori, Reza Moraveji, Javid Taheri, and Chen Wang**, Sydney University, Australia

**Rajkumar Buyya**, University of Melbourne, Australia

**Yongwei Wu, Weimin Zheng, and Kang Chen**, Tsinghua University, China

**Zhenyu Li, Ninghui Sun, Zhiwei Xu, and Gaogang Xie**, Chinese Academy of Sciences

**Zhibin Yu, Xiaofei Liao and Hai Jin**, Huazhong University of Science and Technology

**Judy Qiu, Shrideep Pallickara, Marlon Pierce, Suresh Marru, Gregor Laszewski, Javier Diaz, Archit Kulshrestha, and Andrew J. Younge**, Indiana University

**Michael McLennan, George Adams III, and Gerhard Klimeck**, Purdue University

**Zhongyuan Qin, Kaikun Dong, Vikram Dixit, Xiaosong Lou, Sameer Kulkarni, Ken Wu, Zhou Zhao, and Lizhong Chen**, University of Southern California

**Renato Figueiredo**, University of Florida

**Michael Wilde**, University of Chicago.

## Permissions and Acknowledgements

The permission to use copyrighted illustrations is openly acknowledged in the respective figure captions. In particular, we would like to thank Bill Dally, John Hopcroft, Mendel Roseblum, Dennis Gannon, Jon Kleinberg, Rajkumar Buyya, Albert Zomaya, Randel Bryant, Kang Chen, and Judy Qiu for their generosity in allowing us to use their presentation slides, original figures, and illustrated examples in this book.

We want to thank Ian Foster who wrote the visionary Foreword to introduce this book to our readers. The sponsorship by Todd Green and the editorial work of Robyn Day from Morgan Kaufmann Publishers and the production work led by Dennis Troutman of diacriTech are greatly appreciated. Without the collective effort of all of the above individuals, this book might be still in preparation. We hope that our readers will enjoy reading this timely-produced book and give us feedback for amending omissions and future improvements.

# About the Authors

---

**Kai Hwang** is a Professor of Electrical Engineering and Computer Science, University of Southern California. Presently, he also serves as an Intellectual Venture-endowed Visiting Chair Professor at Tsinghua University in Beijing. He earned his Ph.D. in EECS from University of California at Berkeley in 1972. He has taught at Purdue University for 12 years prior to joining USC in 1985. Hwang has served as the founding Editor-in-Chief of the *Journal of Parallel and Distributed Computing* for 26 years. He is a world-renowned scholar and educator in computer science and engineering. He has published 8 books and 220 original papers in computer architecture, digital arithmetic, parallel processing, distributed systems, Internet security, and cloud computing.

Four of his published books: *Computer Arithmetic* (Wiley, 1978), *Computer Architecture and Parallel Processing* (McGraw-Hill, 1983), *Advanced Computer Architecture* (McGraw-Hill, 1993), and *Scalable Parallel Computing* (McGraw-Hill, 1998) have been translated into Spanish, Japanese, Chinese, and Korean from the English editions. By 2011, his published papers and books were cited over 9,000 times. Dr. Hwang has delivered three dozen keynote addresses in major IEEE/ACM conferences. In 1986, the IEEE Computer Society elevated him to an IEEE Fellow. He received the 2004 Outstanding Achievement Award from China Computer Federation. In May 2011, he received the IPDPS Founders' Award for his pioneering contributions in the field of parallel processing. For details, visit the web page: <http://GridSec.usc.edu/hwang.html>.

**Geoffrey C. Fox** is a Distinguished Professor of Informatics, Computing and Physics and Associate Dean of Graduate Studies and Research in the School of Informatics and Computing, Indiana University. He has taught and led many research groups at Caltech and Syracuse University, previously. He received his Ph.D. from Cambridge University, U.K. Fox is well known for his comprehensive work and extensive publications in parallel architecture, distributed programming, grid computing, web services, and Internet applications. His book on *Grid Computing* (coauthored with F. Berman and Tony Hey) is widely used by the research community. He has produced over 60 Ph.D. students in physics, computer science, and engineering over the years. Contact him via: [gcf@indiana.edu](mailto:gcf@indiana.edu).

**Jack J. Dongarra** is a University Distinguished Professor of Electrical Engineering and Computer Science, University of Tennessee, a Distinguished Research Staff, Oak Ridge National Laboratory, and a Turning Fellow at the University of Manchester. An ACM/IEEE/SIAM/AAAS Fellow, Dongarra pioneered the areas of supercomputer benchmarks, numerical analysis, PDE solvers, and high-performance computing and published extensively in these areas. He leads the Linpack benchmark evaluation of the

Top-500 fastest computers over the years. Based on his high contributions in the supercomputing and high-performance areas, he was elected as a Member of the National Academy of Engineering in the U.S. Contact him at [dongarra@eeecs.utk.edu](mailto:dongarra@eeecs.utk.edu).

# Foreword

---

Ian Foster

*Jackson Hole, Wyoming, August, 2011*

Richard Feynman, in his wonderful autobiography *Surely You're Joking, Mr. Feynman*, recounts how at Los Alamos in 1944 he was responsible for supervising the human computers who performed the long and tedious calculations required by the Manhattan Project. Using the mechanical calculators that were then the state of the art, the best human computer could achieve only one addition or multiplication every few seconds. Feynman and his team thus developed methods for decomposing problems into smaller tasks that could be performed simultaneously by different people (they passed cards with intermediate results between people operating adders, multipliers, collators, and sorters); for running multiple computations at once in the same computing complex (they used different color cards); for prioritizing a more important computation (they eliminated cards of other colors); and for detecting and recovering efficiently from errors (relevant cards, and their descendants, were removed, and computations restarted).

Seventy years later, computer architects face similar challenges—and have adopted similar solutions. Individual computing devices are far faster, but physical constraints still limit their speed. Thus, today's computing landscape is characterized by pervasive parallelism. Individual processors incorporate pipelining, parallel instructions, speculative execution, and multithreading. Essentially every computer system, from the humblest desktop to the most powerful supercomputer, incorporates multiple processors. Designers of future exascale supercomputers, to be capable of  $10^{18}$  operations per second, tell us that these computers will need to support  $10^7$  concurrent operations.

Parallelism is fundamentally about communication and coordination, and those two activities have also been transformed over the past seventy years by dramatic technological change. Light is no faster, at 8 inches or 20 centimeters per nanosecond in fiber, than in Feynman's time; one can never expect to send a message in less than 50 milliseconds from Los Angeles to Auckland. But the rate at which data can be transmitted has changed dramatically, from a few characters per second in 1910 (early telegraphs) to thousands of characters per second in 1960 (ARPANET) to more than 10 billion characters per second over optical fibers in 2010.

Quasi-ubiquitous high-speed communications not only allows call centers to be relocated to India, it also allows computation to be moved to centralized facilities that achieve massive economies of scale, and permits enormous quantities of data to be collected and organized to support decision making by people worldwide. Thus, government agencies, research laboratories, and companies who need to simulate

complex phenomena create and operate enormous supercomputers with hundreds of thousands of processors. Similarly, companies such as Google, Facebook, and Microsoft who need to process large quantities of data operate numerous massive “cloud” data centers that may each occupy tens of thousands of square feet and contain tens or hundreds of thousands of computers. Like Feynman’s Los Alamos team, these computing complexes provide computing as a service for many people, and must juggle many computations performed for different purposes.

Massive parallelism, ultra-fast communication, and massive centralization are all fundamental to human decision making today. The computations that are used to forecast tomorrow’s weather, index the web, recommend movies, suggest social connections, predict the future state of the stock market, or provide any one of a multitude of other desirable information products are typically distributed over thousands of processors and depend on data collected from sometimes millions of sources worldwide. Indeed, little of the modern world could function as it does without parallel and distributed computing.

In this pervasively parallel and distributed world, an understanding of distributed computing is surely an essential part of any undergraduate education in computer science. (Indeed, I would argue, an understanding of these topics should be an essential part of *any* undergraduate education. But I leave that argument for another time.) The most complex computer systems today are no longer individual microprocessors, but entire data centers. The most complex computer programs written today are those that manage or run on data-center-scale systems. A student who graduates with a degree in computer science and does not understand how these systems and programs are constructed is profoundly unprepared to engage productively in the modern workforce.

Hwang, Fox, and Dongarra’s text is thus especially timely. In its three sections, it covers progressively the hardware and software architectures that underpin modern massively parallel computer systems; the concepts and technologies that enable cloud and distributed computing; and advanced topics in distributed computing, including grid, peer-to-peer, and the Internet of Things. In each area, the text takes a systems approach, describing not only concepts but also representative technologies and realistic large-scale distributed computing deployments. Computing is as much an engineering discipline as a science, and these descriptions of real systems will both prepare students to use those systems and help them understand how other architects have maneuvered the constraints associated with large-scale distributed system design.

The text also addresses some of the more challenging issues facing computer science researchers today. To name just two, computers have emerged as a major consumer of electricity, accounting for several percent of all electricity used in the US. (In Japan, it is ironic that following the 2011 tsunami, the large supercomputers that may help prepare for future natural disasters must often be turned off to conserve power.) And, the fact that 98% of the roughly 10 billion processors sold each year are for embedded devices, and that these embedded devices are increasingly communication-enabled, introduces the opportunity and challenge of an “Internet of Things” that will be vastly larger, more complex, and more capable than today’s Internet of People.

I hope that the appearance of this book will stimulate more teaching of distributed computing in universities and colleges—and not just as an optional topic, as is too often the case, but as a core element of the undergraduate curriculum. I hope also that others outside universities will take this opportunity to learn about distributed computing, and more broadly about what computing looks like on the cutting edge: sometimes messy; often complex; but above all tremendously exciting.

# PART 1

# Systems Modeling, Clustering, and Virtualization

## OUTLINE

Introduction

Chapter 1 Distributed System Models and Enabling Technologies

Chapter 2 Computer Clusters for Scalable Parallel Computing

Chapter 3 Virtual Machines and Virtualization of Clusters and Data Centers

# Introduction

---

The first three chapters cover systems models and review two enabling technologies. We model distributed systems and cloud platforms in [Chapter 1](#). The clustering technology is presented in [Chapter 2](#), and virtualization technology in [Chapter 3](#). These two technologies enable distributed and cloud computing. The system models include computer clusters, computing grid, P2P networks, and cloud computing platform. System clustering is supported by hardware, software, and middleware advances. Virtualization creates virtual machines, virtualized clusters, automation of datacenters, and building of elastic cloud platforms.

# Chapter 1: Distributed System Models and Enabling Technologies

This introductory chapter assesses the evolutional changes in computing and IT trends in the past 30 years. We study both high-performance computing (HPC) for scientific computing and high-throughput computing (HTC) systems for business computing. We examine clusters/MPP, grids, P2P networks, and Internet clouds. These systems are distinguished by their platform architectures, OS platforms, processing algorithms, communication protocols, security demands, and service models applied. The study emphasizes the scalability, performance, availability, security, energy-efficiency, workload outsourcing, data center protection, and so on.

This chapter is authored by Kai Hwang with partial contributions by Geoffrey Fox ([Section 1.4.1](#)) and Albert Zomaya ([Section 1.5.4](#)), and assisted by Nikzad Babaii Rizvandi, Young-Choon Lee, Xiaosong Lou, and Lizhong Chen. The final manuscript was edited by Jack Dongarra.

# Chapter 2: Computer Clusters for Scalable Parallel Computing

Clustering enables the construction of scalable parallel and distributed systems for both HPC and HTC applications. Today's cluster nodes are built with either physical servers or virtual machines. In this chapter, we study clustered computing systems and massively parallel processors. We focus on design principles and assess the hardware, software, and middleware support needed. We study scalability, availability, programmability, single system image, job management, and fault tolerance. We study the clustering MPP architectures in three top supercomputers reported in recent years, namely China's Tiahe-1A, the Cray XT5 Jaguar, and IBM's RoadRunner.

This chapter is coauthored by Kai Hwang and Jack Dongarra with partial contributions by Rajkumar Buyya and Ninghui Sun. Special technical assistances are from Zhiwei Xu, Zhou Zhao, Xiaosong Lou, and L zhong Chen.

# **Chapter 3: Virtual Machines and Virtualization of Clusters and Data Centers**

Virtualization technology was primarily designed for the sharing of expensive hardware resources by multiplexing virtual machines (VM) on the same set of hardware hosts. The surge of interest in installing virtual machines has widened the scope of system applications and upgraded computer performance and efficiency in recent years. We study VMs, live migration of VMs, virtual cluster construction, resource provisioning, virtual configuration adaptation, and the design of virtualized data centers for cloud computing applications. We emphasize the roles of using virtual clusters and virtualized resource management for building dynamic grids and cloud platforms.

This chapter is coauthored by Zhibin Yu and Kai Hwang with technical assistance from Hai Jin, Xiaofei Liao, Chongyuan Qin, Lizhong Chen, and Zhou Zhao.

# CHAPTER 1

# Distributed System Models and Enabling Technologies

## CHAPTER OUTLINE

### Summary

#### 1.1 Scalable Computing over the Internet

- 1.1.1 The Age of Internet Computing
- 1.1.2 Scalable Computing Trends and New Paradigms
- 1.1.3 The Internet of Things and Cyber-Physical Systems

#### 1.2 Technologies for Network-based Systems

- 1.2.1 Multicore CPUs and Multithreading Technologies
- 1.2.2 GPU Computing to Exascale and Beyond
- 1.2.3 Memory, Storage, and Wide-Area Networking
- 1.2.4 Virtual Machines and Virtualization Middleware
- 1.2.5 Data Center Virtualization for Cloud Computing

#### 1.3 System Models for Distributed and Cloud Computing

- 1.3.1 Clusters of Cooperative Computers
- 1.3.2 Grid Computing Infrastructures
- 1.3.3 Peer-to-Peer Network Families
- 1.3.4 Cloud Computing over the Internet

#### 1.4 Software Environments for Distributed Systems and Clouds

- 1.4.1 Service-Oriented Architecture (SOA)
- 1.4.2 Trends toward Distributed Operating Systems
- 1.4.3 Parallel and Distributed Programming Models

#### 1.5 Performance, Security, and Energy Efficiency

- 1.5.1 Performance Metrics and Scalability Analysis
- 1.5.2 Fault Tolerance and System Availability
- 1.5.3 Network Threats and Data Integrity
- 1.5.4 Energy Efficiency in Distributed Computing

#### 1.6 Bibliographic Notes and Homework Problems

### Acknowledgments

This chapter presents the evolutionary changes that have occurred in parallel, distributed, and cloud computing over the past 30 years, driven by applications with variable workloads and large data sets. We study both high-performance and high-throughput computing systems in parallel computers appearing as computer clusters, service-oriented architecture, computational grids, peer-to-peer networks, Internet clouds, and the Internet of Things. These systems are distinguished by their hardware architectures, OS platforms, processing algorithms, communication protocols, and service models applied. We also introduce essential issues on the scalability, performance, availability, security, and energy efficiency in distributed systems.

scalable computing, distributed systems, virtualization, parallel programming, security, peer-to-peer networks

## Summary

This chapter presents the evolutionary changes that have occurred in parallel, distributed, and cloud computing over the past 30 years, driven by applications with variable workloads and large data sets. We study both high-performance and high-throughput computing systems in parallel computers appearing as computer clusters, service-oriented architecture, computational grids, peer-to-peer networks, Internet clouds, and the Internet of Things. These systems are distinguished by their hardware architectures, OS platforms, processing algorithms, communication protocols, and service models applied. We also introduce essential issues on the scalability, performance, availability, security, and energy efficiency in distributed systems.

### 1.1 Scalable Computing Over the Internet

Over the past 60 years, computing technology has undergone a series of platform and environment changes. In this section, we assess evolutionary changes in machine architecture, operating system platform, network connectivity, and application workload. Instead of using a centralized computer to solve computational problems, a parallel and distributed computing system uses multiple computers to solve large-scale problems over the Internet. Thus, distributed computing becomes data-intensive and network-centric. This section identifies the applications of modern computer systems that practice parallel and distributed computing. These large-scale Internet applications have significantly enhanced the quality of life and information services in society today.

#### 1.1.1 The Age of Internet Computing

Billions of people use the Internet every day. As a result, supercomputer sites and large

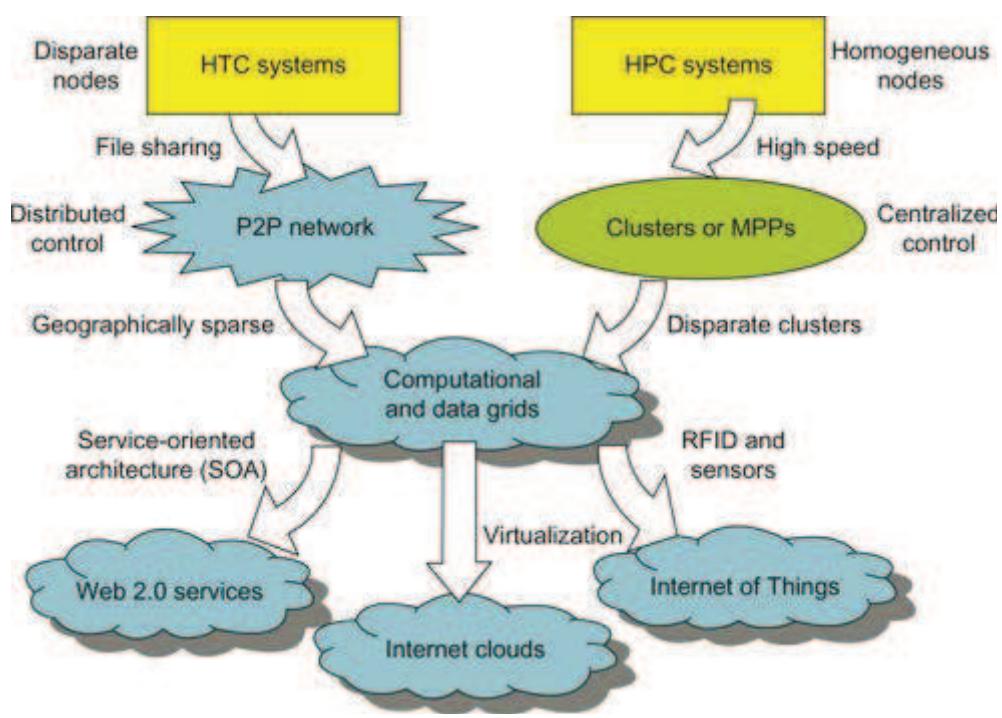
data centers must provide high-performance computing services to huge numbers of Internet users concurrently. Because of this high demand, the Linpack Benchmark for *high-performance computing (HPC)* applications is no longer optimal for measuring system performance. The emergence of computing clouds instead demands *high-throughput computing (HTC)* systems built with parallel and distributed computing technologies [5,6,19,25]. We have to upgrade data centers using fast servers, storage systems, and high-bandwidth networks. The purpose is to advance network-based computing and web services with the emerging new technologies.

### **1.1.1.1 The Platform Evolution**

Computer technology has gone through five generations of development, with each generation lasting from 10 to 20 years. Successive generations are overlapped in about 10 years. For instance, from 1950 to 1970, a handful of mainframes, including the IBM 360 and CDC 6400, were built to satisfy the demands of large businesses and government organizations. From 1960 to 1980, lower-cost minicomputers such as the DEC PDP 11 and VAX Series became popular among small businesses and on college campuses.

From 1970 to 1990, we saw widespread use of personal computers built with VLSI microprocessors. From 1980 to 2000, massive numbers of portable computers and pervasive devices appeared in both wired and wireless applications. Since 1990, the use of both HPC and HTC systems hidden in clusters, grids, or Internet clouds has proliferated. These systems are employed by both consumers and high-end web-scale computing and information services.

The general computing trend is to leverage shared web resources and massive amounts of data over the Internet. [Figure 1.1](#) illustrates the evolution of HPC and HTC systems. On the HPC side, supercomputers (*massively parallel processors* or *MPPs*) are gradually replaced by clusters of cooperative computers out of a desire to share computing resources. The cluster is often a collection of homogeneous compute nodes that are physically connected in close range to one another. We will discuss clusters, MPPs, and grid systems in more detail in [Chapters 2 and 7](#).



**FIGURE 1.1** Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

On the HTC side, *peer-to-peer (P2P)* networks are formed for distributed file sharing and content delivery applications. A P2P system is built over many client machines (a concept we will discuss further in [Chapter 5](#)). Peer machines are globally distributed in nature. P2P, cloud computing, and web service platforms are more focused on HTC applications than on HPC applications. Clustering and P2P technologies lead to the development of computational grids or data grids.

### 1.1.1.2 High-Performance Computing

For many years, HPC systems emphasize the raw speed performance. The speed of HPC systems has increased from Gflops in the early 1990s to now Pflops in 2010. This improvement was driven mainly by the demands from scientific, engineering, and manufacturing communities. For example, the Top 500 most powerful computer systems in the world are measured by floating-point speed in Linpack benchmark results. However, the number of supercomputer users is limited to less than 10% of all computer users. Today, the majority of computer users are using desktop computers or large servers when they conduct Internet searches and market-driven computing tasks.

### 1.1.1.3 High-Throughput Computing

The development of market-oriented high-end computing systems is undergoing a strategic change from an HPC paradigm to an HTC paradigm. This HTC paradigm pays more attention to high-flux computing. The main application for high-flux computing is in Internet searches and web services by millions or more users simultaneously. The performance goal thus shifts to measure *high throughput* or the number of tasks completed per unit of time. HTC technology needs to not only improve in terms of batch processing speed, but also address the acute problems of cost, energy savings, security,

and reliability at many data and enterprise computing centers. This book will address both HPC and HTC systems to meet the demands of all computer users.

#### 1.1.1.4 Three New Computing Paradigms

As Figure 1.1 illustrates, with the introduction of SOA, Web 2.0 services become available. Advances in virtualization make it possible to see the growth of Internet clouds as a new computing paradigm. The maturity of *radio-frequency identification (RFID)*, *Global Positioning System (GPS)*, and sensor technologies has triggered the development of the *Internet of Things (IoT)*. These new paradigms are only briefly introduced here. We will study the details of SOA in Chapter 5; virtualization in Chapter 3; cloud computing in Chapters 4, 6, and 9; and the IoT along with cyber-physical systems (CPS) in Chapter 9.

When the Internet was introduced in 1969, Leonard Kleinrock of UCLA declared: "As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of computer utilities, which like present electric and telephone utilities, will service individual homes and offices across the country." Many people have redefined the term "computer" since that time. In 1984, John Gage of Sun Microsystems created the slogan, "The network is the computer." In 2008, David Patterson of UC Berkeley said, "The data center is the computer. There are dramatic differences between developing software for millions to use as a service versus distributing software to run on their PCs." Recently, Rajkumar Buyya of Melbourne University simply said: "The cloud is the computer."

This book covers clusters, MPPs, P2P networks, grids, clouds, web services, social networks, and the IoT. In fact, the differences among clusters, grids, P2P systems, and clouds may blur in the future. Some people view clouds as grids or clusters with modest changes through virtualization. Others feel the changes could be major, since clouds are anticipated to process huge data sets generated by the traditional Internet, social networks, and the future IoT. In subsequent chapters, the distinctions and dependencies among all distributed and cloud systems models will become clearer and more transparent.

#### 1.1.1.5 Computing Paradigm Distinctions

The high-technology community has argued for many years about the precise definitions of centralized computing, parallel computing, distributed computing, and cloud computing. In general, *distributed computing* is the opposite of *centralized computing*. The field of *parallel computing* overlaps with distributed computing to a great extent, and *cloud computing* overlaps with distributed, centralized, and parallel computing. The following list defines these terms more clearly; their architectural and operational differences are discussed further in subsequent chapters.

- **Centralized computing** This is a computing paradigm by which all computer resources are centralized in one physical system. All resources (processors, memory, and storage) are fully shared and tightly coupled within one integrated OS. Many data

centers and supercomputers are *centralized systems*, but they are used in parallel, distributed, and cloud computing applications [18,26].

- **Parallel computing** In parallel computing, all processors are either tightly coupled with centralized shared memory or loosely coupled with distributed memory. Some authors refer to this discipline as *parallel processing* [15,27]. Interprocessor communication is accomplished through shared memory or via message passing. A computer system capable of parallel computing is commonly known as a *parallel computer* [28]. Programs running in a parallel computer are called *parallel programs*. The process of writing *parallel programs* is often referred to as *parallel programming* [32].
- **Distributed computing** This is a field of computer science/engineering that studies distributed systems. A *distributed system* [8,13,37,46] consists of multiple autonomous computers, each having its own private memory, communicating through a computer network. Information exchange in a distributed system is accomplished through *message passing*. A computer program that runs in a distributed system is known as a *distributed program*. The process of writing distributed programs is referred to as *distributed programming*.
- **Cloud computing** An *Internet cloud* of resources can be either a centralized or a distributed computing system. The cloud applies parallel or distributed computing, or both. Clouds can be built with physical or virtualized resources over large data centers that are centralized or distributed. Some authors consider cloud computing to be a form of *utility computing* or *service computing* [11,19].

As an alternative to the preceding terms, some in the high-tech community prefer the term *concurrent computing* or *concurrent programming*. These terms typically refer to the union of parallel computing and distributing computing, although biased practitioners may interpret them differently. *Ubiquitous computing* refers to computing with pervasive devices at any place and time using wired or wireless communication. The *Internet of Things* (IoT) is a networked connection of everyday objects including computers, sensors, humans, etc. The IoT is supported by Internet clouds to achieve ubiquitous computing with any object at any place and time. Finally, the term *Internet computing* is even broader and covers all computing paradigms over the Internet. This book covers all the aforementioned computing paradigms, placing more emphasis on distributed and cloud computing and their working systems, including the clusters, grids, P2P, and cloud systems.

### 1.1.1.6 Distributed System Families

Since the mid-1990s, technologies for building P2P networks and *networks of clusters* have been consolidated into many national projects designed to establish wide area computing infrastructures, known as *computational grids* or *data grids*. Recently, we have witnessed a surge in interest in exploring Internet cloud resources for data-intensive applications. Internet clouds are the result of moving desktop computing to service-oriented computing using server clusters and huge databases at data centers. This chapter introduces the basics of various parallel and distributed families. Grids and

clouds are disparity systems that place great emphasis on resource sharing in hardware, software, and data sets.

Design theory, enabling technologies, and case studies of these massively distributed systems are also covered in this book. Massively distributed systems are intended to exploit a high degree of parallelism or concurrency among many machines. In October 2010, the highest performing cluster machine was built in China with 86016 CPU processor cores and 3,211,264 GPU cores in a Tianhe-1A system. The largest computational grid connects up to hundreds of server clusters. A typical P2P network may involve millions of client machines working simultaneously. Experimental cloud computing clusters have been built with thousands of processing nodes. We devote the material in [Chapters 4](#) through [6](#) to cloud computing. Case studies of HTC systems will be examined in [Chapters 4](#) and [9](#), including data centers, social networks, and virtualized cloud platforms

In the future, both HPC and HTC systems will demand multicore or many-core processors that can handle large numbers of computing threads per core. Both HPC and HTC systems emphasize parallelism and distributed computing. Future HPC and HTC systems must be able to satisfy this huge demand in computing power in terms of throughput, efficiency, scalability, and reliability. The system efficiency is decided by speed, programming, and energy factors (i.e., *throughput per watt* of energy consumed). Meeting these goals requires to yield the following design objectives:

- **Efficiency** measures the utilization rate of resources in an execution model by exploiting massive parallelism in HPC. For HTC, efficiency is more closely related to job throughput, data access, storage, and power efficiency.
- **Dependability** measures the reliability and self-management from the chip to the system and application levels. The purpose is to provide high-throughput service with Quality of Service (QoS) assurance, even under failure conditions.
- **Adaptation in the programming model** measures the ability to support billions of job requests over massive data sets and virtualized cloud resources under various workload and service models.
- **Flexibility in application deployment** measures the ability of distributed systems to run well in both HPC (science and engineering) and HTC (business) applications.

## 1.1.2 Scalable Computing Trends and New Paradigms

Several predictable trends in technology are known to drive computing applications. In fact, designers and programmers want to predict the technological capabilities of future systems. For instance, Jim Gray's paper, "Rules of Thumb in Data Engineering," is an excellent example of how technology affects applications and vice versa. In addition, Moore's law indicates that processor speed doubles every 18 months. Although Moore's law has been proven valid over the last 30 years, it is difficult to say whether it will continue to be true in the future.

Gilder's law indicates that network bandwidth has doubled each year in the past. Will that trend continue in the future? The tremendous price/performance ratio of commodity

hardware was driven by the desktop, notebook, and tablet computing markets. This has also driven the adoption and use of commodity technologies in large-scale computing. We will discuss the future of these computing trends in more detail in subsequent chapters. For now, it's important to understand how distributed systems emphasize both resource distribution and concurrency or high *degree of parallelism* (*DoP*). Let's review the degrees of parallelism before we discuss the special requirements for distributed computing.

### 1.1.2.1 Degrees of Parallelism

Fifty years ago, when hardware was bulky and expensive, most computers were designed in a bit-serial fashion. In this scenario, *bit-level parallelism* (*BLP*) converts bit-serial processing to word-level processing gradually. Over the years, users graduated from 4-bit microprocessors to 8-, 16-, 32-, and 64-bit CPUs. This led us to the next wave of improvement, known as *instruction-level parallelism* (*ILP*), in which the processor executes multiple instructions simultaneously rather than only one instruction at a time. For the past 30 years, we have practiced ILP through pipelining, superscalar computing, *VLIW* (*very long instruction word*) architectures, and multithreading. ILP requires branch prediction, dynamic scheduling, speculation, and compiler support to work efficiently.

*Data-level parallelism* (*DLP*) was made popular through *SIMD* (*single instruction, multiple data*) and vector machines using vector or array types of instructions. DLP requires even more hardware support and compiler assistance to work properly. Ever since the introduction of multicore processors and *chip multiprocessors* (*CMPs*), we have been exploring *task-level parallelism* (*TLP*). A modern processor explores all of the aforementioned parallelism types. In fact, BLP, ILP, and DLP are well supported by advances in hardware and compilers. However, TLP is far from being very successful due to difficulty in programming and compilation of code for efficient execution on multicore CMPs. As we move from parallel processing to distributed processing, we will see an increase in computing granularity to *job-level parallelism* (*JLP*). It is fair to say that coarse-grain parallelism is built on top of fine-grain parallelism.

### 1.1.2.2 Innovative Applications

Both HPC and HTC systems desire transparency in many application aspects. For example, data access, resource allocation, process location, concurrency in execution, job replication, and failure recovery should be made transparent to both users and system management. Table 1.1 highlights a few key applications that have driven the development of parallel and distributed systems over the years. These applications spread across many important domains in science, engineering, business, education, health care, traffic control, Internet and web services, military, and government applications.

---

**Table 1.1**

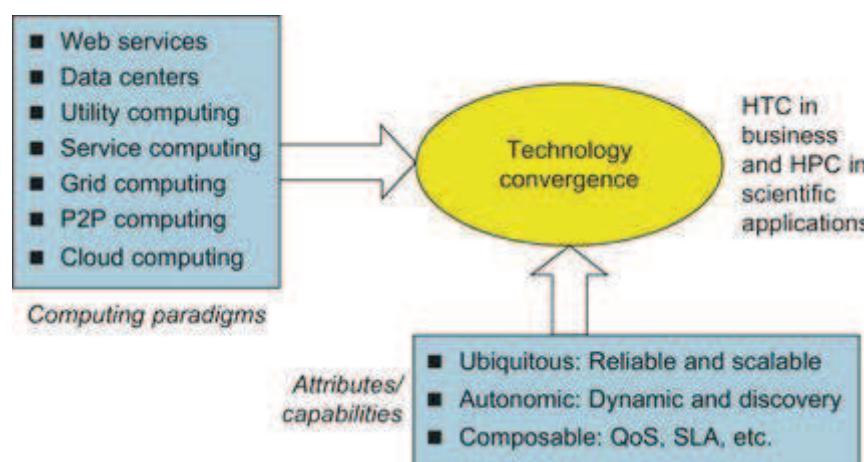
**Applications of High-Performance and High-Throughput Systems**

Domain	Specific Applications
Science and engineering	Scientific simulations, genomic analysis, etc. Earthquake prediction, global warming, weather forecasting, etc.
Business, education, services industry, and health care	Telecommunication, content delivery, e-commerce, etc. Banking, stock exchanges, transaction processing, etc. Air traffic control, electric power grids, distance education, etc. Health care, hospital automation, telemedicine, etc.
Internet and web services, and government applications	Internet search, data centers, decision-making systems, etc. Traffic monitoring, worm containment, cyber security, etc. Digital government, online tax return processing, social networking, etc.
Mission-critical applications	Military command and control, intelligent systems, crisis management, etc.

Almost all applications demand computing economics, web-scale data collection, system reliability, and scalable performance. For example, distributed transaction processing is often practiced in the banking and finance industry. Transactions represent 90 percent of the existing market for reliable banking systems. Users must deal with multiple database servers in distributed transactions. Maintaining the consistency of replicated transaction records is crucial in real-time banking services. Other complications include lack of software support, network saturation, and security threats in these applications. We will study applications and software support in more detail in subsequent chapters.

### 1.1.2.3 The Trend Toward Utility Computing

Figure 1.2 identifies major computing paradigms to facilitate the study of distributed systems and their applications. These paradigms share some common characteristics. First, they are all ubiquitous in daily life. Reliability and scalability are two major design objectives in these computing models. Second, they are aimed at autonomic operations that can be self-organized to support dynamic discovery. Finally, these paradigms are composable with QoS and SLAs (*service-level agreements*). These paradigms and their attributes realize the computer utility vision.

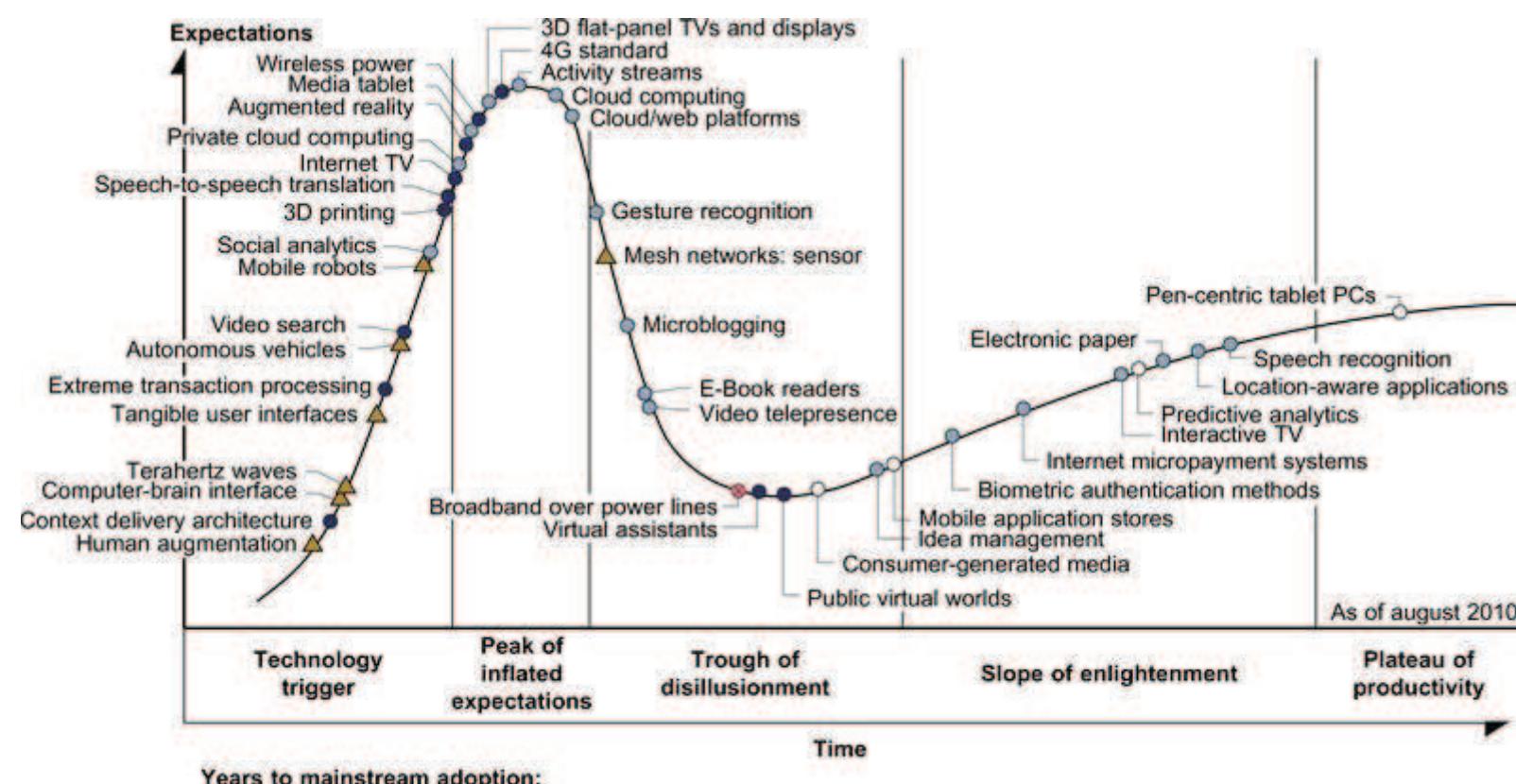


**FIGURE 1.2** The vision of computer utilities in modern distributed computing systems. Modified from presentation slide by Raj Buyya, 2010

*Utility computing* focuses on a business model in which customers receive computing resources from a paid service provider. All grid/cloud platforms are regarded as utility service providers. However, cloud computing offers a broader concept than utility computing. Distributed cloud applications run on any available servers in some edge networks. Major technological challenges include all aspects of computer science and engineering. For example, users demand new network-efficient processors, scalable memory and storage schemes, distributed OSes, middleware for machine virtualization, new programming models, effective resource management, and application program development. These hardware and software supports are necessary to build distributed systems that explore massive parallelism at all processing levels.

### 1.1.2.4 The Hype Cycle of New Technologies

Any new and emerging computing and information technology may go through a hype cycle, as illustrated in [Figure 1.3](#). This cycle shows the expectations for the technology at five different stages. The expectations rise sharply from the trigger period to a high peak of inflated expectations. Through a short period of disillusionment, the expectation may drop to a valley and then increase steadily over a long enlightenment period to a plateau of productivity. The number of years for an emerging technology to reach a certain stage is marked by special symbols. The hollow circles indicate technologies that will reach mainstream adoption in two years. The gray circles represent technologies that will reach mainstream adoption in two to five years. The solid circles represent those that require five to 10 years to reach mainstream adoption, and the triangles denote those that require more than 10 years. The crossed circles represent technologies that will become obsolete before they reach the plateau.



### FIGURE 1.3 Hype cycle for Emerging Technologies, 2010.

#### Hype Cycle Disclaimer

The Hype Cycle is copyrighted 2010 by Gartner, Inc. and its affiliates and is reused with permission. Hype Cycles are graphical representations of the relative maturity of technologies, IT methodologies and management disciplines. They are intended solely as a research tool, and not as a specific guide to action. Gartner disclaims all warranties, express or implied, with respect to this research, including any warranties of merchantability or fitness for a particular purpose.

This Hype Cycle graphic was published by Gartner, Inc. as part of a larger research note and should be evaluated in the context of the entire report. The Gartner report is available at <http://www.gartner.com/it/page.jsp?id=1447613>.

Source: Gartner Press Release "Gartner's 2010 Hype Cycle Special Report Evaluates Maturity of 1,800 Technologies" 7 October 2010.

The hype cycle in Figure 1.3 shows the technology status as of August 2010. For example, at that time *consumer-generated media* was at the disillusionment stage, and it was predicted to take less than two years to reach its plateau of adoption. *Internet micropayment systems* were forecast to take two to five years to move from the enlightenment stage to maturity. It was believed that *3D printing* would take five to 10 years to move from the rising expectation stage to mainstream adoption, and *mesh network sensors* were expected to take more than 10 years to move from the inflated expectation stage to a plateau of mainstream adoption.

Also as shown in Figure 1.3, the *cloud technology* had just crossed the peak of the expectation stage in 2010, and it was expected to take two to five more years to reach the productivity stage. However, *broadband over power line* technology was expected to become obsolete before leaving the valley of disillusionment stage in 2010. Many additional technologies (denoted by dark circles in Figure 1.3) were at their peak expectation stage in August 2010, and they were expected to take five to 10 years to reach their plateau of success. Once a technology begins to climb the slope of enlightenment, it may reach the productivity plateau within two to five years. Among these promising technologies are the clouds, biometric authentication, interactive TV, speech recognition, predictive analytics, and media tablets.

## 1.1.3 The Internet of Things and Cyber-Physical Systems

In this section, we will discuss two Internet development trends: the Internet of Things [48] and cyber-physical systems. These evolutionary trends emphasize the extension of the Internet to everyday objects. We will only cover the basics of these concepts here; we will discuss them in more detail in Chapter 9.

### 1.1.3.1 The Internet of Things

The traditional Internet connects machines to machines or web pages to web pages. The concept of the IoT was introduced in 1999 at MIT [40]. The IoT refers to the networked interconnection of everyday objects, tools, devices, or computers. One can view the IoT as a wireless network of sensors that interconnect all things in our daily life. These things can be large or small and they vary with respect to time and place. The idea is to tag every

object using RFID or a related sensor or electronic technology such as GPS.

With the introduction of the IPv6 protocol,  $2^{128}$  IP addresses are available to distinguish all the objects on Earth, including all computers and pervasive devices. The IoT researchers have estimated that every human being will be surrounded by 1,000 to 5,000 objects. The IoT needs to be designed to track 100 trillion static or moving objects simultaneously. The IoT demands universal addressability of all of the objects or things. To reduce the complexity of identification, search, and storage, one can set the threshold to filter out fine-grain objects. The IoT obviously extends the Internet and is more heavily developed in Asia and European countries.

In the IoT era, all objects and devices are instrumented, interconnected, and interacted with each other intelligently. This communication can be made between people and things or among the things themselves. Three communication patterns co-exist: namely H2H (human-to-human), H2T (human-to-thing), and T2T (thing-to-thing). Here things include machines such as PCs and mobile phones. The idea here is to connect things (including human and machine objects) at any time and any place intelligently with low cost. Any place connections include at the PC, indoor (away from PC), outdoors, and on the move. Any time connections include daytime, night, outdoors and indoors, and on the move as well.

The dynamic connections will grow exponentially into a new dynamic network of networks, called the *Internet of Things* (IoT). The IoT is still in its infancy stage of development. Many prototype IoTs with restricted areas of coverage are under experimentation at the time of this writing. Cloud computing researchers expect to use the cloud and future Internet technologies to support fast, efficient, and intelligent interactions among humans, machines, and any objects on Earth. A smart Earth should have intelligent cities, clean water, efficient power, convenient transportation, good food supplies, responsible banks, fast telecommunications, green IT, better schools, good health care, abundant resources, and so on. This dream living environment may take some time to reach fruition at different parts of the world.

### 1.1.3.2 Cyber-Physical Systems

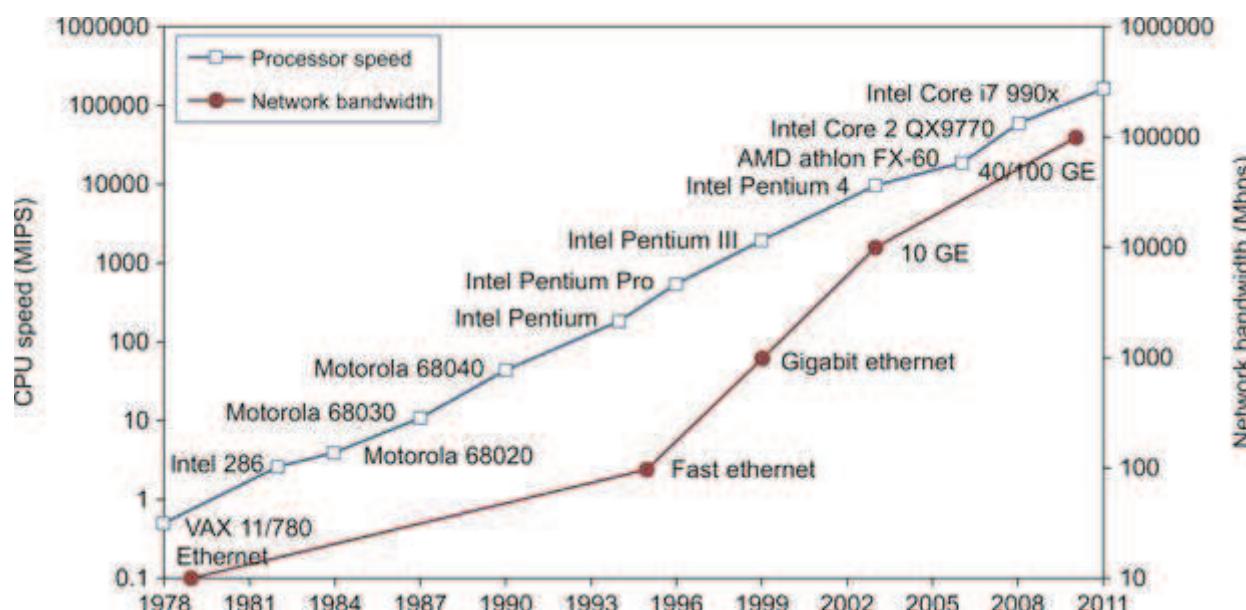
A *cyber-physical system* (CPS) is the result of interaction between computational processes and the physical world. A CPS integrates “cyber” (heterogeneous, asynchronous) with “physical” (concurrent and information-dense) objects. A CPS merges the “3C” technologies of *computation*, *communication*, and *control* into an intelligent closed feedback system between the physical world and the information world, a concept which is actively explored in the United States. The IoT emphasizes various networking connections among physical objects, while the CPS emphasizes exploration of *virtual reality* (VR) applications in the physical world. We may transform how we interact with the physical world just like the Internet transformed how we interact with the virtual world. We will study IoT, CPS, and their relationship to cloud computing in [Chapter 9](#).

## 1.2 Technologies for Network-Based Systems

With the concept of scalable computing under our belt, it's time to explore hardware, software, and network technologies for distributed computing system design and applications. In particular, we will focus on viable approaches to building distributed operating systems for handling massive parallelism in a distributed environment.

## 1.2.1 Multicore CPUs and Multithreading Technologies

Consider the growth of component and network technologies over the past 30 years. They are crucial to the development of HPC and HTC systems. In [Figure 1.4](#), processor speed is measured in *millions of instructions per second* (MIPS) and network bandwidth is measured in *megabits per second* (Mbps) or *gigabits per second* (Gbps). The unit GE refers to 1 Gbps Ethernet bandwidth.



**FIGURE 1.4** Improvement in processor and network technologies over 33 years. Courtesy of Xaosong Lou and Lizhong Chen of University of Southern California, 2011

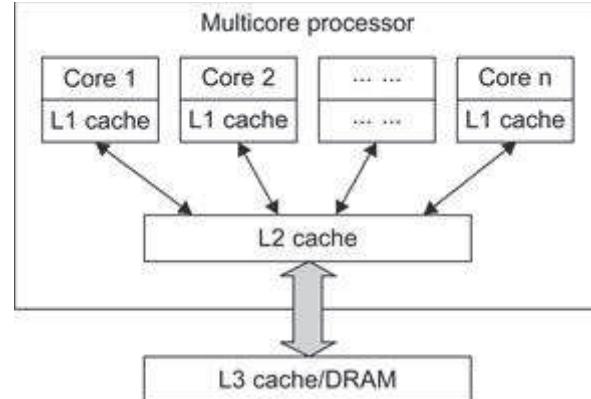
### 1.2.1.1 Advances in CPU Processors

Today, advanced CPUs or microprocessor chips assume a multicore architecture with dual, quad, six, or more processing cores. These processors exploit parallelism at ILP and TLP levels. Processor speed growth is plotted in the upper curve in [Figure 1.4](#) across generations of microprocessors or CMPs. We see growth from 1 MIPS for the VAX 780 in 1978 to 1,800 MIPS for the Intel Pentium 4 in 2002, up to a 22,000 MIPS peak for the Sun Niagara 2 in 2008. As the figure shows, Moore's law has proven to be pretty accurate in this case. The clock rate for these processors increased from 10 MHz for the Intel 286 to 4 GHz for the Pentium 4 in 30 years.

However, the clock rate reached its limit on CMOS-based chips due to power limitations. At the time of this writing, very few CPU chips run with a clock rate exceeding 5 GHz. In other words, clock rate will not continue to improve unless chip

technology matures. This limitation is attributed primarily to excessive heat generation with high frequency or high voltages. The ILP is highly exploited in modern CPU processors. ILP mechanisms include multiple-issue superscalar architecture, dynamic branch prediction, and speculative execution, among others. These ILP techniques demand hardware and compiler support. In addition, DLP and TLP are highly explored in *graphics processing units* (GPUs) that adopt a many-core architecture with hundreds to thousands of simple cores.

Both multi-core CPU and many-core GPU processors can handle multiple instruction threads at different magnitudes today. [Figure 1.5](#) shows the architecture of a typical multicore processor. Each core is essentially a processor with its own private cache (L1 cache). Multiple cores are housed in the same chip with an L2 cache that is shared by all cores. In the future, multiple CMPs could be built on the same CPU chip with even the L3 cache on the chip. Multicore and multithreaded CPUs are equipped with many high-end processors, including the Intel i7, Xeon, AMD Opteron, Sun Niagara, IBM Power 6, and X cell processors. Each core could be also multithreaded. For example, the Niagara II is built with eight cores with eight threads handled by each core. This implies that the maximum ILP and TLP that can be exploited in Niagara is 64 ( $8 \times 8 = 64$ ). In 2011, the Intel Core i7 990x has reported 159,000 MIPS execution rate as shown in the upper-most square in [Figure 1.4](#).



**FIGURE 1.5** Schematic of a modern multicore CPU chip using a hierarchy of caches, where L1 cache is private to each core, on-chip L2 cache is shared and L3 cache or DRAM is off the chip.

### 1.2.1.2 Multicore CPU and Many-Core GPU Architectures

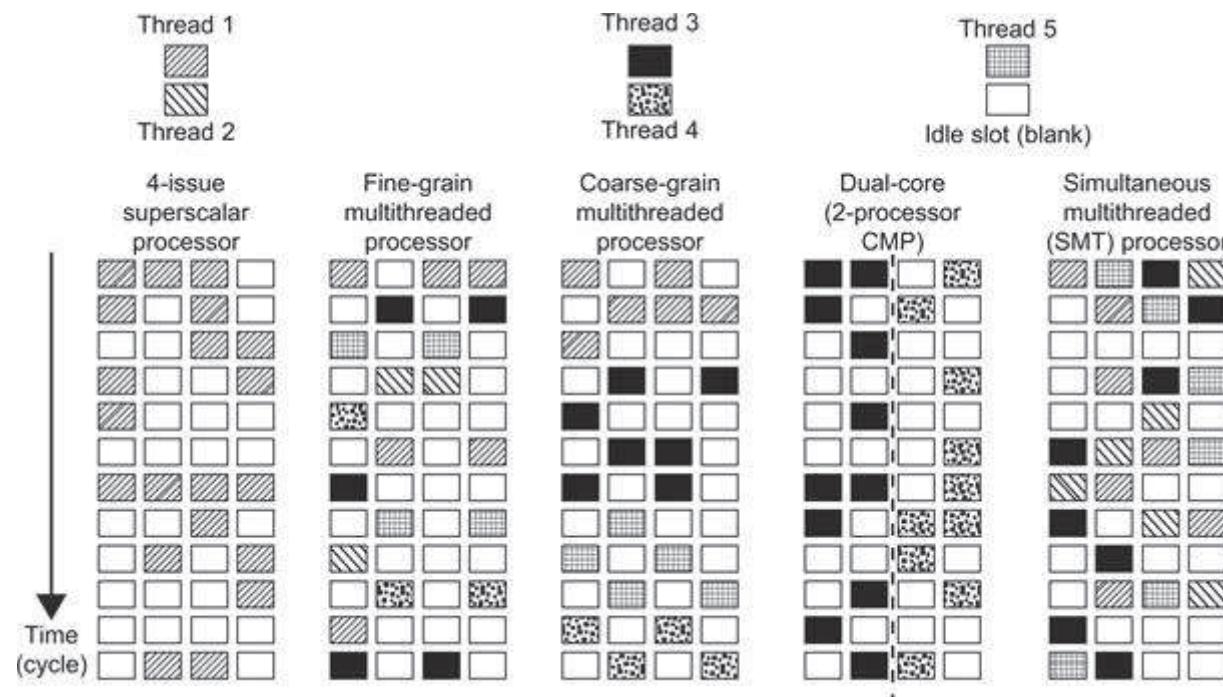
Multicore CPUs may increase from the tens of cores to hundreds or more in the future. But the CPU has reached its limit in terms of exploiting massive DLP due to the aforementioned memory wall problem. This has triggered the development of many-core GPUs with hundreds or more thin cores. Both IA-32 and IA-64 instruction set architectures are built into commercial CPUs. Now, x-86 processors have been extended to serve HPC and HTC systems in some high-end server processors.

Many RISC processors have been replaced with multicore x-86 processors and many-core GPUs in the Top 500 systems. This trend indicates that x-86 upgrades will dominate

in data centers and supercomputers. The GPU also has been applied in large clusters to build supercomputers in MPPs. In the future, the processor industry is also keen to develop asymmetric or heterogeneous chip multiprocessors that can house both fat CPU cores and thin GPU cores on the same chip.

### 1.2.1.3 Multithreading Technology

Consider in [Figure 1.6](#) the dispatch of five independent threads of instructions to four pipelined data paths (functional units) in each of the following five processor categories, from left to right: a *four-issue superscalar processor*, a *fine-grain multithreaded processor*, a *coarse-grain multithreaded processor*, a *two-core CMP*, and a *simultaneous multithreaded (SMT)* processor. The superscalar processor is single-threaded with four functional units. Each of the three multithreaded processors is four-way multithreaded over four functional data paths. In the dual-core processor, assume two processing cores, each a single-threaded two-way superscalar processor.



**FIGURE 1.6** Five micro-architectures in modern CPU processors, that exploit ILP and TLP supported by multicore and multithreading technologies.

Instructions from different threads are distinguished by specific shading patterns for instructions from five independent threads. Typical instruction scheduling patterns are shown here. Only instructions from the same thread are executed in a superscalar processor. Fine-grain multithreading switches the execution of instructions from different threads per cycle. Course-grain multithreading executes many instructions from the same thread for quite a few cycles before switching to another thread. The multicore CMP executes instructions from different threads completely. The SMT allows simultaneous scheduling of instructions from different threads in the same cycle.

These execution patterns closely mimic an ordinary program. The blank squares

correspond to no available instructions for an instruction data path at a particular processor cycle. More blank cells imply lower scheduling efficiency. The maximum ILP or maximum TLP is difficult to achieve at each processor cycle. The point here is to demonstrate your understanding of typical instruction scheduling patterns in these five different micro-architectures in modern processors.

## 1.2.2 GPU Computing to Exascale and Beyond

A GPU is a graphics coprocessor or accelerator mounted on a computer's graphics card or video card. A GPU offloads the CPU from tedious graphics tasks in video editing applications. The world's first GPU, the GeForce 256, was marketed by NVIDIA in 1999. These GPU chips can process a minimum of 10 million polygons per second, and are used in nearly every computer on the market today. Some GPU features were also integrated into certain CPUs. Traditional CPUs are structured with only a few cores. For example, the Xeon X5670 CPU has six cores. However, a modern GPU chip can be built with hundreds of processing cores.

Unlike CPUs, GPUs have a throughput architecture that exploits massive parallelism by executing many concurrent threads slowly, instead of executing a single long thread in a conventional microprocessor very quickly. Lately, parallel GPUs or GPU clusters have been garnering a lot of attention against the use of CPUs with limited parallelism. *General-purpose computing on GPUs*, known as GPGPUs, have appeared in the HPC field. NVIDIA's CUDA model was for HPC using GPGPUs. [Chapter 2](#) will discuss GPU clusters for massively parallel computing in more detail [15,32].

### 1.2.2.1 How GPUs Work

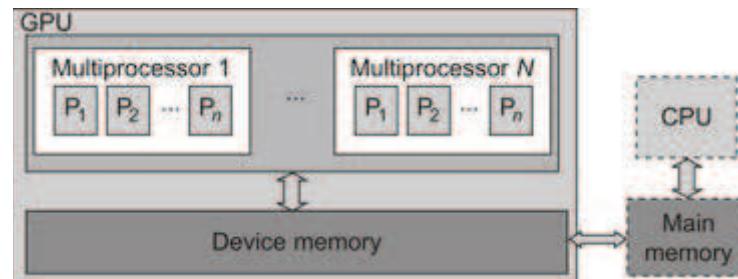
Early GPUs functioned as coprocessors attached to the CPU. Today, the NVIDIA GPU has been upgraded to 128 cores on a single chip. Furthermore, each core on a GPU can handle eight threads of instructions. This translates to having up to 1,024 threads executed concurrently on a single GPU. This is true massive parallelism, compared to only a few threads that can be handled by a conventional CPU. The CPU is optimized for latency caches, while the GPU is optimized to deliver much higher throughput with explicit management of on-chip memory.

Modern GPUs are not restricted to accelerated graphics or video coding. They are used in HPC systems to power supercomputers with massive parallelism at multicore and multithreading levels. GPUs are designed to handle large numbers of floating-point operations in parallel. In a way, the GPU offloads the CPU from all data-intensive calculations, not just those that are related to video processing. Conventional GPUs are widely used in mobile phones, game consoles, embedded systems, PCs, and servers. The NVIDIA CUDA Tesla or Fermi is used in GPU clusters or in HPC systems for parallel processing of massive floating-pointing data.

### 1.2.2.2 GPU Programming Model

Figure 1.7 shows the interaction between a CPU and GPU in performing parallel

execution of floating-point operations concurrently. The CPU is the conventional multicore processor with limited parallelism to exploit. The GPU has a many-core architecture that has hundreds of simple processing cores organized as multiprocessors. Each core can have one or more threads. Essentially, the CPU's floating-point kernel computation role is largely offloaded to the many-core GPU. The CPU instructs the GPU to perform massive data processing. The bandwidth must be matched between the on-board main memory and the on-chip GPU memory. This process is carried out in NVIDIA's CUDA programming using the GeForce 8800 or Tesla and Fermi GPUs. We will study the use of CUDA GPUs in large-scale cluster computing in [Chapter 2](#).

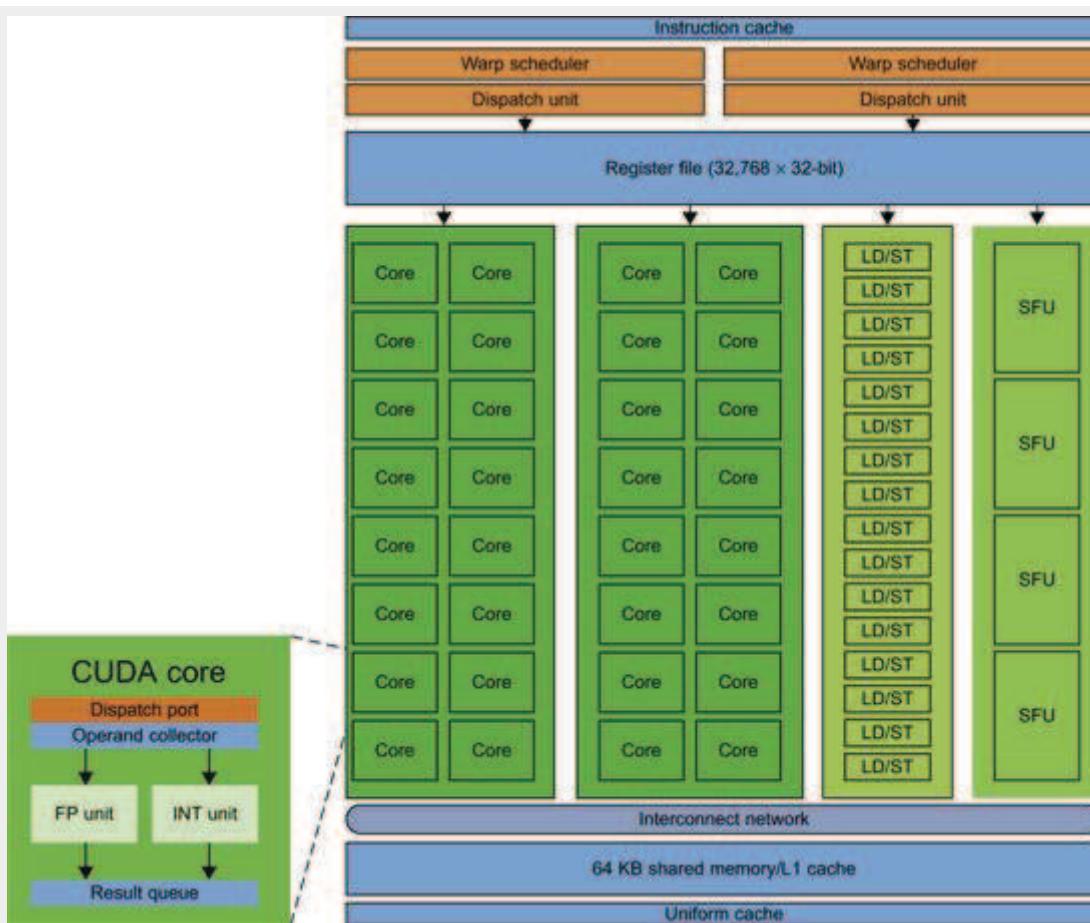


**FIGURE 1.7** The use of a GPU along with a CPU for massively parallel execution in hundreds or thousands of processing cores. Courtesy of B. He, et al., PACT'08 [23]

### Example 1.1

#### The NVIDIA Fermi GPU Chip with 512 CUDA Cores

In November 2010, three of the five fastest supercomputers in the world (the Tianhe-1a, Nebulae, and Tsubame) used large numbers of GPU chips to accelerate floating-point computations. [Figure 1.8](#) shows the architecture of the Fermi GPU, a next-generation GPU from NVIDIA. This is a *streaming multiprocessor (SM)* module. Multiple SMs can be built on a single GPU chip. The Fermi chip has 16 SMs implemented with 3 billion transistors. Each SM comprises up to 512 *streaming processors (SPs)*, known as *CUDA cores*. The Tesla GPUs used in the Tianhe-1a have a similar architecture, with 448 CUDA cores.



**FIGURE 1.8** NVIDIA Fermi GPU built with 16 streaming multiprocessors (SMs) of 32 CUDA cores each; only one SM is shown. More details can be found also in [49]. Courtesy of NVIDIA, 2009 [36] 2011

The Fermi GPU is a newer generation of GPU, first appearing in 2011. The Tesla or Fermi GPU can be used in desktop workstations to accelerate floating-point calculations or for building large-scale data centers. The architecture shown is based on a 2009 white paper by NVIDIA [36]. There are 32 CUDA cores per SM. Only one SM is shown in Figure 1.8. Each CUDA core has a simple pipelined integer ALU and an FPU that can be used in parallel. Each SM has 16 load/store units allowing source and destination addresses to be calculated for 16 threads per clock. There are four *special function units* (SFUs) for executing transcendental instructions.

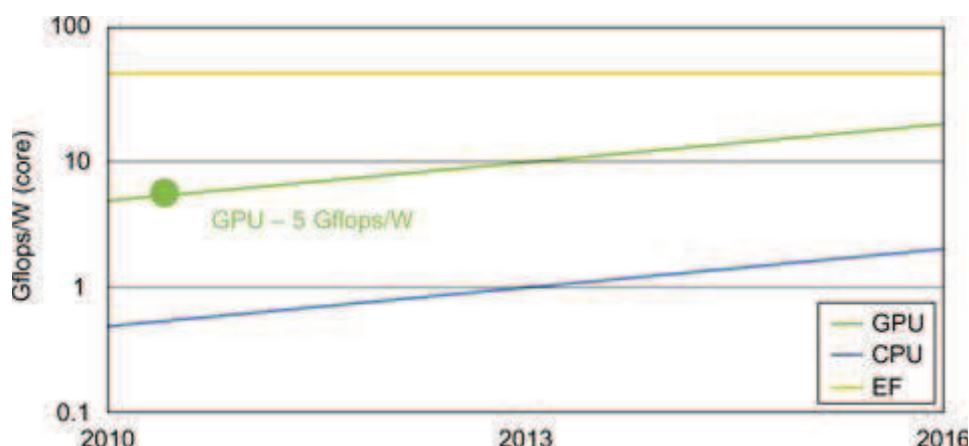
All functional units and CUDA cores are interconnected by an NoC (*network on chip*) to a large number of SRAM banks (L2 caches). Each SM has a 64 KB L1 cache. The 768 KB unified L2 cache is shared by all SMs and serves all load, store, and texture operations. *Memory controllers* are used to connect to 6 GB of off-chip DRAMs. The SM schedules threads in groups of 32 parallel threads called *warps*. In total, 256/512 *FMA* (*fused multiply and add*) operations can be done in parallel to produce 32/64-bit floating-point results. The 512 CUDA cores in an SM can work in parallel to deliver up to 515 Gflops of double-precision results, if fully utilized. With 16 SMs, a single GPU has a peak speed of 82.4 Tflops. Only 12 Fermi GPUs have the potential to reach the Pflops performance.

In the future, thousand-core GPUs may appear in Exascale (Eflops or  $10^{18}$  flops) systems. This reflects a trend toward building future MPPs with hybrid architectures of both types of processing chips. In a DARPA report published in September 2008, four challenges are identified for exascale computing: (1) energy and power, (2) memory and storage, (3) concurrency and locality, and (4) system resiliency. Here, we see the progress of GPUs along with CPU advances in power efficiency, performance, and programmability [16]. In [Chapter 2](#), we will discuss the use of GPUs to build large clusters.

### 1.2.2.3 Power Efficiency of the GPU

Bill Dally of Stanford University considers power and massive parallelism as the major benefits of GPUs over CPUs for the future. By extrapolating current technology and computer architecture, it was estimated that 60 Gflops/watt per core is needed to run an exaflops system (see [Figure 1.10](#)). Power constrains what we can put in a CPU or GPU chip. Dally has estimated that the CPU chip consumes about 2 nJ/instruction, while the GPU chip requires 200 pJ/instruction, which is 1/10 less than that of the CPU. The CPU is optimized for latency in caches and memory, while the GPU is optimized for throughput with explicit management of on-chip memory.

[Figure 1.9](#) compares the CPU and GPU in their performance/power ratio measured in Gflops/watt per core. In 2010, the GPU had a value of 5 Gflops/watt at the core level, compared with less than 1 Gflop/watt per CPU core. This may limit the scaling of future supercomputers. However, the GPUs may close the gap with the CPUs. Data movement dominates power consumption. One needs to optimize the storage hierarchy and tailor the memory to the applications. We need to promote self-aware OS and runtime support and build locality-aware compilers and auto-tuners for GPU-based MPPs. This implies that both power and software are the real challenges in future parallel and distributed computing systems.



**FIGURE 1.9** The GPU performance (middle line, measured 5 Gflops/W/core in 2011), compared with the lower CPU performance (lower line measured 0.8 Gflops/W/core in 2011) and the estimated 60 Gflops/W/core performance in 2011 for the Exascale (EF in upper curve) in the future. Courtesy of Bill Dally [15]

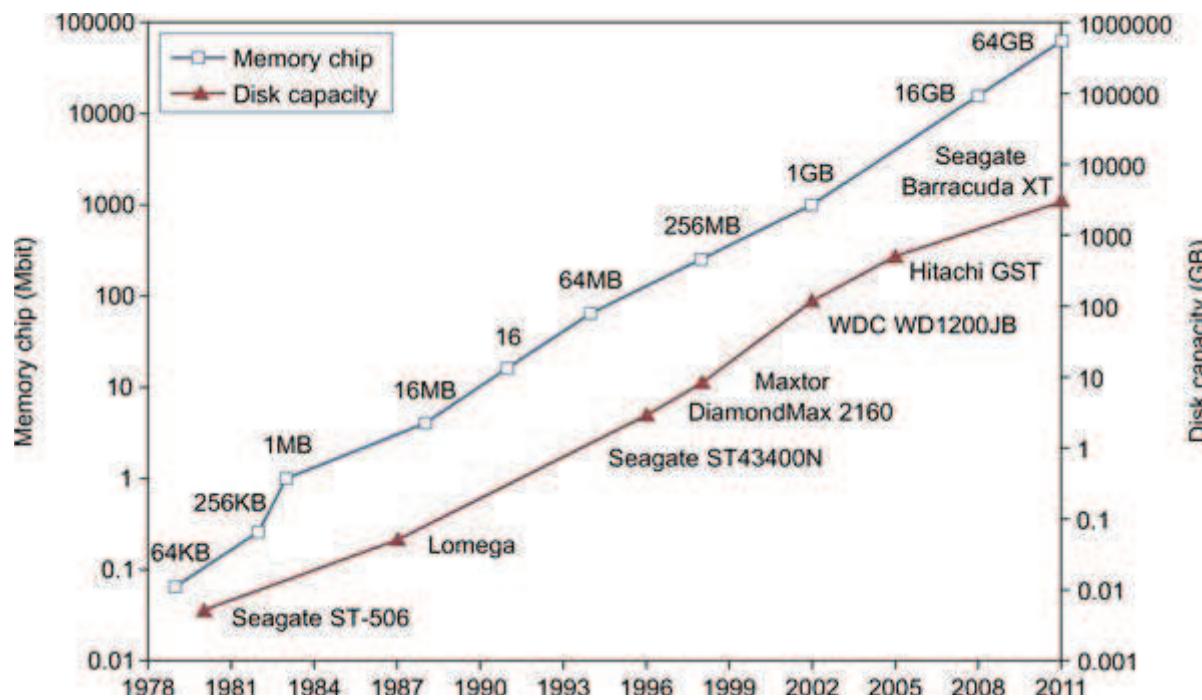
## 1.2.3 Memory, Storage, and Wide-Area Networking

### 1.2.3.1 Memory Technology

The upper curve in Figure 1.10 plots the growth of DRAM chip capacity from 16 KB in 1976 to 64 GB in 2011. This shows that memory chips have experienced a 4x increase in capacity every three years. Memory access time did not improve much in the past. In fact, the memory wall problem is getting worse as the processor gets faster. For hard drives, capacity increased from 260 MB in 1981 to 250 GB in 2004. The Seagate Barracuda XT hard drive reached 3 TB in 2011. This represents an approximately 10x increase in capacity every eight years. The capacity increase of disk arrays will be even greater in the years to come. Faster processor speed and larger memory capacity result in a wider gap between processors and memory. The memory wall may become even worse a problem limiting the CPU performance in the future.

### 1.2.3.2 Disks and Storage Technology

Beyond 2011, disks or disk arrays have exceeded 3 TB in capacity. The lower curve in Figure 1.10 shows the disk storage growth in 7 orders of magnitude in 33 years. The rapid growth of flash memory and *solid-state drives* (SSDs) also impacts the future of HPC and HTC systems. The mortality rate of SSD is not bad at all. A typical SSD can handle 300,000 to 1 million write cycles per block. So the SSD can last for several years, even under conditions of heavy write usage. Flash and SSD will demonstrate impressive speedups in many applications.



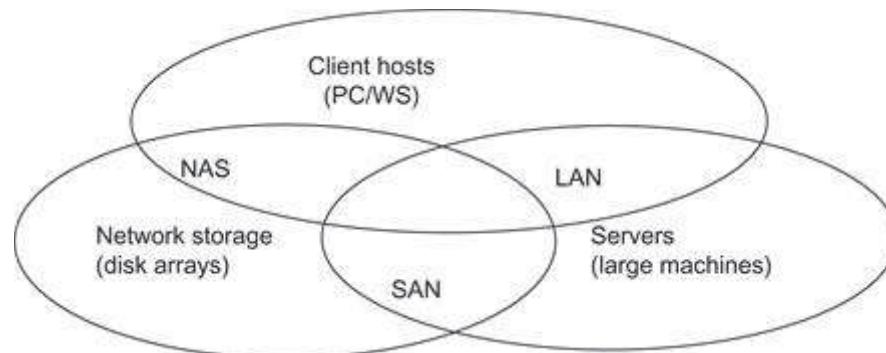
**FIGURE 1.10** Improvement in memory and disk technologies over 33 years. The Seagate Barracuda XT disk has a capacity of 3 TB in 2011. Courtesy of Xiaosong Lou and Lizhong Chen of University of Southern California, 2011

Eventually, power consumption, cooling, and packaging will limit large system

development. Power increases linearly with respect to clock frequency and quadratic ally with respect to voltage applied on chips. Clock rate cannot be increased indefinitely. Lowered voltage supplies are very much in demand. Jim Gray once said in an invited talk at the University of Southern California, “*Tape units are dead, disks are tape units, flashes are disks, and memory are caches now.*” This clearly paints the future for disk and storage technology. In 2011, the SSDs are still too expensive to replace stable disk arrays in the storage market.

### 1.2.3.3 System-Area Interconnects

The nodes in small clusters are mostly interconnected by an Ethernet switch or a *local area network* (LAN). As Figure 1.11 shows, a LAN typically is used to connect client hosts to big servers. A *storage area network* (SAN) connects servers to network storage such as disk arrays. *Network attached storage* (NAS) connects client hosts directly to the disk arrays. All three types of networks often appear in a large cluster built with commercial network components. If no large distributed storage is shared, a small cluster could be built with a multiport Gigabit Ethernet switch plus copper cables to link the end machines. All three types of networks are commercially available.



**FIGURE 1.11** Three interconnection networks for connecting servers, client hosts, and storage devices; the LAN connects client hosts and servers, the SAN connects servers with disk arrays, and the NAS connects clients with large storage systems in the network environment.

### 1.2.3.4 Wide-Area Networking

The lower curve in Figure 1.10 plots the rapid growth of Ethernet bandwidth from 10 Mbps in 1979 to 1 Gbps in 1999, and 40 ~ 100 GE in 2011. It has been speculated that 1 Tbps network links will become available by 2013. According to Berman, Fox, and Hey [6], network links with 1,000, 1,000, 100, 10, and 1 Gbps bandwidths were reported, respectively, for international, national, organization, optical desktop, and copper desktop connections in 2006.

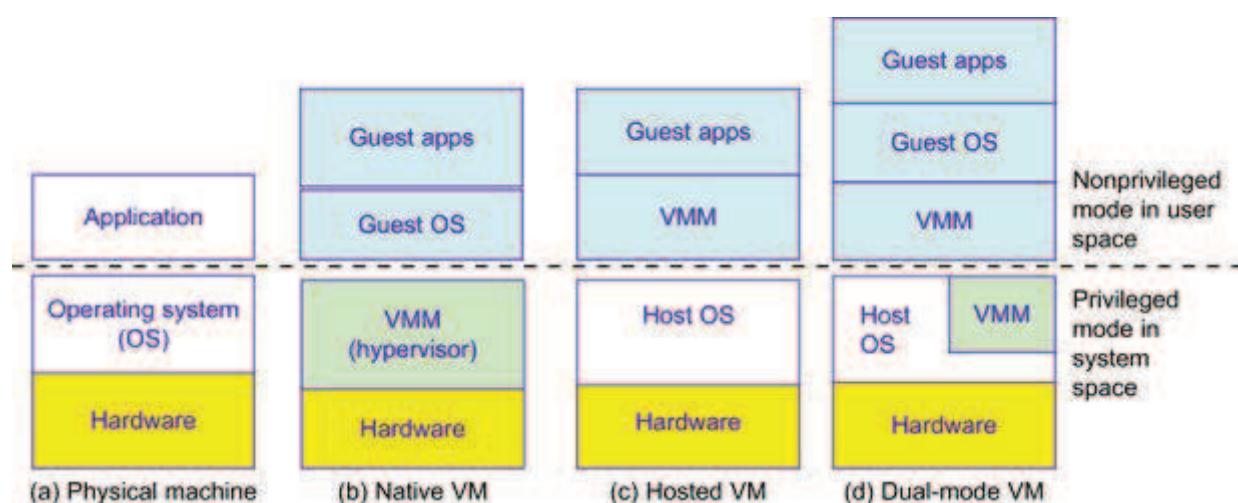
An increase factor of two per year on network performance was reported, which is faster than Moore’s law on CPU speed doubling every 18 months. The implication is that more computers will be used concurrently in the future. High-bandwidth networking increases the capability of building massively distributed systems. The IDC 2010 report

predicted that both InfiniBand and Ethernet will be the two major interconnect choices in the HPC arena. Most data centers are using Gigabit Ethernet as the interconnect in their server clusters.

## 1.2.4 Virtual Machines and Virtualization Middleware

A conventional computer has a single OS image. This offers a rigid architecture that tightly couples application software to a specific hardware platform. Some software running well on one machine may not be executable on another platform with a different instruction set under a fixed OS. *Virtual machines* (VMs) offer novel solutions to underutilized resources, application inflexibility, software manageability, and security concerns in existing physical machines.

Today, to build large clusters, grids, and clouds, we need to access large amounts of computing, storage, and networking resources in a virtualized manner. We need to aggregate those resources, and hopefully, offer a single system image. In particular, a cloud of provisioned resources must rely on virtualization of processors, memory, and I/O facilities dynamically. We will cover virtualization in [Chapter 3](#). However, the basic concepts of virtualized resources, such as VMs, virtual storage, and virtual networking and their virtualization software or middleware, need to be introduced first. [Figure 1.12](#) illustrates the architectures of three VM configurations.



**FIGURE 1.12** Three VM architectures in (b), (c), and (d), compared with the traditional physical machine shown in (a). Courtesy of M. Abde-Majeed and S. Kulkarni, 2009 USC

### 1.2.4.1 Virtual Machines

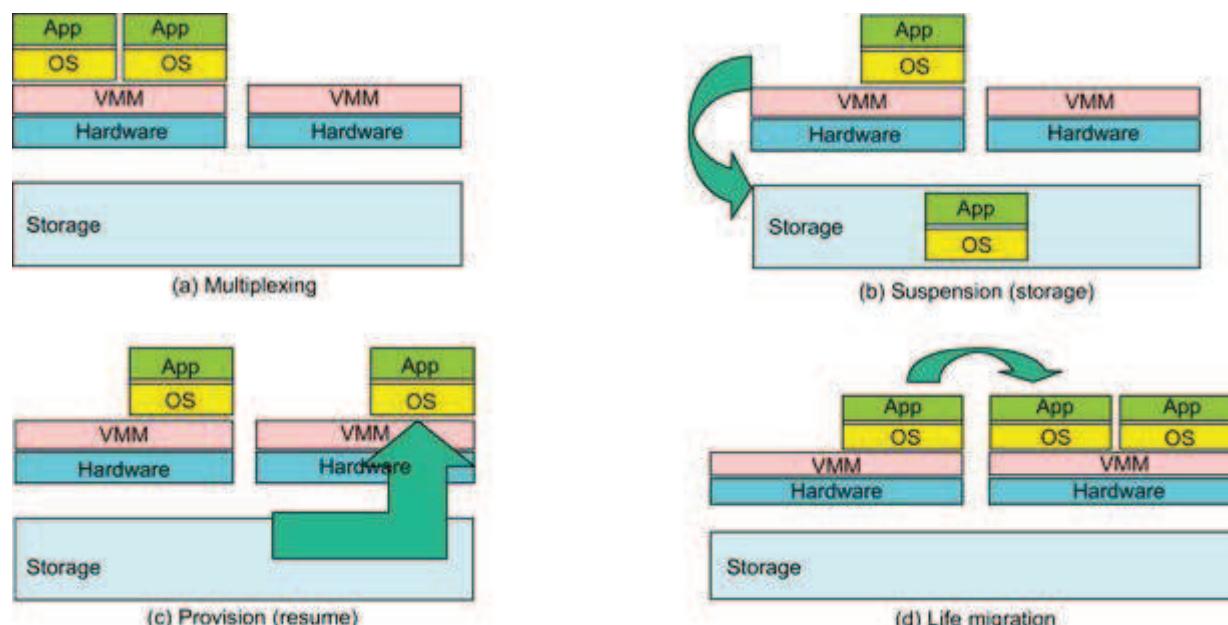
In [Figure 1.12](#), the host machine is equipped with the physical hardware, as shown at the bottom of the figure. An example is an x-86 architecture desktop running its installed Windows OS, as shown in part (a) of the figure. The VM can be provisioned for any hardware system. The VM is built with virtual resources managed by a guest OS to run a specific application. Between the VMs and the host platform, one needs to deploy a

middleware layer called a *virtual machine monitor* (VMM). Figure 1.12(b) shows a native VM installed with the use of a VMM called a *hypervisor* in privileged mode. For example, the hardware has x-86 architecture running the Windows system.

The guest OS could be a Linux system and the hypervisor is the XEN system developed at Cambridge University. This hypervisor approach is also called *bare-metal VM*, because the hypervisor handles the bare hardware (CPU, memory, and I/O) directly. Another architecture is the host VM shown in Figure 1.12(c). Here the VMM runs in nonprivileged mode. The host OS need not be modified. The VM can also be implemented with a dual mode, as shown in Figure 1.12(d). Part of the VMM runs at the user level and another part runs at the supervisor level. In this case, the host OS may have to be modified to some extent. Multiple VMs can be ported to a given hardware system to support the virtualization process. The VM approach offers hardware independence of the OS and applications. The user application running on its dedicated OS could be bundled together as a *virtual appliance* that can be ported to any hardware platform. The VM could run on an OS different from that of the host computer.

### 1.2.4.2 VM Primitive Operations

The VMM provides the VM abstraction to the guest OS. With full virtualization, the VMM exports a VM abstraction identical to the physical machine so that a standard OS such as Windows 2000 or Linux can run just as it would on the physical hardware. Low-level VMM operations are indicated by Mendel Rosenblum [41] and illustrated in Figure 1.13.



**FIGURE 1.13** VM multiplexing, suspension, provision, and migration in a distributed computing environment.  
Courtesy of M. Rosenblum, Keynote address, ACM ASPLOS 2006 [41]

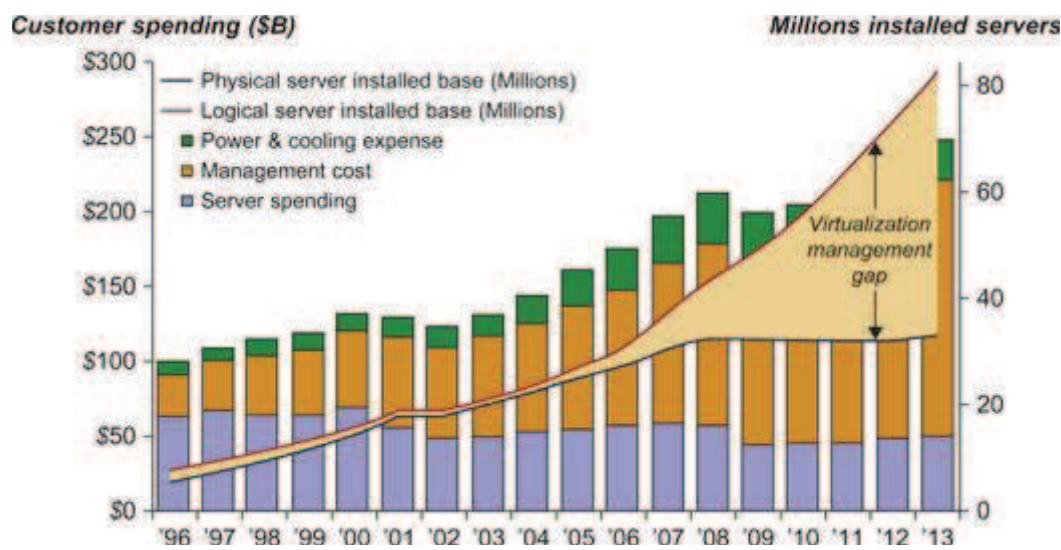
- First, the VMs can be multiplexed between hardware machines, as shown in Figure 1.13(a).
- Second, a VM can be suspended and stored in stable storage, as shown in Figure 1.13(b).

- Third, a suspended VM can be resumed or provisioned to a new hardware platform, as shown in [Figure 1.13\(c\)](#).
- Finally, a VM can be migrated from one hardware platform to another, as shown in [Figure 1.13\(d\)](#).

These VM operations enable a VM to be provisioned to any available hardware platform. They also enable flexibility in porting distributed application executions. Furthermore, the VM approach will significantly enhance the utilization of server resources. Multiple server functions can be consolidated on the same hardware platform to achieve higher system efficiency. This will eliminate server sprawl via deployment of systems as VMs, which move transparency to the shared hardware. With this approach, VMware claimed that server utilization could be increased from its current 5–15 percent to 60–80 percent.

### 1.2.4.3 Virtual Infrastructures

Physical resources for compute, storage, and networking at the bottom of [Figure 1.14](#) are mapped to the needy applications embedded in various VMs at the top. Hardware and software are then separated. Virtual infrastructure is what connects resources to distributed applications. It is a dynamic mapping of system resources to specific applications. The result is decreased costs and increased efficiency and responsiveness. Virtualization for server consolidation and containment is a good example of this. We will discuss VMs and virtualization support in [Chapter 3](#). Virtualization support for clusters, clouds, and grids is covered in [Chapters 3, 4, and 7](#), respectively.



**FIGURE 1.14** Growth and cost breakdown of data centers over the years. Source: IDC Report, 2009

### 1.2.5 Data Center Virtualization for Cloud Computing

In this section, we discuss basic architecture and design considerations of data centers. Cloud architecture is built with commodity hardware and network devices. Almost all

cloud platforms choose the popular x86 processors. Low-cost terabyte disks and Gigabit Ethernet are used to build data centers. Data center design emphasizes the performance/price ratio over speed performance alone. In other words, storage and energy efficiency are more important than sheer speed performance. [Figure 1.13](#) shows the server growth and cost breakdown of data centers over the past 15 years. Worldwide, about 43 million servers are in use as of 2010. The cost of utilities exceeds the cost of hardware after three years.

### **1.2.5.1 Data Center Growth and Cost Breakdown**

A large data center may be built with thousands of servers. Smaller data centers are typically built with hundreds of servers. The cost to build and maintain data center servers has increased over the years. According to a 2009 IDC report (see [Figure 1.14](#)), typically only 30 percent of data center costs goes toward purchasing IT equipment (such as servers and disks), 33 percent is attributed to the chiller, 18 percent to the *uninterruptible power supply (UPS)*, 9 percent to *computer room air conditioning (CRAC)*, and the remaining 7 percent to power distribution, lighting, and transformer costs. Thus, about 60 percent of the cost to run a data center is allocated to management and maintenance. The server purchase cost did not increase much with time. The cost of electricity and cooling did increase from 5 percent to 14 percent in 15 years.

### **1.2.5.2 Low-Cost Design Philosophy**

High-end switches or routers may be too cost-prohibitive for building data centers. Thus, using high-bandwidth networks may not fit the economics of cloud computing. Given a fixed budget, commodity switches and networks are more desirable in data centers. Similarly, using commodity x86 servers is more desired over expensive mainframes. The software layer handles network traffic balancing, fault tolerance, and expandability. Currently, nearly all cloud computing data centers use Ethernet as their fundamental network technology.

### **1.2.5.3 Convergence of Technologies**

Essentially, cloud computing is enabled by the convergence of technologies in four areas: (1) hardware virtualization and multi-core chips, (2) utility and grid computing, (3) SOA, Web 2.0, and WS mashups, and (4) autonomic computing and data center automation. Hardware virtualization and multicore chips enable the existence of dynamic configurations in the cloud. Utility and grid computing technologies lay the necessary foundation for computing clouds. Recent advances in SOA, Web 2.0, and mashups of platforms are pushing the cloud another step forward. Finally, achievements in autonomic computing and automated data center operations contribute to the rise of cloud computing.

Jim Gray once posted the following question: “*Science faces a data deluge. How to manage and analyze information?*” This implies that science and our society face the same challenge of data deluge. Data comes from sensors, lab experiments, simulations,

individual archives, and the web in all scales and formats. Preservation, movement, and access of massive data sets require generic tools supporting high-performance, scalable file systems, databases, algorithms, workflows, and visualization. With science becoming data-centric, a new paradigm of scientific discovery is becoming based on data-intensive technologies.

On January 11, 2007, the *Computer Science and Telecommunication Board* (CSTB) recommended fostering tools for data capture, data creation, and data analysis. A cycle of interaction exists among four technical areas. First, cloud technology is driven by a surge of interest in data deluge. Also, cloud computing impacts e-science greatly, which explores multicore and parallel computing technologies. These two hot areas enable the buildup of data deluge. To support data-intensive computing, one needs to address workflows, databases, algorithms, and virtualization issues.

By linking computer science and technologies with scientists, a spectrum of e-science or e-research applications in biology, chemistry, physics, the social sciences, and the humanities has generated new insights from interdisciplinary activities. Cloud computing is a transformative approach as it promises much more than a data center model. It fundamentally changes how we interact with information. The cloud provides services on demand at the infrastructure, platform, or software level. At the platform level, MapReduce offers a new programming model that transparently handles data parallelism with natural fault tolerance capability. We will discuss MapReduce in more detail in [Chapter 6](#).

Iterative MapReduce extends MapReduce to support a broader range of data mining algorithms commonly used in scientific applications. The cloud runs on an extremely large cluster of commodity computers. Internal to each cluster node, multithreading is practiced with a large number of cores in many-core GPU clusters. Data-intensive science, cloud computing, and multicore computing are converging and revolutionizing the next generation of computing in architectural design and programming challenges. They enable the pipeline: Data becomes information and knowledge, and in turn becomes machine wisdom as desired in SOA.

## 1.3 System Models for Distributed and Cloud Computing

Distributed and cloud computing systems are built over a large number of autonomous computer nodes. These node machines are interconnected by SANs, LANs, or WANs in a hierarchical manner. With today's networking technology, a few LAN switches can easily connect hundreds of machines as a working cluster. A WAN can connect many local clusters to form a very large cluster of clusters. In this sense, one can build a massive system with millions of computers connected to edge networks.

Massive systems are considered highly scalable, and can reach web-scale connectivity, either physically or logically. In [Table 1.2](#), massive systems are classified into four groups: *clusters*, *P2P networks*, *computing grids*, and *Internet clouds* over huge data centers. In terms of node number, these four system classes may involve hundreds, thousands, or even millions of computers as participating nodes. These machines work collectively,

cooperatively, or collaboratively at various levels. The table entries characterize these four system classes in various technical and application aspects.

**Table 1.2**

### Classification of Parallel and Distributed Computing Systems

Functionality, Applications	Computer Clusters [10,28,38]	Peer-to-Peer Networks [34,46]	Data/Computational Grids [6,18,51]	Cloud Platforms [1,9,11,12,30]
Architecture, Network Connectivity, and Size	Network of compute nodes interconnected by SAN, LAN, or WAN hierarchically	Flexible network of client machines logically connected by an overlay network	Heterogeneous clusters interconnected by high-speed network links over selected resource sites	Virtualized cluster of servers over data centers via SLA
Control and Resources Management	Homogeneous nodes with distributed control, running UNIX or Linux	Autonomous client nodes, free in and out, with self-organization	Centralized control, server-oriented with authenticated security	Dynamic resource provisioning of servers, storage, and networks
Applications and Network-centric Services	High-performance computing, search engines, and web services, etc.	Most appealing to business file sharing, content delivery, and social networking	Distributed supercomputing, global problem solving, and data center services	Upgraded web search, utility computing, and outsourced computing services
Representative Operational Systems	Google search engine, SunBlade, IBM Road Runner, Cray XT4, etc.	Gnutella, eMule, BitTorrent, Napster, KaZaA, Skype, JXTA	TeraGrid, GriPhyN, UK EGEE, D-Grid, ChinaGrid, etc.	Google App Engine, IBM Bluecloud, AWS, and Microsoft Azure

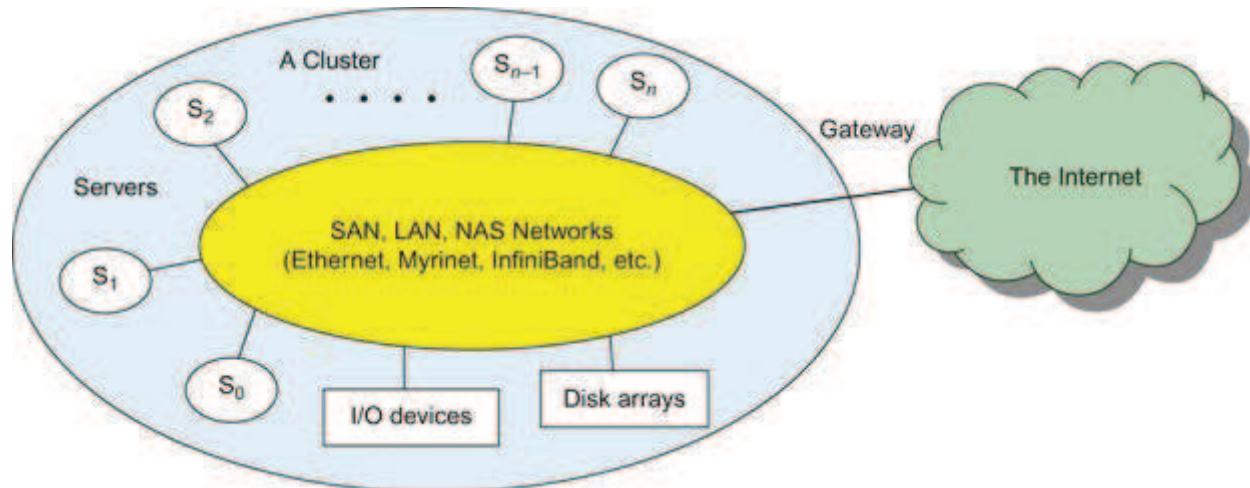
From the application perspective, clusters are most popular in supercomputing applications. In 2009, 417 of the Top 500 supercomputers were built with cluster architecture. It is fair to say that clusters have laid the necessary foundation for building large-scale grids and clouds. P2P networks appeal most to business applications. However, the content industry was reluctant to accept P2P technology for lack of copyright protection in ad hoc networks. Many national grids built in the past decade were underutilized for lack of reliable middleware or well-coded applications. Potential advantages of cloud computing include its low cost and simplicity for both providers and users.

#### 1.3.1 Clusters of Cooperative Computers

A computing cluster consists of interconnected stand-alone computers which work cooperatively as a single integrated computing resource. In the past, clustered computer systems have demonstrated impressive results in handling heavy workloads with large data sets.

### 1.3.1.1 Cluster Architecture

Figure 1.15 shows the architecture of a typical server cluster built around a low-latency, high-bandwidth interconnection network. This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet). To build a larger cluster with more nodes, the interconnection network can be built with multiple levels of Gigabit Ethernet, Myrinet, or InfiniBand switches. Through hierarchical construction using a SAN, LAN, or WAN, one can build scalable clusters with an increasing number of nodes. The cluster is connected to the Internet via a virtual private network (VPN) gateway. The gateway IP address locates the cluster. The system image of a computer is decided by the way the OS manages the shared cluster resources. Most clusters have loosely coupled node computers. All resources of a server node are managed by their own OS. Thus, most clusters have multiple system images as a result of having many autonomous nodes under different OS control.



**FIGURE 1.15** A cluster of servers interconnected by a high-bandwidth SAN or LAN with shared I/O devices and disk arrays; the cluster acts as a single computer attached to the Internet.

### 1.3.1.2 Single-System Image

Greg Pfister [38] has indicated that an ideal cluster should merge multiple system images into a *single-system image (SSI)*. Cluster designers desire a *cluster operating system* or some middleware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes. An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource. SSI makes the cluster appear like a single machine to the user. A cluster with multiple system images is nothing but a collection of independent computers.

### 1.3.1.3 Hardware, Software, and Middleware Support

In Chapter 2, we will discuss cluster design principles for both small and large clusters. Clusters exploring massive parallelism are commonly known as MPPs. Almost all HPC clusters in the Top 500 list are also MPPs. The building blocks are computer nodes (PCs,

workstations, servers, or SMP), special communication software such as PVM or MPI, and a network interface card in each computer node. Most clusters run under the Linux OS. The computer nodes are interconnected by a high-bandwidth network (such as Gigabit Ethernet, Myrinet, InfiniBand, etc.).

Special cluster middleware supports are needed to create SSI or *high availability (HA)*. Both sequential and parallel applications can run on the cluster, and special parallel environments are needed to facilitate use of the cluster resources. For example, distributed memory has multiple images. Users may want all distributed memory to be shared by all servers by forming *distributed shared memory (DSM)*. Many SSI features are expensive or difficult to achieve at various cluster operational levels. Instead of achieving SSI, many clusters are loosely coupled machines. Using virtualization, one can build many virtual clusters dynamically, upon user demand. We will discuss virtual clusters in [Chapter 3](#) and the use of virtual clusters for cloud computing in [Chapters 4, 5, 6, and 9](#).

### 1.3.1.4 Major Cluster Design Issues

Unfortunately, a cluster-wide OS for complete resource sharing is not available yet. Middleware or OS extensions were developed at the user space to achieve SSI at selected functional levels. Without this middleware, cluster nodes cannot work together effectively to achieve cooperative computing. The software environments and applications must rely on the middleware to achieve high performance. The cluster benefits come from scalable performance, efficient message passing, high system availability, seamless fault tolerance, and cluster-wide job management, as summarized in [Table 1.3](#). We will address these issues in [Chapter 2](#).

**Table 1.3**

#### Critical Cluster Design Issues and Feasible Implementations

Features	Functional Characterization	Feasible Implementations
Availability and Support	Hardware and software support for sustained HA in cluster	Failover, failback, check pointing, rollback recovery, nonstop OS, etc.
Hardware Fault Tolerance	Automated failure management to eliminate all single points of failure	Component redundancy, hot swapping, RAID, multiple power supplies, etc.
Single System Image (SSI)	Achieving SSI at functional level with hardware and software support, middleware, or OS extensions	Hardware mechanisms or middleware support to achieve DSM at coherent cache level
Efficient Communications	To reduce message-passing system overhead and hide latencies	Fast message passing, active messages, enhanced MPI library, etc.
Cluster-wide Job Management	Using a global job management system with better scheduling and monitoring	Application of single-job management systems such as LSF, Codine, etc.
Dynamic Load Balancing	Balancing the workload of all processing nodes along with failure recovery	Workload monitoring, process migration, job replication and gang scheduling, etc.
Scalability and Programmability	Adding more servers to a cluster or adding more clusters to a grid as the workload or data set increases	Use of scalable interconnect, performance monitoring, distributed execution environment, and better software tools

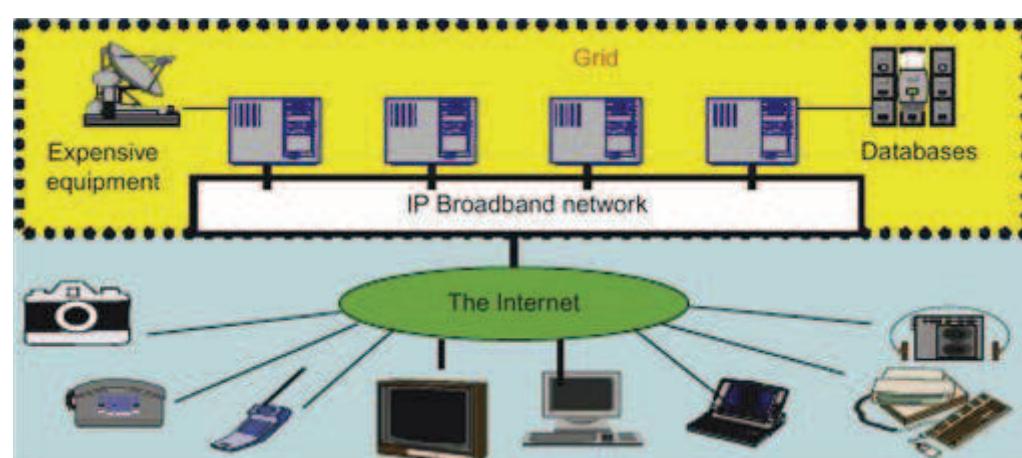
## 1.3.2 Grid Computing Infrastructures

In the past 30 years, users have experienced a natural growth path from Internet to web and grid computing services. Internet services such as the *Telnet* command enables a local computer to connect to a remote computer. A web service such as HTTP enables remote access of remote web pages. Grid computing is envisioned to allow close interaction among applications running on distant computers simultaneously. *Forbes Magazine* has projected the global growth of the IT-based economy from \$1 trillion in 2001 to \$20 trillion by 2015. The evolution from Internet to web and grid services is certainly playing a major role in this growth.

### 1.3.2.1 Computational Grids

Like an electric utility power grid, a *computing grid* offers an infrastructure that couples computers, software/middleware, special instruments, and people and sensors together. The grid is often constructed across LAN, WAN, or Internet backbone networks at a regional, national, or global scale. Enterprises or organizations present grids as integrated computing resources. They can also be viewed as *virtual platforms* to support *virtual organizations*. The computers used in a grid are primarily workstations, servers, clusters, and supercomputers. Personal computers, laptops, and PDAs can be used as access devices to a grid system.

Figure 1.16 shows an example computational grid built over multiple resource sites owned by different organizations. The resource sites offer complementary computing resources, including workstations, large servers, a mesh of processors, and Linux clusters to satisfy a chain of computational needs. The grid is built across various IP broadband networks including LANs and WANs already used by enterprises or organizations over the Internet. The grid is presented to users as an integrated resource pool as shown in the upper half of the figure.



**FIGURE 1.16** Computational grid or data grid providing computing utility, data, and information services through resource sharing and cooperation among participating organizations. Courtesy of Z. Xu, Chinese Academy of Science, 2004

Special instruments may be involved such as using the radio telescope in SETI@Home

search of life in the galaxy and the austrophysics@Swineburne for pulsars. At the server end, the grid is a network. At the client end, we see wired or wireless terminal devices. The grid integrates the computing, communication, contents, and transactions as rented services. Enterprises and consumers form the user base, which then defines the usage trends and service characteristics. Many national and international grids will be reported in [Chapter 7](#), the NSF TeraGrid in US, EGEE in Europe, and ChinaGrid in China for various distributed scientific grid applications.

### 1.3.2.2 Grid Families

Grid technology demands new distributed computing models, software/middleware support, network protocols, and hardware infrastructures. National grid projects are followed by industrial grid platform development by IBM, Microsoft, Sun, HP, Dell, Cisco, EMC, Platform Computing, and others. New *grid service providers* (GSPs) and new grid applications have emerged rapidly, similar to the growth of Internet and web services in the past two decades. In [Table 1.4](#), grid systems are classified in essentially two categories: *computational or data grids* and *P2P grids*. Computing or data grids are built primarily at the national level. In [Chapter 7](#), we will cover grid applications and lessons learned.

**Table 1.4**

#### Two Grid Computing Infrastructures and Representative Systems

Design Issues	Computational and Data Grids	P2P Grids
Grid Applications Reported	Distributed supercomputing, National Grid initiatives, etc.	Open grid with P2P flexibility, all resources from client machines
Representative Systems	TeraGrid built in US, ChinaGrid in China, and the e-Science grid built in UK	JXTA, FightAid@home, SETI@home
Development Lessons Learned	Restricted user groups, middleware bugs, protocols to acquire resources	Unreliable user-contributed resources, limited to a few apps

### 1.3.3 Peer-to-Peer Network Families

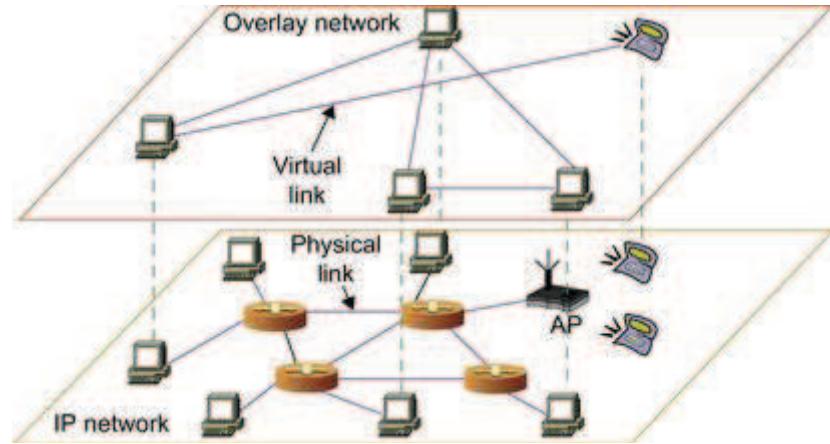
An example of a well-established distributed system is the *client-server architecture*. In this scenario, client machines (PCs and workstations) are connected to a central server for compute, e-mail, file access, and database applications. The *P2P architecture* offers a distributed model of networked systems. First, a P2P network is client-oriented instead of server-oriented. In this section, P2P systems are introduced at the physical level and overlay networks at the logical level.

#### 1.3.3.1 P2P Systems

In a P2P system, every node acts as both a client and a server, providing part of the system resources. Peer machines are simply client computers connected to the Internet.

All client machines act autonomously to join or leave the system freely. This implies that no master-slave relationship exists among the peers. No central coordination or central database is needed. In other words, no peer machine has a global view of the entire P2P system. The system is self-organizing with distributed control.

Figure 1.17 shows the architecture of a P2P network at two abstraction levels. Initially, the peers are totally unrelated. Each peer machine joins or leaves the P2P network voluntarily. Only the participating peers form the *physical network* at any time. Unlike the cluster or grid, a P2P network does not use a dedicated interconnection network. The physical network is simply an ad hoc network formed at various Internet domains randomly using the TCP/IP and NAI protocols. Thus, the physical network varies in size and topology dynamically due to the free membership in the P2P network.



**FIGURE 1.17** The structure of a P2P system by mapping a physical IP network to an overlay network built with virtual links. Courtesy of Zhenyu Li, Institute of Computing Technology, Chinese Academy of Sciences, 2010

### 1.3.3.2 Overlay Networks

Data items or files are distributed in the participating peers. Based on communication or file-sharing needs, the peer IDs form an *overlay network* at the logical level. This overlay is a virtual network formed by mapping each physical machine with its ID, logically, through a virtual mapping as shown in Figure 1.17. When a new peer joins the system, its peer ID is added as a node in the overlay network. When an existing peer leaves the system, its peer ID is removed from the overlay network automatically. Therefore, it is the P2P overlay network that characterizes the logical connectivity among the peers.

There are two types of overlay networks: *unstructured* and *structured*. An *unstructured overlay network* is characterized by a random graph. There is no fixed route to send messages or files among the nodes. Often, flooding is applied to send a query to all nodes in an unstructured overlay, thus resulting in heavy network traffic and nondeterministic search results. *Structured overlay networks* follow certain connectivity topology and rules for inserting and removing nodes (peer IDs) from the overlay graph. Routing mechanisms are developed to take advantage of the structured overlays.

### 1.3.3.3 P2P Application Families

Based on application, P2P networks are classified into four groups, as shown in [Table 1.5](#). The first family is for distributed file sharing of digital contents (music, videos, etc.) on the P2P network. This includes many popular P2P networks such as Gnutella, Napster, and BitTorrent, among others. Collaboration P2P networks include MSN or Skype chatting, instant messaging, and collaborative design, among others. The third family is for distributed P2P computing in specific applications. For example, SETI@home provides 25 Tflops of distributed computing power, collectively, over 3 million Internet host machines. Other P2P platforms, such as JXTA, .NET, and FightingAID@home, support naming, discovery, communication, security, and resource aggregation in some P2P applications. We will discuss these topics in more detail in [Chapters 8 and 9](#).

**Table 1.5**

**Major Categories of P2P Network Families [46]**

System Features	Distributed File Sharing	Collaborative Platform	Distributed P2P Computing	P2P Platform
Attractive Applications	Content distribution of MP3 music, video, open software, etc.	Instant messaging, collaborative design and gaming	Scientific exploration and social networking	Open networks for public resources
Operational Problems	Loose security and serious online copyright violations	Lack of trust, disturbed by spam, privacy, and peer collusion	Security holes, selfish partners, and peer collusion	Lack of standards or protection protocols
Example Systems	Gnutella, Napster, eMule, BitTorrent, Aimster, KaZaA, etc.	ICQ, AIM, Groove, Magi, Multiplayer Games, Skype, etc.	SETI@home, Geonome@home, etc.	JXTA, .NET, FightingAID@home, etc.

### 1.3.3.4 P2P Computing Challenges

P2P computing faces three types of heterogeneity problems in hardware, software, and network requirements. There are too many hardware models and architectures to select from; incompatibility exists between software and the OS; and different network connections and protocols make it too complex to apply in real applications. We need system scalability as the workload increases. System scaling is directly related to performance and bandwidth. P2P networks do have these properties. Data location is also important to affect collective performance. Data locality, network proximity, and interoperability are three design objectives in distributed P2P applications.

P2P performance is affected by routing efficiency and self-organization by participating peers. Fault tolerance, failure management, and load balancing are other important issues in using overlay networks. Lack of trust among peers poses another problem. Peers are strangers to one another. Security, privacy, and copyright violations are major worries by those in the industry in terms of applying P2P technology in business applications [\[35\]](#). In a P2P network, all clients provide resources including

computing power, storage space, and I/O bandwidth. The distributed nature of P2P networks also increases robustness, because limited peer failures do not form a single point of failure.

By replicating data in multiple peers, one can easily lose data in failed nodes. On the other hand, disadvantages of P2P networks do exist. Because the system is not centralized, managing it is difficult. In addition, the system lacks security. Anyone can log on to the system and cause damage or abuse. Further, all client computers connected to a P2P network cannot be considered reliable or virus-free. In summary, P2P networks are reliable for a small number of peer nodes. They are only useful for applications that require a low level of security and have no concern for data sensitivity. We will discuss P2P networks in [Chapter 8](#), and extending P2P technology to social networking in [Chapter 9](#).

### 1.3.4 Cloud Computing Over the Internet

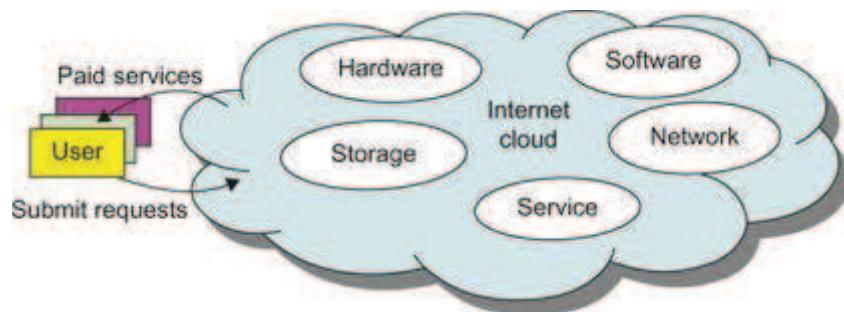
Gordon Bell, Jim Gray, and Alex Szalay [\[5\]](#) have advocated: “Computational science is changing to be data-intensive. Supercomputers must be balanced systems, not just CPU farms but also petascale I/O and networking arrays.” In the future, working with large data sets will typically mean sending the computations (programs) to the data, rather than copying the data to the workstations. This reflects the trend in IT of moving computing and data from desktops to large data centers, where there is on-demand provision of software, hardware, and data as a service. This data explosion has promoted the idea of cloud computing.

Cloud computing has been defined differently by many users and designers. For example, IBM, a major player in cloud computing, has defined it as follows: *“A cloud is a pool of virtualized computer resources. A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user-facing applications.”* Based on this definition, a cloud allows workloads to be deployed and scaled out quickly through rapid provisioning of virtual or physical machines. The cloud supports redundant, self-recovering, highly scalable programming models that allow workloads to recover from many unavoidable hardware/software failures. Finally, the cloud system should be able to monitor resource use in real time to enable rebalancing of allocations when needed.

#### 1.3.4.1 Internet Clouds

Cloud computing applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically (see [Figure 1.18](#)). The idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers. Cloud computing leverages its low cost and simplicity to benefit both users and providers. Machine virtualization has enabled such cost-effectiveness. Cloud computing intends to satisfy many user applications simultaneously. The cloud ecosystem must be designed to be secure, trustworthy, and dependable. Some computer users think of the cloud as a centralized resource pool. Others consider the cloud to be a server cluster which practices distributed computing

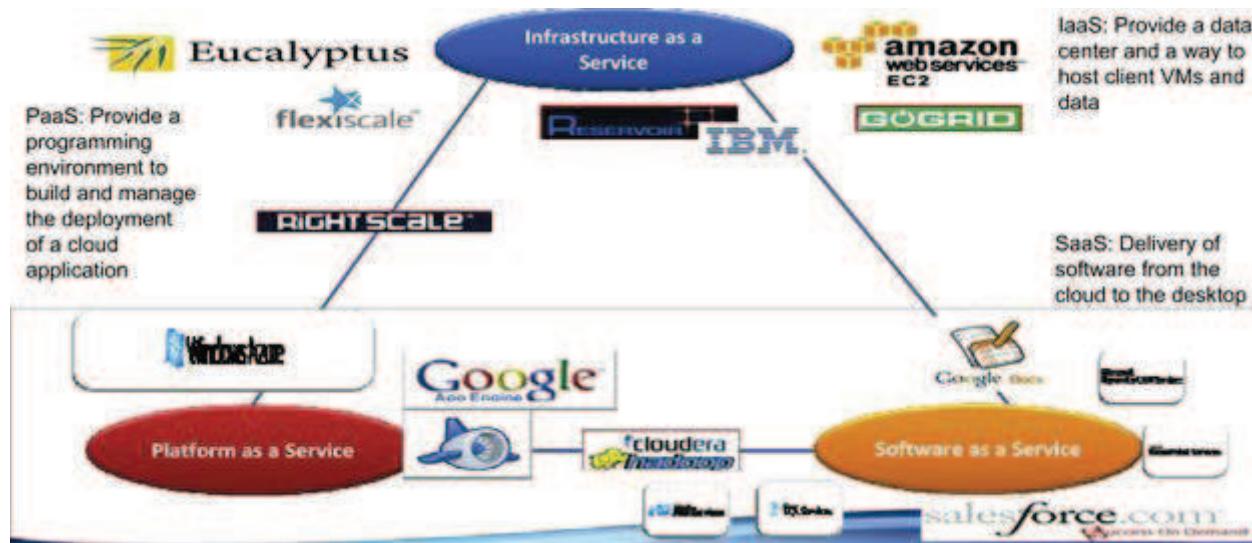
over all the servers used.



**FIGURE 1.18** Virtualized resources from data centers to form an Internet cloud, provisioned with hardware, software, storage, network, and services for paid users to run their applications.

### 1.3.4.2 The Cloud Landscape

Traditionally, a distributed computing system tends to be owned and operated by an autonomous administrative domain (e.g., a research laboratory or company) for on-premises computing needs. However, these traditional systems have encountered several performance bottlenecks: constant system maintenance, poor utilization, and increasing costs associated with hardware/software upgrades. Cloud computing as an on-demand computing paradigm resolves or relieves us from these problems. Figure 1.19 depicts the cloud landscape and major cloud players, based on three cloud service models. Chapters 4, 6, and 9 provide details regarding these cloud service offerings. Chapter 3 covers the relevant virtualization tools.



**FIGURE 1.19** Three cloud service models in a cloud landscape of major providers. Courtesy of Dennis Gannon, keynote address at Cloudcom2010 [19]

- **Infrastructure as a Service (IaaS)** This model puts together infrastructures demanded by users—namely servers, storage, networks, and the data center fabric. The user can deploy and run on multiple VMs running guest OSes on specific applications. The

user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources.

- **Platform as a Service (PaaS)** This model enables the user to deploy user-built applications onto a virtualized cloud platform. PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java. The platform includes both hardware and software integrated with specific programming interfaces. The provider supplies the API and software tools (e.g., Java, Python, Web 2.0, .NET). The user is freed from managing the cloud infrastructure.
- **Software as a Service (SaaS)** This refers to browser-initiated application software over thousands of paid cloud customers. The SaaS model applies to business processes, industry applications, *consumer relationship management (CRM)*, *enterprise resources planning (ERP)*, *human resources (HR)*, and collaborative applications. On the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are rather low, compared with conventional hosting of user applications.

Internet clouds offer four deployment modes: *private*, *public*, *managed*, and *hybrid* [11]. These modes demand different levels of security implications. The different SLAs imply that the security responsibility is shared among all the cloud providers, the cloud resource consumers, and the third-party cloud-enabled software providers. Advantages of cloud computing have been advocated by many IT experts, industry leaders, and computer science researchers.

In [Chapter 4](#), we will describe major cloud platforms that have been built and various cloud services offerings. The following list highlights eight reasons to adapt the cloud for upgraded Internet applications and web services:

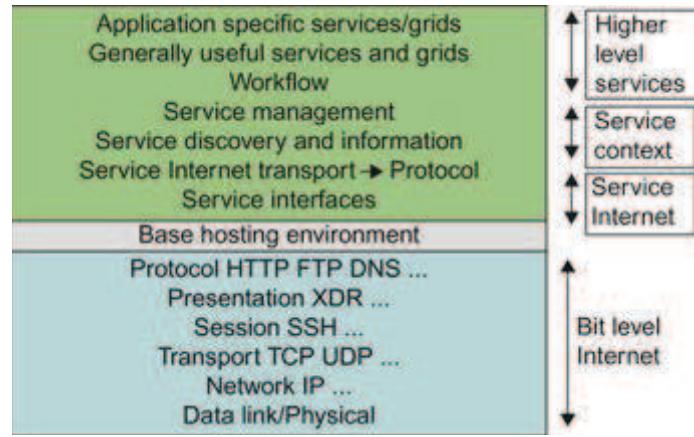
1. Desired location in areas with protected space and higher energy efficiency
2. Sharing of peak-load capacity among a large pool of users, improving overall utilization
3. Separation of infrastructure maintenance duties from domain-specific application development
4. Significant reduction in cloud computing cost, compared with traditional computing paradigms
5. Cloud computing programming and application development
6. Service and data discovery and content/service distribution
7. Privacy, security, copyright, and reliability issues
8. Service agreements, business models, and pricing policies

## 1.4 Software Environments for Distributed Systems and Clouds

This section introduces popular software environments for using distributed and cloud computing systems. [Chapters 5](#) and [6](#) discuss this subject in more depth.

### 1.4.1 Service-Oriented Architecture (SOA)

In grids/web services, Java, and CORBA, an entity is, respectively, a service, a Java object, and a CORBA distributed object in a variety of languages. These architectures build on the traditional seven Open Systems Interconnection (OSI) layers that provide the base networking abstractions. On top of this we have a base software environment, which would be .NET or Apache Axis for web services, the Java Virtual Machine for Java, and a broker network for CORBA. On top of this base environment one would build a higher level environment reflecting the special features of the distributed computing environment. This starts with entity interfaces and inter-entity communication, which rebuild the top four OSI layers but at the entity and not the bit level. [Figure 1.20](#) shows the layered architecture for distributed entities used in web services and grid systems.



**FIGURE 1.20** Layered achitecture for web services and the grids.

### 1.4.1.1 Layered Architecture for Web Services and Grids

The entity interfaces correspond to the *Web Services Description Language* (WSDL), Java method, and CORBA *interface definition language* (IDL) specifications in these example distributed systems. These interfaces are linked with customized, high-level communication systems: SOAP, RMI, and IIOP in the three examples. These communication systems support features including particular message patterns (such as *Remote Procedure Call* or RPC), fault recovery, and specialized routing. Often, these communication systems are built on message-oriented middleware (enterprise bus) infrastructure such as WebSphere MQ or *Java Message Service* (JMS) which provide rich functionality and support virtualization of routing, senders, and recipients.

In the case of fault tolerance, the features in the *Web Services Reliable Messaging* (WSRM) framework mimic the OSI layer capability (as in TCP fault tolerance) modified to match the different abstractions (such as messages versus packets, virtualized addressing) at the entity levels. Security is a critical capability that either uses or reimplements the capabilities seen in concepts such as *Internet Protocol Security* (IPsec) and secure sockets in the OSI layers. Entity communication is supported by higher level services for registries, metadata, and management of the entities discussed in [Section 5.4](#).

Here, one might get several models with, for example, JNDI (*Jini and Java Naming and*

*Directory Interface*) illustrating different approaches within the Java distributed object model. The CORBA Trading Service, UDDI (*Universal Description, Discovery, and Integration*), LDAP (*Lightweight Directory Access Protocol*), and ebXML (*Electronic Business using eXtensible Markup Language*) are other examples of discovery and information services described in [Section 5.4](#). Management services include service state and lifetime support; examples include the CORBA Life Cycle and Persistent states, the different Enterprise JavaBeans models, Jini's lifetime model, and a suite of web services specifications in [Chapter 5](#). The above language or interface terms form a collection of entity-level capabilities.

The latter can have performance advantages and offers a “shared memory” model allowing more convenient exchange of information. However, the distributed model has two critical advantages: namely, higher performance (from multiple CPUs when communication is unimportant) and a cleaner separation of software functions with clear software reuse and maintenance advantages. The distributed model is expected to gain popularity as the default approach to software systems. In the earlier years, CORBA and Java approaches were used in distributed systems rather than today's SOAP, XML, or REST (*Representational State Transfer*).

### **1.4.1.2 Web Services and Tools**

Loose coupling and support of heterogeneous implementations make services more attractive than distributed objects. [Figure 1.20](#) corresponds to two choices of service architecture: web services or REST systems (these are further discussed in [Chapter 5](#)). Both web services and REST systems have very distinct approaches to building reliable interoperable systems. In web services, one aims to fully specify all aspects of the service and its environment. This specification is carried with communicated messages using Simple Object Access Protocol (SOAP). The hosting environment then becomes a universal distributed operating system with fully distributed capability carried by SOAP messages. This approach has mixed success as it has been hard to agree on key parts of the protocol and even harder to efficiently implement the protocol by software such as Apache Axis.

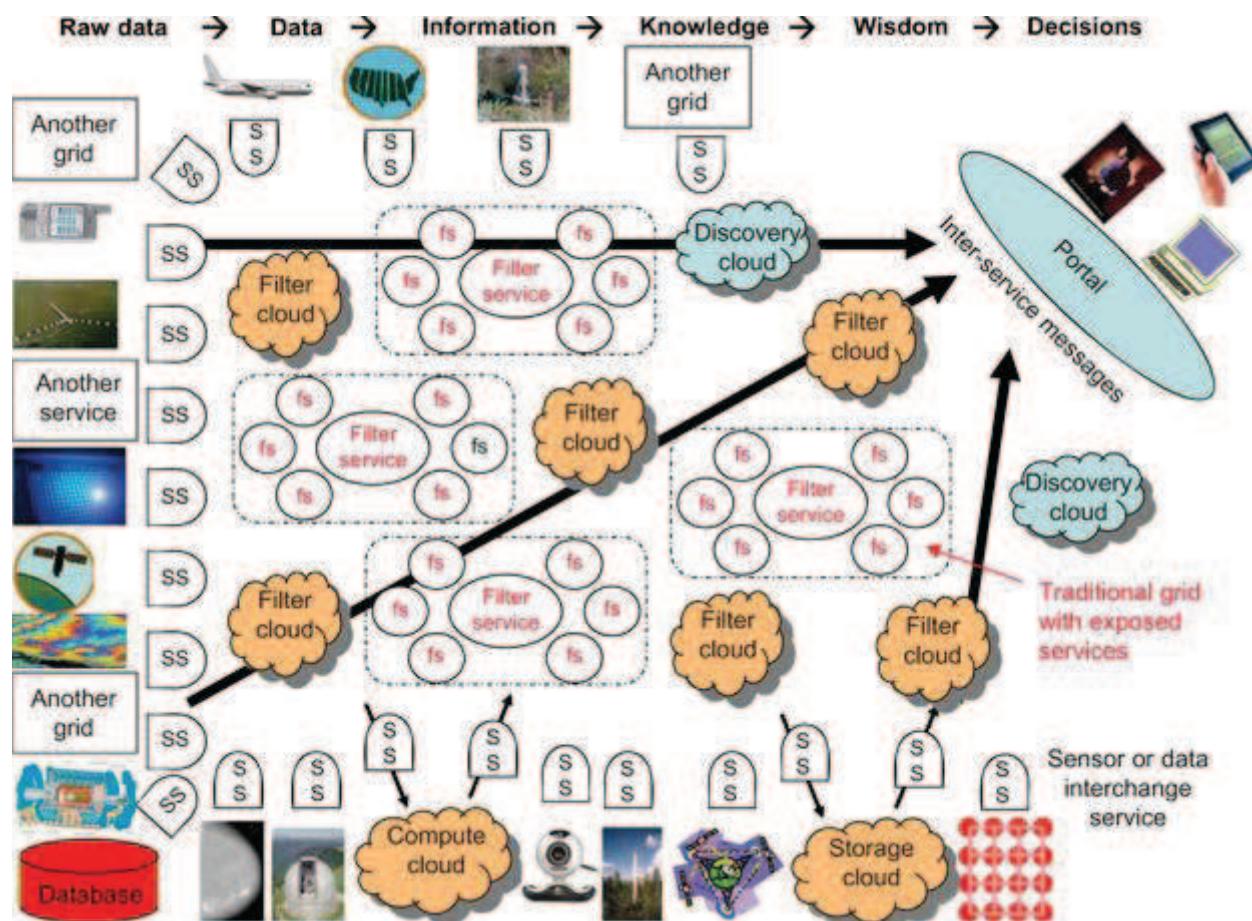
In the REST approach, one adopts simplicity as the universal principle and delegates most of the difficult problems to application (implementation-specific) software. In a web services language, REST has minimal information in the header, and the message body (that is opaque to generic message processing) carries all the needed information. REST architectures are clearly more appropriate for rapid technology environments. However, the ideas in web services are important and probably will be required in mature systems at a different level in the stack (as part of the application). Note that REST can use XML schemas but not those that are part of SOAP; “XML over HTTP” is a popular design choice in this regard. Above the communication and management layers, we have the ability to compose new entities or distributed programs by integrating several entities together.

In CORBA and Java, the distributed entities are linked with RPCs, and the simplest way to build composite applications is to view the entities as objects and use the

traditional ways of linking them together. For Java, this could be as simple as writing a Java program with method calls replaced by Remote Method Invocation (RMI), while CORBA supports a similar model with a syntax reflecting the C++ style of its entity (object) interfaces. Allowing the term “grid” to refer to a single service or to represent a collection of services, here sensors represent entities that output data (as messages), and grids and clouds represent collections of services that have multiple message-based inputs and outputs.

### 1.4.1.3 The Evolution of SOA

As shown in Figure 1.21, *service-oriented architecture (SOA)* has evolved over the years. SOA applies to building grids, clouds, grids of clouds, clouds of grids, clouds of clouds (also known as interclouds), and systems of systems in general. A large number of sensors provide data-collection services, denoted in the figure as SS (*sensor service*). A sensor can be a ZigBee device, a Bluetooth device, a WiFi access point, a personal computer, a GPA, or a wireless phone, among other things. Raw data is collected by sensor services. All the SS devices interact with large or small computers, many forms of grids, databases, the compute cloud, the storage cloud, the filter cloud, the discovery cloud, and so on. *Filter services* (fs in the figure) are used to eliminate unwanted raw data, in order to respond to specific requests from the web, the grid, or web services.



**FIGURE 1.21** The evolution of SOA: grids of clouds and grids, where “SS” refers to a sensor service and “fs” to a filter or transforming service.

A collection of filter services forms a filter cloud. We will cover various clouds for compute, storage, filter, and discovery in [Chapters 4, 5, and 6](#), and various grids, P2P networks, and the IoT in [Chapters 7, 8, and 9](#). SOA aims to search for, or sort out, the useful data from the massive amounts of raw data items. Processing this data will generate useful information, and subsequently, the knowledge for our daily use. In fact, wisdom or intelligence is sorted out of large knowledge bases. Finally, we make intelligent decisions based on both biological and machine wisdom. Readers will see these structures more clearly in subsequent chapters.

Most distributed systems require a web interface or portal. For raw data collected by a large number of sensors to be transformed into useful information or knowledge, the data stream may go through a sequence of compute, storage, filter, and discovery clouds. Finally, the inter-service messages converge at the portal, which is accessed by all users. Two example portals, OGFCE and HUBzero, are described in [Section 5.3](#) using both web service (portlet) and Web 2.0 (gadget) technologies. Many distributed programming models are also built on top of these basic constructs.

#### **1.4.1.4 Grids versus Clouds**

The boundary between grids and clouds are getting blurred in recent years. For web services, workflow technologies are used to coordinate or orchestrate services with certain specifications used to define critical business process models such as two-phase transactions. [Section 5.2](#) discusses the general approach used in workflow, the BPEL Web Service standard, and several important workflow approaches including Pegasus, Taverna, Kepler, Trident, and Swift. In all approaches, one is building a collection of services which together tackle all or part of a distributed computing problem.

In general, a grid system applies static resources, while a cloud emphasizes elastic resources. For some researchers, the differences between grids and clouds are limited only in dynamic resource allocation based on virtualization and autonomic computing. One can build a grid out of multiple clouds. This type of grid can do a better job than a pure cloud, because it can explicitly support negotiated resource allocation. Thus one may end up building with a *system of systems*: such as a *cloud of clouds*, a *grid of clouds*, or a *cloud of grids*, or *inter-clouds* as a basic SOA architecture.

#### **1.4.2 Trends toward Distributed Operating Systems**

The computers in most distributed systems are loosely coupled. Thus, a distributed system inherently has multiple system images. This is mainly due to the fact that all node machines run with an independent operating system. To promote resource sharing and fast communication among node machines, it is best to have a *distributed OS* that manages all resources coherently and efficiently. Such a system is most likely to be a closed system, and it will likely rely on message passing and RPCs for internode communications. It should be pointed out that a distributed OS is crucial for upgrading the performance, efficiency, and flexibility of distributed applications.

#### 1.4.2.1 Distributed Operating Systems

Tanenbaum [26] identifies three approaches for distributing resource management functions in a distributed computer system. The first approach is to build a *network OS* over a large number of heterogeneous OS platforms. Such an OS offers the lowest transparency to users, and is essentially a distributed file system, with independent computers relying on file sharing as a means of communication. The second approach is to develop middleware to offer a limited degree of resource sharing, similar to the MOSIX/OS developed for clustered systems (see [Section 2.4.4](#)). The third approach is to develop a truly *distributed OS* to achieve higher use or system transparency. [Table 1.6](#) compares the functionalities of these three distributed operating systems.

**Table 1.6**

**Feature Comparison of Three Distributed Operating Systems**

Distributed OS Functionality	AMOEBA Developed at Vrije University [46]	DCE as OSF/1 by Open Software Foundation [7]	MOSIX for Linux Clusters at Hebrew University [3]
History and Current System Status	Written in C and tested in the European community; version 5.2 released in 1995	Built as a user extension on top of UNIX, VMS, Windows, OS/2, etc.	Developed since 1977, now called MOSIX2 used in HPC Linux and GPU clusters
Distributed OS Architecture	Microkernel-based and location-transparent, uses many servers to handle files, directory, replication, run, boot, and TCP/IP services	Middleware OS providing a platform for running distributed applications; The system supports RPC, security, and threads	A distributed OS with resource discovery, process migration, runtime support, load balancing, flood control, configuration, etc.
OS Kernel, Middleware, and Virtualization Support	A special microkernel that handles low-level process, memory, I/O, and communication functions	DCE packages handle file, time, directory, security services, RPC, and authentication at middleware or user space	MOSIX2 runs with Linux 2.6; extensions for use in multiple clusters and clouds with provisioned VMs
Communication Mechanisms	Uses a network-layer FLIP protocol and RPC to implement point-to-point and group communication	RPC supports authenticated communication and other security services in user programs	Using PVM, MPI in collective communications, priority process control, and queuing services

#### 1.4.2.2 Amoeba versus DCE

DCE is a middleware-based system for distributed computing environments. The Amoeba was academically developed at Free University in the Netherlands. The Open Software Foundation (OSF) has pushed the use of DCE for distributed computing. However, the Amoeba, DCE, and MOSIX2 are still research prototypes that are primarily used in academia. No successful commercial OS products followed these research systems.

We need new web-based operating systems to support virtualization of resources in

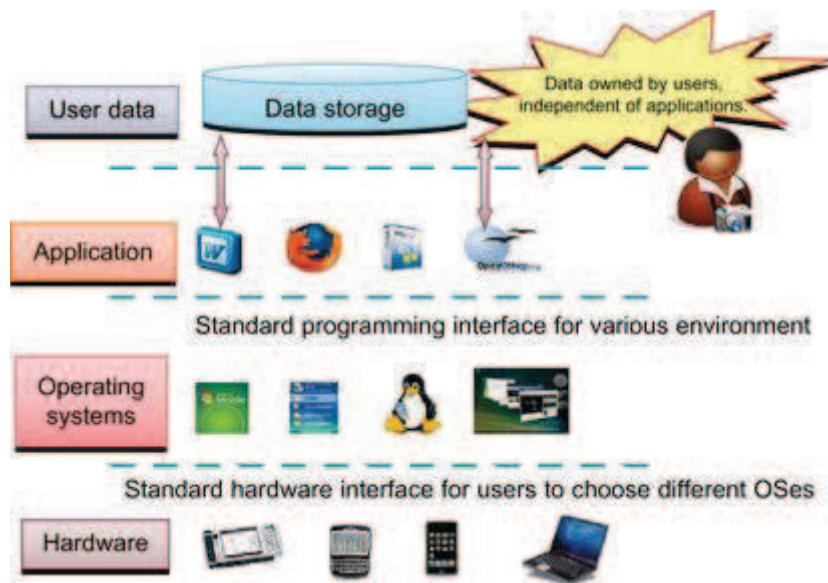
distributed environments. This is still a wide-open area of research. To balance the resource management workload, the functionalities of such a distributed OS should be distributed to any available server. In this sense, the conventional OS runs only on a centralized platform. With the distribution of OS services, the distributed OS design should take a lightweight microkernel approach like the Amoeba [46], or should extend an existing OS like the DCE [7] by extending UNIX. The trend is to free users from most resource management duties.

### **1.4.2.3 MOSIX2 for Linux Clusters**

MOSIX2 is a distributed OS [3], which runs with a virtualization layer in the Linux environment. This layer provides a partial *single-system image* to user applications. MOSIX2 supports both sequential and parallel applications, and discovers resources and migrates software processes among Linux nodes. MOSIX2 can manage a Linux cluster or a grid of multiple clusters. Flexible management of a grid allows owners of clusters to share their computational resources among multiple cluster owners. A MOSIX-enabled grid can extend indefinitely as long as trust exists among the cluster owners. The MOSIX2 is being explored for managing resources in all sorts of clusters, including Linux clusters, GPU clusters, grids, and even clouds if VMs are used. We will study MOSIX and its applications in Section 2.4.4.

### **1.4.2.4 Transparency in Programming Environments**

Figure 1.22 shows the concept of a transparent computing infrastructure for future computing platforms. The user data, applications, OS, and hardware are separated into four levels. Data is owned by users, independent of the applications. The OS provides clear interfaces, standard programming interfaces, or system calls to application programmers. In future cloud infrastructure, the hardware will be separated by standard interfaces from the OS. Thus, users will be able to choose from different OSes on top of the hardware devices they prefer to use. To separate user data from specific application programs, users can enable cloud applications as SaaS. Thus, users can switch among different services. The data will not be bound to specific applications.



**FIGURE 1.22** A transparent computing environment that separates the user data, application, OS, and hardware in time and space – an ideal model for cloud computing.

### 1.4.3 Parallel and Distributed Programming Models

In this section, we will explore four programming models for distributed computing with expected scalable performance and application flexibility. [Table 1.7](#) summarizes three of these models, along with some software tool sets developed in recent years. As we will discuss, MPI is the most popular programming model for message-passing systems. Google's MapReduce and BigTable are for effective use of resources from Internet clouds and data centers. Service clouds demand extending Hadoop, EC2, and S3 to facilitate distributed computing over distributed storage systems. Many other models have also been proposed or developed in the past. In [Chapters 5 and 6](#), we will discuss parallel and distributed programming in more details.

**Table 1.7**

#### Parallel and Distributed Programming Models and Tool Sets

Model	Description	Features
MPI	A library of subprograms that can be called from C or FORTRAN to write parallel programs running on distributed computer systems [6,28,42]	Specify synchronous or asynchronous point-to-point and collective communication commands and I/O operations in user programs for message-passing execution
MapReduce	A web programming model for scalable data processing on large clusters over large data sets, or in web search operations [16]	<i>Map</i> function generates a set of intermediate key/value pairs; <i>Reduce</i> function merges all intermediate values with the same key
Hadoop	A software library to write and run large user applications on vast data sets in business applications ( <a href="http://hadoop.apache.org/core">http://hadoop.apache.org/core</a> )	A scalable, economical, efficient, and reliable tool for providing users with easy access of commercial clusters

#### 1.4.3.1 Message-Passing Interface (MPI)

This is the primary programming standard used to develop parallel and concurrent programs to run on a distributed system. MPI is essentially a library of subprograms that can be called from C or FORTRAN to write parallel programs running on a distributed system. The idea is to embody clusters, grid systems, and P2P systems with upgraded web services and utility computing applications. Besides MPI, distributed programming can be also supported with low-level primitives such as the *Parallel Virtual Machine* (PVM). Both MPI and PVM are described in Hwang and Xu [28].

### 1.4.3.2 MapReduce

This is a web programming model for scalable data processing on large clusters over large data sets [16]. The model is applied mainly in web-scale search and cloud computing applications. The user specifies a *Map* function to generate a set of intermediate key/value pairs. Then the user applies a *Reduce* function to merge all intermediate values with the same intermediate key. MapReduce is highly scalable to explore high degrees of parallelism at different job levels. A typical MapReduce computation process can handle terabytes of data on tens of thousands or more client machines. Hundreds of MapReduce programs can be executed simultaneously; in fact, thousands of MapReduce jobs are executed on Google's clusters every day.

### 1.4.3.3 Hadoop Library

Hadoop offers a software platform that was originally developed by a Yahoo! group. The package enables users to write and run applications over vast amounts of distributed data. Users can easily scale Hadoop to store and process petabytes of data in the web space. Also, Hadoop is economical in that it comes with an open source version of MapReduce that minimizes overhead in task spawning and massive data communication. It is efficient, as it processes data with a high degree of parallelism across a large number of commodity nodes, and it is reliable in that it automatically keeps multiple data copies to facilitate redeployment of computing tasks upon unexpected system failures.

### 1.4.3.4 Open Grid Services Architecture (OGSA)

The development of grid infrastructure is driven by large-scale distributed computing applications. These applications must count on a high degree of resource and data sharing. Table 1.8 introduces OGSA as a common standard for general public use of grid services. Genesis II is a realization of OGSA. Key features include a distributed execution environment, *Public Key Infrastructure (PKI)* services using a local *certificate authority (CA)*, trust management, and security policies in grid computing.

---

**Table 1.8**

**Grid Standards and Toolkits for Scientific and Engineering Applications [6]**

---

Standards	Service Functionalities	Key Features and Security Infrastructure
OGSA	Open Grid Services Architecture; offers	Supports a heterogeneous distributed environment, bridging CAs, multiple

Standard	common grid service standards for general public use	trusted intermediaries, dynamic policies, multiple security mechanisms, etc.
Globus Toolkits	Resource allocation, Globus security infrastructure (GSI), and generic security service API	Sign-in multisite authentication with PKI, Kerberos, SSL, Proxy, delegation, and GSS API for message integrity and confidentiality
IBM Grid Toolbox	AIX and Linux grids built on top of Globus Toolkit, autonomic computing, replica services	Uses simple CA, grants access, grid service (ReGS), supports grid application for Java (GAF4J), GridMap in IntraGrid for security update

### 1.4.3.5 Globus Toolkits and Extensions

*Globus* is a middleware library jointly developed by the U.S. Argonne National Laboratory and USC Information Science Institute over the past decade. This library implements some of the OGSA standards for resource discovery, allocation, and security enforcement in a grid environment. The Globus packages support multisite mutual authentication with PKI certificates. The current version of Globus, GT 4, has been in use since 2008. In addition, IBM has extended Globus for business applications. We will cover Globus and other grid computing middleware in more detail in [Chapter 7](#).

## 1.5 Performance, Security, and Energy Efficiency

In this section, we will discuss the fundamental design principles along with rules of thumb for building massively distributed computing systems. Coverage includes scalability, availability, programming models, and security issues in clusters, grids, P2P networks, and Internet clouds.

### 1.5.1 Performance Metrics and Scalability Analysis

Performance metrics are needed to measure various distributed systems. In this section, we will discuss various dimensions of scalability and performance laws. Then we will examine system scalability against OS images and the limiting factors encountered.

#### 1.5.1.1 Performance Metrics

We discussed *CPU speed* in MIPS and *network bandwidth* in Mbps in [Section 1.3.1](#) to estimate processor and network performance. In a distributed system, performance is attributed to a large number of factors. *System throughput* is often measured in MIPS, Tflops (*tera floating-point operations per second*), or TPS (*transactions per second*). Other measures include *job response time* and *network latency*. An interconnection network that has low latency and high bandwidth is preferred. System overhead is often attributed to OS boot time, compile time, I/O data rate, and the runtime support system used. Other performance-related metrics include the QoS for Internet and web services; *system availability and dependability*; and *security resilience* for system defense against network attacks.

#### 1.5.1.2 Dimensions of Scalability

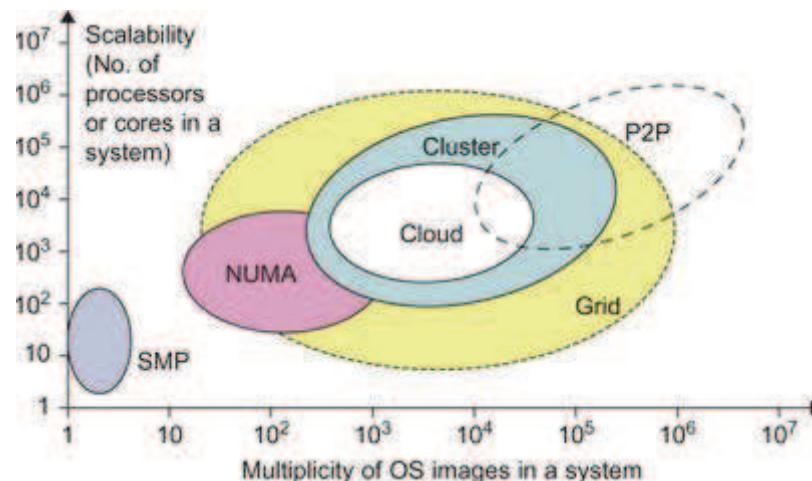
Users want to have a distributed system that can achieve scalable performance. Any resource upgrade in a system should be backward compatible with existing hardware and software resources. Overdesign may not be cost-effective. System scaling can increase or decrease resources depending on many practical factors. The following dimensions of scalability are characterized in parallel and distributed systems:

- **Size scalability** This refers to achieving higher performance or more functionality by increasing the *machine size*. The word “size” refers to adding processors, cache, memory, storage, or I/O channels. The most obvious way to determine size scalability is to simply count the number of processors installed. Not all parallel computer or distributed architectures are equally size-scalable. For example, the IBM S2 was scaled up to 512 processors in 1997. But in 2008, the IBM BlueGene/L system scaled up to 65,000 processors.
- **Software scalability** This refers to upgrades in the OS or compilers, adding mathematical and engineering libraries, porting new application software, and installing more user-friendly programming environments. Some software upgrades may not work with large system configurations. Testing and fine-tuning of new software on larger systems is a nontrivial job.
- **Application scalability** This refers to matching *problem size* scalability with *machine size* scalability. Problem size affects the size of the data set or the workload increase. Instead of increasing machine size, users can enlarge the problem size to enhance system efficiency or cost-effectiveness.
- **Technology scalability** This refers to a system that can adapt to changes in building technologies, such as the component and networking technologies discussed in [Section 3.1](#). When scaling a system design with new technology one must consider three aspects: *time*, *space*, and *heterogeneity*. (1) Time refers to generation scalability. When changing to new-generation processors, one must consider the impact to the motherboard, power supply, packaging and cooling, and so forth. Based on past experience, most systems upgrade their commodity processors every three to five years. (2) Space is related to packaging and energy concerns. Technology scalability demands harmony and portability among suppliers. (3) Heterogeneity refers to the use of hardware components or software packages from different vendors. Heterogeneity may limit the scalability.

### 1.5.1.3 Scalability versus OS Image Count

In [Figure 1.23](#), *scalable performance* is estimated against the *multiplicity of OS images* in distributed systems deployed up to 2010. Scalable performance implies that the system can achieve higher speed by adding more processors or servers, enlarging the physical node’s memory size, extending the disk capacity, or adding more I/O channels. The OS image is counted by the number of independent OS images observed in a cluster, grid, P2P network, or the cloud. SMP and NUMA are included in the comparison. An *SMP* (*symmetric multiprocessor*) server has a single system image, which could be a single node in a large cluster. By 2010 standards, the largest shared-memory SMP node was limited to a few hundred processors. The scalability of SMP systems is constrained primarily by

packaging and the system interconnect used.



**FIGURE 1.23** System scalability versus multiplicity of OS images based on 2010 technology.

*NUMA (nonuniform memory access)* machines are often made out of SMP nodes with distributed, shared memory. A NUMA machine can run with multiple operating systems, and can scale to a few thousand processors communicating with the MPI library. For example, a NUMA machine may have 2,048 processors running 32 SMP operating systems, resulting in 32 OS images in the 2,048-processor NUMA system. The cluster nodes can be either SMP servers or high-end machines that are loosely coupled together. Therefore, clusters have much higher scalability than NUMA machines. The number of OS images in a cluster is based on the cluster nodes concurrently in use. The cloud could be a virtualized cluster. As of 2010, the largest cloud was able to scale up to a few thousand VMs.

Keeping in mind that many cluster nodes are SMP or multicore servers, the total number of processors or cores in a cluster system is one or two orders of magnitude greater than the number of OS images running in the cluster. The grid node could be a server cluster, or a mainframe, or a supercomputer, or an MPP. Therefore, the number of OS images in a large grid structure could be hundreds or thousands fewer than the total number of processors in the grid. A P2P network can easily scale to millions of independent peer nodes, essentially desktop machines. P2P performance depends on the QoS in a public network. Low-speed P2P networks, Internet clouds, and computer clusters should be evaluated at the same networking level.

#### 1.5.1.4 Amdahl's Law

Consider the execution of a given program on a uniprocessor workstation with a total execution time of  $T$  minutes. Now, let's say the program has been parallelized or partitioned for parallel execution on a cluster of many processing nodes. Assume that a fraction  $\alpha$  of the code must be executed sequentially, called the *sequential bottleneck*. Therefore,  $(1 - \alpha)$  of the code can be compiled for parallel execution by  $n$  processors. The total execution time of the program is calculated by  $\alpha T + (1 - \alpha)T/n$ , where the first term

is the sequential execution time on a single processor and the second term is the parallel execution time on  $n$  processing nodes.

All system or communication overhead is ignored here. The I/O time or exception handling time is also not included in the following speedup analysis. Amdahl's Law states that the *speedup factor* of using the  $n$ -processor system over the use of a single processor is expressed by:

(1.1)

The maximum speedup of  $n$  is achieved only if the *sequential bottleneck*  $\alpha$  is reduced to zero or the code is fully parallelizable with  $\alpha = 0$ . As the cluster becomes sufficiently large, that is,  $n \rightarrow \infty$ ,  $S$  approaches  $1/\alpha$ , an upper bound on the speedup  $S$ . Surprisingly, this upper bound is independent of the cluster size  $n$ . The sequential bottleneck is the portion of the code that cannot be parallelized. For example, the maximum speedup achieved is 4, if  $\alpha = 0.25$  or  $1 - \alpha = 0.75$ , even if one uses hundreds of processors. Amdahl's law teaches us that we should make the sequential bottleneck as small as possible. Increasing the cluster size alone may not result in a good speedup in this case.

### 1.5.1.5 Problem with Fixed Workload

In Amdahl's law, we have assumed the same amount of workload for both sequential and parallel execution of the program with a fixed problem size or data set. This was called *fixed-workload speedup* by Hwang and Xu [14]. To execute a fixed workload on  $n$  processors, parallel processing may lead to a *system efficiency* defined as follows:

(1.2)

Very often the system efficiency is rather low, especially when the cluster size is very large. To execute the aforementioned program on a cluster with  $n = 256$  nodes, extremely low efficiency  $E = 1/[0.25 \times 256 + 0.75] = 1.5\%$  is observed. This is because only a few processors (say, 4) are kept busy, while the majority of the nodes are left idling.

### 1.5.1.6 Gustafson's Law

To achieve higher efficiency when using a large cluster, we must consider scaling the problem size to match the cluster capability. This leads to the following speedup law proposed by John Gustafson (1988), referred as *scaled-workload speedup* in [14]. Let  $W$  be the workload in a given program. When using an  $n$ -processor system, the user scales the workload to  $W' = \alpha W + (1 - \alpha)nW$ . Note that only the parallelizable portion of the workload is scaled  $n$  times in the second term. This scaled workload  $W'$  is essentially the sequential execution time on a single processor. The parallel execution time of a scaled workload  $W'$  on  $n$  processors is defined by a *scaled-workload speedup* as follows:

This speedup is known as Gustafson's law. By fixing the parallel execution time at level  $W$ , the following efficiency expression is obtained:

For the preceding program with a scaled workload, we can improve the efficiency of using a 256-node cluster to  $E' = 0.25/256 + 0.75 = 0.751$ . One should apply Amdahl's law and Gustafson's law under different workload conditions. For a fixed workload, users should apply Amdahl's law. To solve scaled problems, users should apply Gustafson's law.

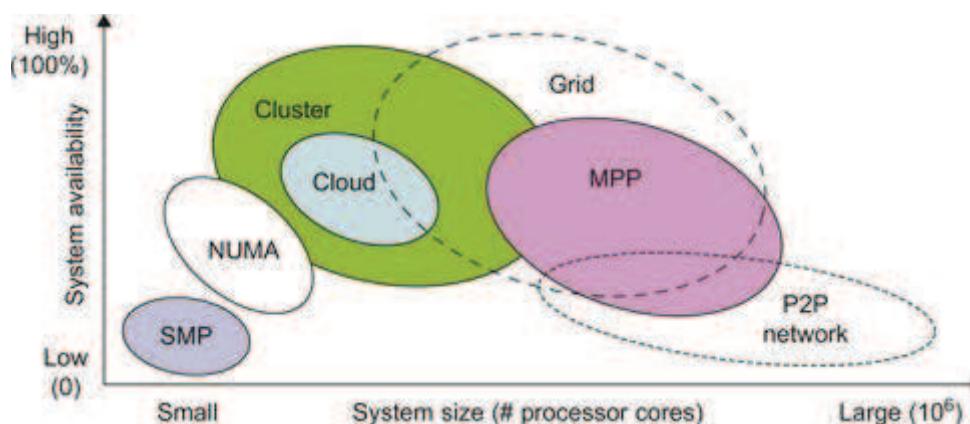
## 1.5.2 Fault Tolerance and System Availability

In addition to performance, system availability and application flexibility are two other important design goals in a distributed computing system.

### 1.5.2.1 System Availability

HA (high availability) is desired in all clusters, grids, P2P networks, and cloud systems. A system is highly available if it has a long *mean time to failure* (MTTF) and a short *mean time to repair* (MTTR). *System availability* is formally defined as follows:

System availability is attributed to many factors. All hardware, software, and network components may fail. Any failure that will pull down the operation of the entire system is called a *single point of failure*. The rule of thumb is to design a dependable computing system with no single point of failure. Adding hardware redundancy, increasing component reliability, and designing for testability will help to enhance system availability and dependability. In [Figure 1.24](#), the effects on system availability are estimated by scaling the system size in terms of the number of processor cores in the system.



**FIGURE 1.24** Estimated system availability by system size of common configurations in 2010.

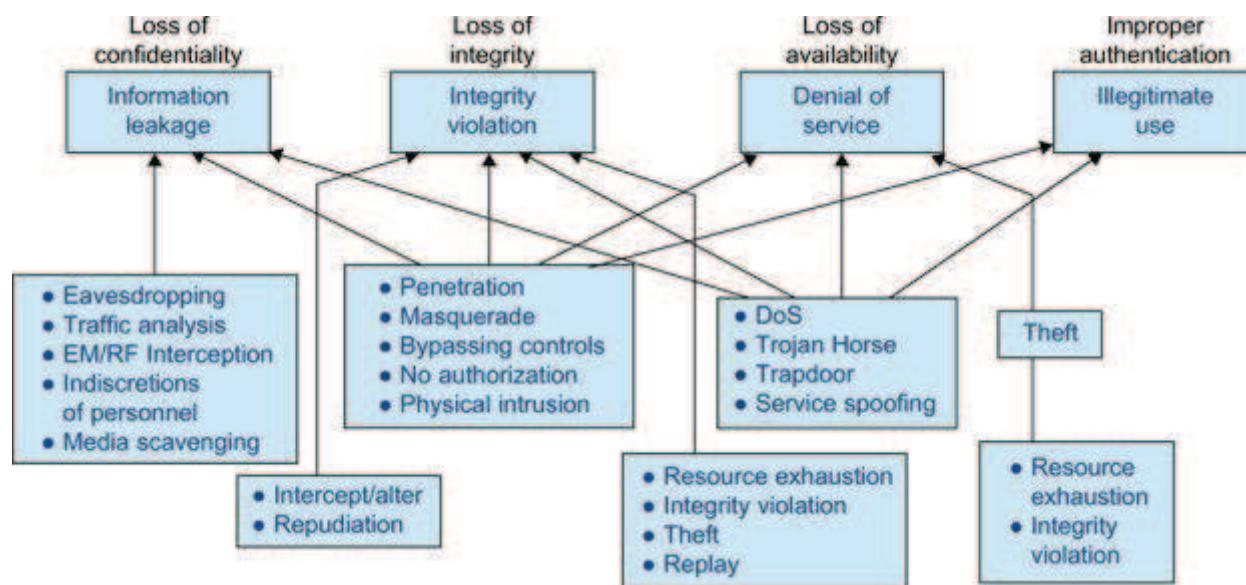
In general, as a distributed system increases in size, availability decreases due to a higher chance of failure and a difficulty in isolating the failures. Both SMP and MPP are very vulnerable with centralized resources under one OS. NUMA machines have improved in availability due to the use of multiple OSes. Most clusters are designed to have HA with failover capability. Meanwhile, private clouds are created out of virtualized data centers; hence, a cloud has an estimated availability similar to that of the hosting cluster. A grid is visualized as a hierarchical cluster of clusters. Grids have higher availability due to the isolation of faults. Therefore, clusters, clouds, and grids have decreasing availability as the system increases in size. A P2P file-sharing network has the highest aggregation of client machines. However, it operates independently with low availability, and even many peer nodes depart or fail simultaneously.

## 1.5.3 Network Threats and Data Integrity

Clusters, grids, P2P networks, and clouds demand security and copyright protection if they are to be accepted in today's digital society. This section introduces system vulnerability, network threats, defense countermeasures, and copyright protection in distributed or cloud computing systems.

### 1.5.3.1 Threats to Systems and Networks

Network viruses have threatened many users in widespread attacks. These incidents have created a worm epidemic by pulling down many routers and servers, and are responsible for the loss of billions of dollars in business, government, and services. Figure 1.25 summarizes various attack types and their potential damage to users. As the figure shows, information leaks lead to a loss of confidentiality. Loss of data integrity may be caused by user alteration, Trojan horses, and service spoofing attacks. A *denial of service* (DoS) results in a loss of system operation and Internet connections.



Lack of authentication or authorization leads to attackers' illegitimate use of computing resources. Open resources such as data centers, P2P networks, and grid and cloud infrastructures could become the next targets. Users need to protect clusters, grids, clouds, and P2P systems. Otherwise, users should not use or trust them for outsourced work. Malicious intrusions to these systems may destroy valuable hosts, as well as network and storage resources. Internet anomalies found in routers, gateways, and distributed hosts may hinder the acceptance of these public-resource computing services.

### **1.5.3.2 Security Responsibilities**

Three security requirements are often considered: *confidentiality*, *integrity*, and *availability* for most Internet service providers and cloud users. In the order of SaaS, PaaS, and IaaS, the providers gradually release the responsibility of security control to the cloud users. In summary, the SaaS model relies on the cloud provider to perform all security functions. At the other extreme, the IaaS model wants the users to assume almost all security functions, but to leave availability in the hands of the providers. The PaaS model relies on the provider to maintain data integrity and availability, but burdens the user with confidentiality and privacy control.

### **1.5.3.3 Copyright Protection**

Collusive piracy is the main source of intellectual property violations within the boundary of a P2P network. Paid clients (colluders) may illegally share copyrighted content files with unpaid clients (pirates). Online piracy has hindered the use of open P2P networks for commercial content delivery. One can develop a proactive content poisoning scheme to stop colluders and pirates from alleged copyright infringements in P2P file sharing. Pirates are detected in a timely manner with identity-based signatures and timestamped tokens. This scheme stops collusive piracy from occurring without hurting legitimate P2P clients. [Chapters 4](#) and [7](#) cover grid and cloud security, P2P reputation systems, and copyright protection.

### **1.5.3.4 System Defense Technologies**

Three generations of network defense technologies have appeared in the past. In the first generation, tools were designed to prevent or avoid intrusions. These tools usually manifested themselves as access control policies or tokens, cryptographic systems, and so forth. However, an intruder could always penetrate a secure system because there is always a weak link in the security provisioning process. The second generation detected intrusions in a timely manner to exercise remedial actions. These techniques included firewalls, intrusion detection systems (IDSe), PKI services, reputation systems, and so on. The third generation provides more intelligent responses to intrusions.

### **1.5.3.5 Data Protection Infrastructure**

Security infrastructure is required to safeguard web and cloud services. At the user level, one needs to perform trust negotiation and reputation aggregation over all users. At the application end, we need to establish security precautions in worm containment and intrusion detection against virus, worm, and distributed DoS (DDoS) attacks. We also need to deploy mechanisms to prevent online piracy and copyright violations of digital content. In [Chapter 4](#), we will study reputation systems for protecting cloud systems and data centers. Security responsibilities are divided between cloud providers and users differently for the three cloud service models. The providers are totally responsible for platform availability. The IaaS users are more responsible for the confidentiality issue. The IaaS providers are more responsible for data integrity. In PaaS and SaaS services, providers and users are equally responsible for preserving data integrity and confidentiality.

## 1.5.4 Energy Efficiency in Distributed Computing

Primary performance goals in conventional parallel and distributed computing systems are high performance and high throughput, considering some form of performance reliability (e.g., fault tolerance and security). However, these systems recently encountered new challenging issues including energy efficiency, and workload and resource outsourcing. These emerging issues are crucial not only on their own, but also for the sustainability of large-scale computing systems in general. This section reviews energy consumption issues in servers and HPC systems, an area known as *distributed power management (DPM)*.

Protection of data centers demands integrated solutions. Energy consumption in parallel and distributed computing systems raises various monetary, environmental, and system performance issues. For example, Earth Simulator and Petaflop are two systems with 12 and 100 megawatts of peak power, respectively. With an approximate price of \$100 per megawatt, their energy costs during peak operation times are \$1,200 and \$10,000 per hour; this is beyond the acceptable budget of many (potential) system operators. In addition to power cost, cooling is another issue that must be addressed due to negative effects of high temperature on electronic components. The rising temperature of a circuit not only derails the circuit from its normal range, but also decreases the lifetime of its components.

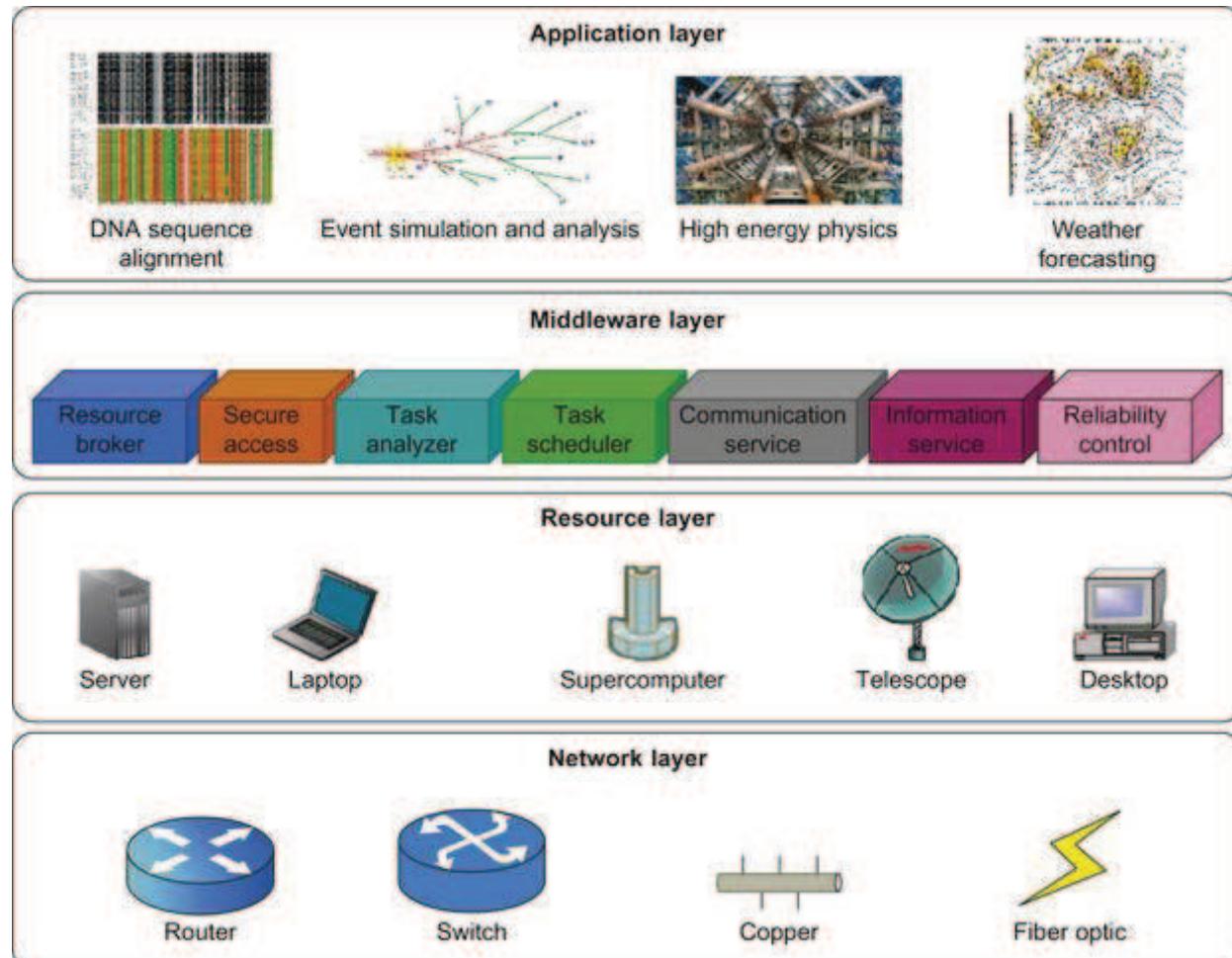
### 1.5.4.1 Energy Consumption of Unused Servers

To run a server farm (data center) a company has to spend a huge amount of money for hardware, software, operational support, and energy every year. Therefore, companies should thoroughly identify whether their installed server farm (more specifically, the volume of provisioned resources) is at an appropriate level, particularly in terms of utilization. It was estimated in the past that, on average, one-sixth (15 percent) of the full-time servers in a company are left powered on without being actively used (i.e., they are idling) on a daily basis. This indicates that with 44 million servers in the world, around 4.7 million servers are not doing any useful work.

The potential savings in turning off these servers are large—\$3.8 billion globally in energy costs alone, and \$24.7 billion in the total cost of running nonproductive servers, according to a study by 1E Company in partnership with the Alliance to Save Energy (ASE). This amount of wasted energy is equal to 11.8 million tons of carbon dioxide per year, which is equivalent to the CO pollution of 2.1 million cars. In the United States, this equals 3.17 million tons of carbon dioxide, or 580,678 cars. Therefore, the first step in IT departments is to analyze their servers to find unused and/or underutilized servers.

### 1.5.4.2 Reducing Energy in Active Servers

In addition to identifying unused/underutilized servers for energy savings, it is also necessary to apply appropriate techniques to decrease energy consumption in active distributed systems with negligible influence on their performance. Power management issues in distributed computing platforms can be categorized into four layers (see [Figure 1.26](#)): the application layer, middleware layer, resource layer, and network layer.



**FIGURE 1.26** Four operational layers of distributed computing systems. Courtesy of Zomaya, Rivandi and Lee of the University of Sydney [33]

### 1.5.4.3 Application Layer

Until now, most user applications in science, business, engineering, and financial areas

tend to increase a system's speed or quality. By introducing energy-aware applications, the challenge is to design sophisticated multilevel and multi-domain energy management applications without hurting performance. The first step toward this end is to explore a relationship between performance and energy consumption. Indeed, an application's energy consumption depends strongly on the number of instructions needed to execute the application and the number of transactions with the storage unit (or memory). These two factors (compute and storage) are correlated and they affect completion time.

#### **1.5.4.4 Middleware Layer**

The middleware layer acts as a bridge between the application layer and the resource layer. This layer provides resource broker, communication service, task analyzer, task scheduler, security access, reliability control, and information service capabilities. It is also responsible for applying energy-efficient techniques, particularly in task scheduling. Until recently, scheduling was aimed at minimizing *makespan*, that is, the execution time of a set of tasks. Distributed computing systems necessitate a new cost function covering both makespan and energy consumption.

#### **1.5.4.5 Resource Layer**

The resource layer consists of a wide range of resources including computing nodes and storage units. This layer generally interacts with hardware devices and the operating system; therefore, it is responsible for controlling all distributed resources in distributed computing systems. In the recent past, several mechanisms have been developed for more efficient power management of hardware and operating systems. The majority of them are hardware approaches particularly for processors.

*Dynamic power management (DPM)* and *dynamic voltage-frequency scaling (DVFS)* are two popular methods incorporated into recent computer hardware systems [21]. In DPM, hardware devices, such as the CPU, have the capability to switch from idle mode to one or more lower-power modes. In DVFS, energy savings are achieved based on the fact that the power consumption in CMOS circuits has a direct relationship with frequency and the square of the voltage supply. Execution time and power consumption are controllable by switching among different frequencies and voltages [31].

#### **1.5.4.6 Network Layer**

Routing and transferring packets and enabling network services to the resource layer are the main responsibility of the network layer in distributed computing systems. The major challenge to build energy-efficient networks is, again, determining how to measure, predict, and create a balance between energy consumption and performance. Two major challenges to designing energy-efficient networks are:

- The models should represent the networks comprehensively as they should give a full understanding of interactions among time, space, and energy.
- New, energy-efficient routing algorithms need to be developed. New, energy-efficient

protocols should be developed against network attacks.

As information resources drive economic and social development, data centers become increasingly important in terms of where the information items are stored and processed, and where services are provided. Data centers become another core infrastructure, just like the power grid and transportation systems. Traditional data centers suffer from high construction and operational costs, complex resource management, poor usability, low security and reliability, and huge energy consumption. It is necessary to adopt new technologies in next-generation data-center designs, a topic we will discuss in more detail in [Chapter 4](#).

#### **1.5.4.7 DVFS Method for Energy Efficiency**

The DVFS method enables the exploitation of the slack time (idle time) typically incurred by inter-task relationship. Specifically, the slack time associated with a task is utilized to execute the task in a lower voltage frequency. The relationship between energy and voltage frequency in CMOS circuits is related by:

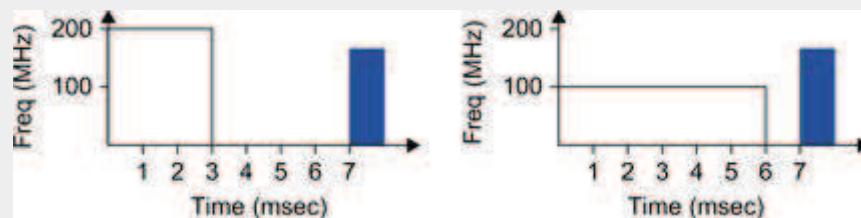
(1.6)

where  $v$ ,  $C_{eff}$ ,  $K$ , and  $v_t$  are the voltage, circuit switching capacity, a technology dependent factor, and threshold voltage, respectively, and the parameter  $t$  is the execution time of the task under clock frequency  $f$ . By reducing voltage and frequency, the device's energy consumption can also be reduced.

#### **Example 1.2**

#### **Energy Efficiency in Distributed Power Management**

[Figure 1.27](#) illustrates the DVFS method. This technique as shown on the right saves the energy compared to traditional practices shown on the left. The idea is to reduce the frequency and/or voltage during workload slack time. The transition latencies between lower-power modes are very small. Thus energy is saved by switching between operational modes. Switching between low-power modes affects performance. Storage units must interact with the computing nodes to balance power consumption. According to Ge, Feng, and Cameron [21], the storage devices are responsible for about 27 percent of the total energy consumption in a data center. This figure increases rapidly due to a 60 percent increase in storage needs annually, making the situation even worse.



**FIGURE 1.27** The DVFS technique (right) saves energy, compared to traditional practices (left) by reducing the frequency or voltage during slack time.

## 1.6 Bibliographic Notes and Homework Problems

Over the past four decades, parallel processing and distributed computing have been hot topics for research and development. Earlier work in parallel computing can be found in several classic books [14,17,27,28]. Recent coverage of distributed systems can be found in [8,13,20,22]. Cluster computing was covered in [2,10,28,38] and grid computing in [6,18,42,47,51]. P2P networks were introduced in [6,34,46]. Multicore CPUs and many-core GPU processors were discussed in [15,32,36]. Information on the Top 500 supercomputers can be found in [50].

Data centers are introduced in [4,19,26], and recent computer architecture in [24,26]. Cloud computing is studied in [1,9,11,18,29,30,39,44]. The edited volume [11] on cloud computing by Buyya, Broberg, and Goscinski serves as a good resource on cloud computing research. Chou's book [12] emphasizes business models for cloud services. Virtualization techniques were introduced in [40–44]. The articles by Bell, Gray, and Szalay [5]; Foster, et al. [18]; and Hey [25] address critical issues concerning data-intensive grid and cloud computing. Massive data parallelism and programming are covered in [14,32].

Distributed algorithms and MPI programming are studied in [3,12,15,22,28]. Distributed operating systems and software tools are covered in [3,7,13,46]. Energy efficiency and power management are studied in [21,31,52]. The Internet of Things is studied in [45,48]. The work of Hwang and Li [30] suggested ways to cope with cloud security and data protection problems. In subsequent chapters, additional references will be provided. The following list highlights some international conferences, magazines, and journals that often report on the latest developments in parallelism, clusters, grids, P2P systems, and cloud and distributed systems:

- **IEEE and Related Conference Publications** Internet Computing, TPDS (Transactions on Parallel and Distributed Systems), TC (Transactions on Computers), TON (Transactions on Networking), ICDCS (International Conference on Distributed Computing Systems), IPDPS (International Parallel and Distributed Processing Symposium), INFOCOM, GLOBECOM, CCGrid (Clusters, Clouds and The Grid), P2P Computing, HPDC (High-Performance Distributed Computing), CloudCom (International Conference on Cloud Computing Technology and Science), ISCA

(International Symposium on Computer Architecture), HPCA (High-Performance Computer Architecture), Computer Magazine, TDSC (Transactions on Dependable and Secure Computing), TKDE (Transactions on Knowledge and Data Engineering), HPCC (High Performance Computing and Communications), ICPADS (International Conference on Parallel and Distributed Applications and Systems), NAS (Networks, Architectures, and Storage), and GPC (Grid and Pervasive Computing)

- **ACM, Internet Society, IFIP, and Other Relevant Publications** Supercomputing Conference, ACM Transactions on Computing Systems, USENIX Technical Conference, JPDC (Journal of Parallel and Distributed Computing), Journal of Cloud Computing, Journal of Distributed Computing, Journal of Cluster Computing, Future Generation Computer Systems, Journal of Grid Computing, Journal of Parallel Computing, International Conference on Parallel Processing (ICPP), European Parallel Computing Conference (EuroPAR), Concurrency: Practice and Experiences (Wiley), NPC (IFIP Network and Parallel Computing), and PDCS (ISCA Parallel and Distributed Computer Systems)

## Acknowledgments

This chapter was authored by Kai Hwang primarily. Geoffrey Fox and Albert Zomaya have contributed to [Sections 1.4.1](#) and [1.5.4](#), respectively. Xiaosong Lou and Lihong Chen at the University of Southern California have assisted in plotting [Figures 1.4](#) and [1.10](#). Nikzad Babaii Rizvandi and Young-Choon Lee of the University of Sydney have contributed partially to [Section 1.5.4](#). Jack Dongarra has edited the entire Chapter 1.

## References

1. Amazon EC2 and S3, Elastic Compute Cloud (EC2) and Simple Scalable Storage (S3). [http://en.wikipedia.org/wiki/Amazon\\_Elastic\\_Compute\\_Cloud](http://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud) and [http://spatten\\_presentations.s3.amazonaws.com/s3-on-rails.pdf](http://spatten_presentations.s3.amazonaws.com/s3-on-rails.pdf).
2. Baker M, Buyya R. Cluster computing at a glance. In: Buyya R, ed. Upper Saddle River, NJ: Prentice-Hall; 1999;3–47. *High-Performance Cluster Computing, Architecture and Systems*. vol. 1 Chapter 1.
3. Barak A, Shiloh A. *The MOSIX Management System for Linux Clusters, Multi-Clusters, CPU Clusters, and Clouds*, White paper 2010; [www.MOSIX.org//txt\\_pub.html](http://www.MOSIX.org//txt_pub.html).
4. Barroso L, Holzle U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines* Morgan & Claypool Publishers 2009.
5. Bell G, Gray J, Szalay A. Petascale computational systems: balanced cyberstructure in a data-centric World. *IEEE Comput Mag* 2006.
6. Berman F, Fox G, Hey T, eds. *Grid Computing*. Wiley 2003.
7. Bever M, et al. Distributed systems, OSF DCE, and beyond. In: Schill A, ed. *DCE- The OSF Distributed Computing Environment*. Springer-Verlag 1993;1–20.
8. Birman K. *Reliable Distributed Systems: Technologies, Web Services, and Applications*

Springer-Verlag 2005.

9. Boss G, et al. *Cloud Computing—The BlueCloud Project* October 2007; [www.ibm.com/developerworks/websphere/zones/hipods/](http://www.ibm.com/developerworks/websphere/zones/hipods/).
10. *High-Performance Cluster Computing*. Vol. 1 and 2. Buyya R, ed. Englewood Cliffs, NJ: Prentice-Hall; 1999.
11. Buyya R, Broberg J, Goscinski A, eds. *Cloud Computing: Principles and Paradigms*. Wiley 2011.
12. Chou T. *Introduction to Cloud Computing: Business and Technology. Lecture Notes at Stanford University and Tsinghua University* Active Book Press 2010.
13. Coulouris G, Dollimore J, Kindberg T. *Distributed Systems: Concepts and Design*. Wesley 2005.
14. Culler D, Singh J, Gupta A. *Parallel Computer Architecture* Kaufmann Publishers 1999.
15. Dally B. In: *GPU Computing to Exascale and Beyond*. Keynote Address at ACM Supercomputing Conference. November 2010.
16. J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Proceedings of OSDI 2004. Also, Communication of ACM, Vol. 51, 2008, pp. 107–113.
17. Dongarra J, ed. *Source Book of Parallel Computing*. San Francisco: Morgan Kaufmann; 2003.
18. I. Foster, Y. Zhao, J. Raicu, S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, Grid Computing Environments Workshop, 12–16 November 2008.
19. D. Gannon, The Client+Cloud: Changing the Paradigm for Scientific Research, Keynote Address, IEEE CloudCom2010, Indianapolis, 2 November 2010.
20. Garg VK. *Elements of Distributed Computing* Wiley-IEEE Press 2002.
21. Ge R, Feng X, Cameron K. Performance Constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters. In: *Proceedings Supercomputing Conf.* Washington, DC. 2005.
22. Ghosh S. *Distributed Systems—An Algorithmic Approach* Chapman & Hall/CRC 2007.
23. He B, Fang W, Luo Q, Govindaraju N, Wang T. Mars: A MapReduce Framework on Graphics Processor. In: *ACM PACT'08*. Toronto, Canada. 25–29 October 2008.
24. Hennessy J, Patterson D. *Computer Architecture: A Quantitative Approach* Morgan Kaufmann 2007.
25. Hey T, et al. The Fourth Paradigm: Data-Intensive Scientific Discovery. *Microsoft Research* 2009.
26. Hill MD, et al. *The Data Center as a Computer* Morgan & Claypool Publishers 2009.
27. Hwang K. *Advanced Computer Architecture: Parallelism, Scalability, Programming* McGraw-Hill 1993.
28. Hwang K, Xu Z. *Scalable Parallel Computing* McGraw-Hill 1998.
29. Hwang K, Kulkarni S, Hu Y. Cloud Security with Virtualized Defense and Reputation-based Trust Management. In: *IEEE Conference on Dependable, Autonomous, and Secure Computing (DAC-2009)*. Chengdu, China. 14 December

2009.

30. Hwang K, Li D. Trusted Cloud Computing with Secure Resources and Data Coloring. In: *IEEE Internet Computing, Special Issue on Trust and Reputation Management*. September 2010;14–22.
31. Kelton Research, Server Energy & Efficiency Report. [www.1e.com/EnergyCampaign/downloads/Server\\_Energy\\_and\\_Efficiency\\_Report\\_2009.pdf](http://www.1e.com/EnergyCampaign/downloads/Server_Energy_and_Efficiency_Report_2009.pdf), September 2009.
32. Kirk D, Hwu W. *Programming Massively Processors: A Hands-on Approach* Morgan Kaufmann 2010.
33. Lee YC, Zomaya AY. A Novel State Transition Method for Metaheuristic-Based Scheduling in Heterogeneous Computing Systems. In: *IEEE Transactions on Parallel and Distributed Systems*. September 2008.
34. Li ZY, Xie G, Hwang K, Li ZC. Churn-Resilient Protocol for Massive Data Dissemination in P2P Networks. In: *IEEE Trans Parallel and Distributed Systems*. May 2011.
35. Lou X, Hwang K. Collusive Piracy Prevention in P2P Content Delivery Networks. In: *IEEE Trans on Computers*. July, 2009;970–983.
36. NVIDIA Corp. Fermi: NVIDIA's Next-Generation CUDA Compute Architecture, White paper, 2009.
37. Peleg D. *Distributed Computing: A Locality-Sensitive Approach* SIAM 2000.
38. Pfister GF. *In Search of Clusters* Second ed. Prentice-Hall 2001.
39. Qiu J, Gunarathne T, Ekanayake J, et al. Hybrid Cloud and Cluster Computing Paradigms for Life Science Applications. In: *11th Annual Bioinformatics Open Source Conference BOSC 2010*. 9–10 July 2010.
40. Rosenblum M, Garfinkel T. Virtual machine monitors: current technology and future trends. *IEEE Computer* 2005;(May):39–47.
41. M. Rosenblum, Recent Advances in Virtual Machines and Operating Systems, Keynote Address, ACM ASPLOS 2006.
42. Smith J, Nair R. *Virtual Machines* Morgan Kaufmann 2005.
43. B. Sotomayor, R. Montero, I. Foster, Virtual Infrastructure Management in Private and Hybrid Clouds, *IEEE Internet Computing*, September 2009.
44. SRI. The Internet of Things, in: Disruptive Technologies: Global Trends 2025, [www.dni.gov/nic/PDF\\_GIF\\_Confreports/disruptivetech/appendix\\_F.pdf](http://www.dni.gov/nic/PDF_GIF_Confreports/disruptivetech/appendix_F.pdf), 2010.
45. Tanenbaum A. *Distributed Operating Systems* Prentice-Hall 1995.
46. Taylor I. *From P2P to Web Services and Grids* London: Springer-Verlag; 2005.
47. Twister, Open Source Software for Iterative MapReduce, <http://www.iterativemapreduce.org/>.
48. Wikipedia. Internet of Things, [http://en.wikipedia.org/wiki/Internet\\_of\\_Things](http://en.wikipedia.org/wiki/Internet_of_Things), June 2010.
49. Wikipedia. CUDA, <http://en.wikipedia.org/wiki/CUDA>, March 2011.
50. Wikipedia. TOP500, <http://en.wikipedia.org/wiki/TOP500>, February 2011.
51. Wu Y, Hwang K, Yuan Y, Zheng W. Adaptive Workload Prediction of Grid Performance in Confidence Windows. In: *IEEE Trans on Parallel and Distributed Systems*.

## Homework Problems

### Problem 1.1

Briefly define the following basic techniques and technologies that represent recent related advances in computer architecture, parallel processing, distributed computing, Internet technology, and information services:

- a. High-performance computing (HPC) system
- b. High-throughput computing (HTC) system
- c. Peer-to-peer (P2P) network
- d. Computer cluster versus computational grid
- e. Service-oriented architecture (SOA)
- f. Pervasive computing versus Internet computing
- g. Virtual machine versus virtual infrastructure
- h. Public cloud versus private cloud
- i. Radio-frequency identifier (RFID)
- j. Global positioning system (GPS)
- k. Sensor network
- l. Internet of Things (IoT)
- m. Cyber-physical system (CPS)

### Problem 1.2

Circle only one correct answer in each of the following two questions:

- 1. In the 2009 Top 500 list of the fastest computer systems, which architecture dominates?
  - a. Symmetric shared-memory multiprocessor systems.
  - b. Centralized massively parallel processor (MPP) systems.
  - c. Clusters of cooperative computers.
- 2. In a cloud formed by a cluster of servers, all servers must be selected as follows:
  - a. All cloud machines must be built on physical servers.
  - b. All cloud machines must be built with virtual servers.
  - c. The cloud machines can be either physical or virtual servers.

### Problem 1.3

An increasing number of organizations in industry and business sectors adopt cloud systems. Answer the following questions regarding cloud computing:

- a. List and describe the main characteristics of cloud computing systems.
- b. Discuss key enabling technologies in cloud computing systems.
- c. Discuss different ways for cloud service providers to maximize their revenues.

## Problem 1.4

Match 10 abbreviated terms and system models on the left with their descriptions on the right. Enter the description label (**a**, **b**, **c**, ..., **j**) in the underlined blanks in front of the terms.

<u>Globus</u>	(a) A scalable software platform promoted by Apache for web users to write and run applications over vast amounts of distributed data
<u>BitTorrent</u>	(b) A P2P network for MP3 music delivery with a centralized directory server
<u>Gnutella</u>	(c) The programming model and associated implementation by Google for distributed mapping and reduction of very large data sets
<u>EC2</u>	(d) A middleware library jointly developed by USC/ISI and Argonne National Lab for grid resource management and job scheduling
<u>TeraGrid</u>	(e) A distributed storage program by Google for managing structured data that can scale to very large sizes
<u>EGEE</u>	(f) A P2P file-sharing network using multiple file index trackers
<u>Hadoop</u>	(g) A critical design goal of clusters of computers to tolerate nodal faults or recovery from host failures
<u>SETI@home</u>	(h) The service architecture specification as an open grid standard
<u>Napster</u>	(i) An elastic and flexible computing environment that allows web application developers to acquire cloud resources effectively
<u>BigTable</u>	(j) A P2P grid over 3 million desktops for distributed signal processing in search of extraterrestrial intelligence

## Problem 1.5

Consider a multicore processor with four heterogeneous cores labeled A, B, C, and D. Assume cores A and D have the same speed. Core B runs twice as fast as core A, and core C runs three times faster than core A. Assume that all four cores start executing the following application at the same time and no cache misses are encountered in all core operations. Suppose an application needs to compute the square of each element of an array of 256 elements. Assume 1 unit time for core A or D to compute the square of an element. Thus, core B takes    unit time and core C takes    unit time to compute the square of an element. Given the following division of labor in four cores:

Core A	32 elements
Core B	128 elements
Core C	64 elements
Core D	32 elements

- Compute the *total execution time* (in time units) for using the four-core processor to compute the squares of 256 elements in parallel. The four cores have different speeds. Some faster cores finish the job and may become idle, while others are still busy computing until all squares are computed.
- Calculate the *processor utilization rate*, which is the total amount of time the cores are busy (not idle) divided by the total execution time they are using all cores in the processor to execute the above application.

## Problem 1.6

Consider parallel execution of an MPI-coded C program in SPMD (single program and multiple data streams) mode on a server cluster consisting of  $n$  identical Linux servers. SPMD mode means the same MPI program is running simultaneously on all servers but over different data sets of identical workloads. Assume that 25 percent of the program execution is attributed to the execution of MPI commands. For simplicity, assume that all MPI commands take the same amount of execution time. Answer the following questions using Amdahl's law:

- a. Given that the total execution time of the MPI program on a four-server cluster is  $T$  minutes, what is the speedup factor of executing the same MPI program on a 256-server cluster, compared with using the four-server cluster? Assume that the program execution is deadlock-free and ignore all other runtime execution overheads in the calculation.
- b. Suppose that all MPI commands are now enhanced by a factor of 2 by using active messages executed by message handlers at the user space. The enhancement can reduce the execution time of all MPI commands by half. What is the speedup of the 256-server cluster installed with this MPI enhancement, computed with the old 256-server cluster without MPI enhancement?

## Problem 1.7

Consider a program for multiplying two large-scale  $N \times N$  matrices, where  $N$  is the matrix size. The sequential multiply time on a single server is  $T_1 = cN^3$  minutes, where  $c$  is a constant determined by the server used. An MPI-code parallel program requires  $T_n = cN^3/n + dN^2/n^{0.5}$  minutes to complete execution on an  $n$ -server cluster system, where  $d$  is a constant determined by the MPI version used. Assume the program has a zero sequential bottleneck ( $\alpha = 0$ ). The second term in  $T_n$  accounts for the total message-passing overhead experienced by  $n$  servers.

Answer the following questions for a given cluster configuration with  $n = 64$  servers,  $c = 0.8$ , and  $d = 0.1$ . Parts (a, b) have a fixed workload corresponding to the matrix size  $N = 15,000$ . Parts (c, d) have a scaled workload associated with an enlarged matrix size  $N' = n^{1/3} N = 64^{1/3} \times 15,000 = 4 \times 15,000 = 60,000$ . Assume the same cluster configuration to process both workloads. Thus, the system parameters  $n$ ,  $c$ , and  $d$  stay unchanged. Running the scaled workload, the overhead also increases with the enlarged matrix size  $N'$ .

- a. Using Amdahl's law, calculate the speedup of the  $n$ -server cluster over a single server.
- b. What is the efficiency of the cluster system used in Part (a)?
- c. Calculate the speedup in executing the scaled workload for an enlarged  $N' \times N'$  matrix on the same cluster configuration using Gustafson's law.
- d. Calculate the efficiency of running the scaled workload in Part (c) on the 64-processor cluster.
- e. Compare the above speedup and efficiency results and comment on their implications.

## Problem 1.8

Compare the similarities and differences between traditional computing clusters/grids and the computing clouds launched in recent years. Consider all technical and economic aspects as listed below. Answer the following questions against real example systems or platforms built in recent years. Also discuss the possible convergence of the two computing paradigms in the future.

- a. Hardware, software, and networking support
- b. Resource allocation and provisioning methods
- c. Infrastructure management and protection
- d. Support of utility computing services
- e. Operational and cost models applied

## Problem 1.9

Answer the following questions regarding PC and HPC systems:

- a. Explain why PCs and HPCs were evolutionary rather than revolutionary in the past 30 years.
- b. Discuss the drawbacks in disruptive changes in processor architecture. Why is the memory wall a major problem in achieving scalable changes in performance?
- c. Explain why x-86 processors are still dominating the PC and HPC markets.

## Problem 1.10

Multicore and many-core processors have appeared in widespread use in both desktop computers and HPC systems. Answer the following questions regarding advanced processors, memory devices, and system interconnects:

- a. What are the differences between multicore CPUs and GPUs in terms of architecture and usage?
- b. Explain why parallel programming cannot match the progress of processor technology.
- c. Suggest ideas and defend your argument with some plausible solutions to this mismatch problem between core scaling and effective programming and use of multicores.
- d. Explain why flash memory SSD can deliver better speedups in some HPC or HTC applications.
- e. Justify the prediction that InfiniBand and Ethernet will continue to dominate the HPC market.

## Problem 1.11

In [Figure 1.7](#), you studied five categories of modern processors. Characterize in [Table 1.9](#) five micro-architectures for designing these processors. Comment on their advantages/shortcomings and identify the names of two example commercial processors that are built in each processor category. Assume a single core in the superscalar

processor and the three multithreaded processors. The last processor category is a multicore CMP and each core is assumed to handle one thread at a time.

**Table 1.9**

**Comparison of Five Micro-architectures for Modern Processors**

Processor Micro-architectures	Architecture Characteristics	Advantages/Shortcomings	Representative Processors
Single-threaded Superscalar			
Fine-grain Multithreading			
Coarse-grain Multithreading			
Simultaneous Multithreading (SMT)			
Multicore Chip Multiprocessor (CMP)			

## Problem 1.12

Discuss the major advantages and disadvantages in the following areas:

- Why are virtual machines and virtual clusters suggested in cloud computing systems?
- What breakthroughs are required to build virtualized cloud systems cost-effectively?
- What are the impacts of cloud platforms on the future of the HPC and HTC industry?

## Problem 1.13

Characterize the following three cloud computing models:

- What is an IaaS (Infrastructure-as-a-Service) cloud? Give one example system.
- What is a PaaS (Platform-as-a-Service) cloud? Give one example system.
- What is a SaaS (Software-as-a-Service) cloud? Give one example system.

## Problem 1.14

Briefly explain each of the following cloud computing services. Identify two cloud providers by company name in each service category.

- Application cloud services
- Platform cloud services
- Compute and storage services
- Collocation cloud services
- Network cloud services

## Problem 1.15

Briefly explain the following terms associated with network threats or security defense in a distributed computing system:

- Denial of service (DoS)

- b. Trojan horse
- c. Network worm
- d. Service spoofing
- e. Authorization
- f. Authentication
- g. Data integrity
- h. Confidentiality

## Problem 1.16

Briefly answer the following questions regarding green information technology and energy efficiency in distributed systems:

- a. Why is power consumption critical to data-center operations?
- b. What constitutes the dynamic voltage frequency scaling (DVFS) technique?
- c. Conduct in-depth research on recent progress in green IT research, and write a report on its applications to data-center design and cloud service applications.

## Problem 1.17

Compare GPU and CPU chips in terms of their strengths and weaknesses. In particular, discuss the trade-offs between power efficiency, programmability, and performance. Also compare various MPP architectures in processor selection, performance target, efficiency, and packaging constraints.

## Problem 1.18

Compare three distributed operating systems: Amoeba, DCE, and MOSIX. Research their recent developments and their impact on applications in clusters, grids, and clouds. Discuss the suitability of each system in its commercial or experimental distributed applications. Also discuss each system's limitations and explain why they were not successful as commercial systems.

## CHAPTER 2

# Computer Clusters for Scalable Parallel Computing

## CHAPTER OUTLINE

### Summary

#### 2.1 Clustering for Massive Parallelism

- 2.1.1 Cluster Development Trends
- 2.1.2 Design Objectives of Computer Clusters
- 2.1.3 Fundamental Cluster Design Issues
- 2.1.4 Analysis of the Top 500 Supercomputers

#### 2.2 Computer Clusters and MPP Architectures

- 2.2.1 Cluster Organization and Resource Sharing
- 2.2.2 Node Architectures and MPP Packaging
- 2.2.3 Cluster System Interconnects
- 2.2.4 Hardware, Software, and Middleware Support
- 2.2.5 GPU Clusters for Massive Parallelism

#### 2.3 Design Principles of Computer Clusters

- 2.3.1 Single-System Image Features
- 2.3.2 High Availability through Redundancy
- 2.3.3 Fault-Tolerant Cluster Configurations
- 2.3.4 Checkpointing and Recovery Techniques

#### 2.4 Cluster Job and Resource Management

- 2.4.1 Cluster Job Scheduling Methods
- 2.4.2 Cluster Job Management Systems
- 2.4.3 Load Sharing Facility (LSF) for Cluster Computing
- 2.4.4 MOSIX: An OS for Linux Clusters and Clouds

#### 2.5 Case Studies of Top Supercomputer Systems

- 2.5.1 Tianhe-1A: The World Fastest Supercomputer in 2010
- 2.5.2 Cray XT5 Jaguar: The Top Supercomputer in 2009
- 2.5.3 IBM Roadrunner: The Top Supercomputer in 2008

#### 2.6 Bibliographic Notes and Homework Problems

[Acknowledgments](#)

[References](#)

[Homework Problems](#)

Clustering of computers enables scalable parallel and distributed computing in both science and business applications. This chapter is devoted to building cluster-structured massively parallel processors. We focus on the design principles and assessment of the hardware, software, middleware, and operating system support to achieve scalability, availability, programmability, single-system images, and fault tolerance in clusters. We will examine the cluster architectures of Tianhe-1A, Cray XT5 Jaguar, and IBM Roadrunner. The study also covers the LSF middleware and MOSIX/OS for job and resource management in Linux clusters, GPU clusters and cluster extensions to building grids and clouds. Only physical clusters are studied in this chapter. Virtual clusters will be studied in [Chapters 3 and 4](#).

clusters, massive parallelism, high availability, supercomputers, fault tolerance, resource management

## Summary

Clustering of computers enables scalable parallel and distributed computing in both science and business applications. This chapter is devoted to building cluster-structured massively parallel processors. We focus on the design principles and assessment of the hardware, software, middleware, and operating system support to achieve scalability, availability, programmability, single-system images, and fault tolerance in clusters. We will examine the cluster architectures of Tianhe-1A, Cray XT5 Jaguar, and IBM Roadrunner. The study also covers the LSF middleware and MOSIX/OS for job and resource management in Linux clusters, GPU clusters and cluster extensions to building grids and clouds. Only physical clusters are studied in this chapter. Virtual clusters will be studied in [Chapters 3 and 4](#).

## 2.1 Clustering for Massive Parallelism

A *computer cluster* is a collection of interconnected stand-alone computers which can work together collectively and cooperatively as a single integrated computing resource pool. Clustering explores massive parallelism at the job level and achieves *high availability* (HA) through stand-alone operations. The benefits of computer clusters and *massively parallel processors* (MPPs) include scalable performance, HA, fault tolerance, modular growth, and use of commodity components. These features can sustain the generation changes experienced in hardware, software, and network components. Cluster computing became popular in the mid-1990s as traditional mainframes and vector supercomputers were proven to be less cost-effective in many *high-performance computing* (HPC)

applications.

Of the Top 500 supercomputers reported in 2010, 85 percent were computer clusters or MPPs built with homogeneous nodes. Computer clusters have laid the foundation for today's supercomputers, computational grids, and Internet clouds built over data centers. We have come a long way toward becoming addicted to computers. According to a recent IDC prediction, the HPC market will increase from \$8.5 billion in 2010 to \$10.5 billion by 2013. A majority of the Top 500 supercomputers are used for HPC applications in science and engineering. Meanwhile, the use of *high-throughput computing* (HTC) clusters of servers is growing rapidly in business and web services applications.

## 2.1.1 Cluster Development Trends

Support for clustering of computers has moved from interconnecting high-end mainframe computers to building clusters with massive numbers of x86 engines. Computer clustering started with the linking of large mainframe computers such as the IBM Sysplex and the SGI Origin 3000. Originally, this was motivated by a demand for cooperative group computing and to provide higher availability in critical enterprise applications. Subsequently, the clustering trend moved toward the networking of many minicomputers, such as DEC's VMS cluster, in which multiple VAXes were interconnected to share the same set of disk/tape controllers. Tandem's Himalaya was designed as a business cluster for fault-tolerant *online transaction processing* (OLTP) applications.

In the early 1990s, the next move was to build UNIX-based workstation clusters represented by the Berkeley NOW (Network of Workstations) and IBM SP2 AIX-based server cluster. Beyond 2000, we see the trend moving to the clustering of RISC or x86 PC engines. Clustered products now appear as integrated systems, software tools, availability infrastructure, and operating system extensions. This clustering trend matches the downsizing trend in the computer industry. Supporting clusters of smaller nodes will increase sales by allowing modular incremental growth in cluster configurations. From IBM, DEC, Sun, and SGI to Compaq and Dell, the computer industry has leveraged clustering of low-cost servers or x86 desktops for their cost-effectiveness, scalability, and HA features.

### 2.1.1.1 Milestone Cluster Systems

Clustering has been a hot research challenge in computer architecture. Fast communication, job scheduling, SSI, and HA are active areas in cluster research. [Table 2.1](#) lists some milestone cluster research projects and commercial cluster products. Details of these old clusters can be found in [14]. These milestone projects have pioneered clustering hardware and middleware development over the past two decades. Each cluster project listed has developed some unique features. Modern clusters are headed toward HPC clusters as studied in [Section 2.5](#).

Project	Special Features That Support Clustering
DEC VAXcluster (1991)	A UNIX cluster of symmetric multiprocessing (SMP) servers running the VMS OS with extensions, mainly used in HA applications
U.C. Berkeley NOW Project (1995)	A serverless network of workstations featuring active messaging, cooperative filing, and GLUnix development
Rice University TreadMarks (1996)	Software-implemented distributed shared memory for use in clusters of UNIX workstations based on page migration
Sun Solaris MC Cluster (1995)	A research cluster built over Sun Solaris workstations; some cluster OS functions were developed but were never marketed successfully
Tandem Himalaya Cluster (1994)	A scalable and fault-tolerant cluster for OLTP and database processing, built with nonstop operating system support
IBM SP2 Server Cluster (1996)	An AIX server cluster built with Power2 nodes and the Omega network, and supported by IBM LoadLeveler and MPI extensions
Google Search Engine Cluster (2003)	A 4,000-node server cluster built for Internet search and web service applications, supported by a distributed file system and fault tolerance
MOSIX (2010) <a href="http://www.mosix.org">www.mosix.org</a>	A distributed operating system for use in Linux clusters, multiclusters, grids, and clouds; used by the research community

The NOW project addresses a whole spectrum of cluster computing issues, including architecture, software support for web servers, single system image, I/O and file system, efficient communication, and enhanced availability. The Rice University TreadMarks is a good example of software-implemented shared-memory cluster of workstations. The memory sharing is implemented with a user-space runtime library. This was a research cluster built over Sun Solaris workstations. Some cluster OS functions were developed, but were never marketed successfully.

A Unix cluster of SMP servers running VMS/OS with extensions, mainly used in high-availability applications. An AIX server cluster built with Power2 nodes and Omega network and supported by IBM Loadleveler and MPI extensions. A scalable and fault-tolerant cluster for OLTP and database processing built with non-stop operating system support. The Google search engine was built at Google using commodity components. MOSIX is a distributed operating systems for use in Linux clusters, multi-clusters, grids, and the clouds, originally developed by Hebrew University in 1999.

## 2.1.2 Design Objectives of Computer Clusters

Clusters have been classified in various ways in the literature. We classify clusters using six orthogonal attributes: *scalability, packaging, control, homogeneity, programmability, and security*.

### 2.1.2.1 Scalability

Clustering of computers is based on the concept of modular growth. To scale a cluster from hundreds of uniprocessor nodes to a supercluster with 10,000 multicore nodes is a nontrivial task. The scalability could be limited by a number of factors, such as the

multicore chip technology, cluster topology, packaging method, power consumption, and cooling scheme applied. The purpose is to achieve scalable performance constrained by the aforementioned factors. We have to also consider other limiting factors such as the memory wall, disk I/O bottlenecks, and latency tolerance, among others.

### **2.1.2.2 Packaging**

Cluster nodes can be packaged in a compact or a slack fashion. In a *compact* cluster, the nodes are closely packaged in one or more racks sitting in a room, and the nodes are not attached to peripherals (monitors, keyboards, mice, etc.). In a *slack* cluster, the nodes are attached to their usual peripherals (i.e., they are complete SMPs, workstations, and PCs), and they may be located in different rooms, different buildings, or even remote regions. Packaging directly affects communication wire length, and thus the selection of interconnection technology used. While a compact cluster can utilize a high-bandwidth, low-latency communication network that is often proprietary, nodes of a slack cluster are normally connected through standard LANs or WANs.

### **2.1.2.3 Control**

A cluster can be either controlled or managed in a *centralized* or *decentralized* fashion. A compact cluster normally has centralized control, while a slack cluster can be controlled either way. In a centralized cluster, all the nodes are owned, controlled, managed, and administered by a central operator. In a decentralized cluster, the nodes have individual owners. For instance, consider a cluster comprising an interconnected set of desktop workstations in a department, where each workstation is individually owned by an employee. The owner can reconfigure, upgrade, or even shut down the workstation at any time. This lack of a single point of control makes system administration of such a cluster very difficult. It also calls for special techniques for process scheduling, workload migration, checkpointing, accounting, and other similar tasks.

### **2.1.2.4 Homogeneity**

A *homogeneous* cluster uses nodes from the same platform, that is, the same processor architecture and the same operating system; often, the nodes are from the same vendors. A *heterogeneous* cluster uses nodes of different platforms. Interoperability is an important issue in heterogeneous clusters. For instance, process migration is often needed for load balancing or availability. In a homogeneous cluster, a binary process image can migrate to another node and continue execution. This is not feasible in a heterogeneous cluster, as the binary code will not be executable when the process migrates to a node of a different platform.

### **2.1.2.5 Security**

Intracluster communication can be either *exposed* or *enclosed*. In an exposed cluster, the communication paths among the nodes are exposed to the outside world. An outside machine can access the communication paths, and thus individual nodes, using standard

protocols (e.g., TCP/IP). Such exposed clusters are easy to implement, but have several disadvantages:

- Being exposed, intracluster communication is not secure, unless the communication subsystem performs additional work to ensure privacy and security.
- Outside communications may disrupt intracluster communications in an unpredictable fashion. For instance, heavy BBS traffic may disrupt production jobs.
- Standard communication protocols tend to have high overhead.

In an enclosed cluster, intracluster communication is shielded from the outside world, which alleviates the aforementioned problems. A disadvantage is that there is currently no standard for efficient, enclosed intracluster communication. Consequently, most commercial or academic clusters realize fast communications through one-of-a-kind protocols.

### **2.1.2.6 Dedicated versus Enterprise Clusters**

A *dedicated cluster* is typically installed in a deskside rack in a central computer room. It is homogeneously configured with the same type of computer nodes and managed by a single administrator group like a frontend host. Dedicated clusters are used as substitutes for traditional mainframes or supercomputers. A dedicated cluster is installed, used, and administered as a single machine. Many users can log in to the cluster to execute both interactive and batch jobs. The cluster offers much enhanced throughput, as well as reduced response time.

An *enterprise cluster* is mainly used to utilize idle resources in the nodes. Each node is usually a full-fledged SMP, workstation, or PC, with all the necessary peripherals attached. The nodes are typically geographically distributed, and are not necessarily in the same room or even in the same building. The nodes are individually owned by multiple owners. The cluster administrator has only limited control over the nodes, as a node can be turned off at any time by its owner. The owner's "local" jobs have higher priority than enterprise jobs. The cluster is often configured with heterogeneous computer nodes. The nodes are often connected through a low-cost Ethernet network. Most data centers are structured with clusters of low-cost servers. Virtual clusters play a crucial role in upgrading data centers. We will discuss virtual clusters in [Chapter 6](#) and clouds in [Chapters 7, 8, and 9](#).

### **2.1.3 Fundamental Cluster Design Issues**

In this section, we will classify various cluster and MPP families. Then we will identify the major design issues of clustered and MPP systems. Both physical and virtual clusters are covered. These systems are often found in computational grids, national laboratories, business data centers, supercomputer sites, and virtualized cloud platforms. A good understanding of how clusters and MPPs work collectively will pave the way toward understanding the ins and outs of large-scale grids and Internet clouds in subsequent chapters. Several issues must be considered in developing and using a cluster. Although much work has been done in this regard, this is still an active research and development

area.

### **2.1.3.1 Scalable Performance**

This refers to the fact that scaling of resources (cluster nodes, memory capacity, I/O bandwidth, etc.) leads to a proportional increase in performance. Of course, both scale-up and scale-down capabilities are needed, depending on application demand or cost-effectiveness considerations. Clustering is driven by scalability. One should not ignore this factor in all applications of cluster or MPP computing systems.

### **2.1.3.2 Single-System Image (SSI)**

A set of workstations connected by an Ethernet network is not necessarily a cluster. A cluster is a single system. For example, suppose a workstation has a 300 Mflops/second processor, 512 MB of memory, and a 4 GB disk and can support 50 active users and 1,000 processes. By clustering 100 such workstations, can we get a single system that is equivalent to one huge workstation, or a *megastation*, that has a 30 Gflops/second processor, 50 GB of memory, and a 400 GB disk and can support 5,000 active users and 100,000 processes? This is an appealing goal, but it is very difficult to achieve. SSI techniques are aimed at achieving this goal.

### **2.1.3.3 Availability Support**

Clusters can provide cost-effective HA capability with lots of redundancy in processors, memory, disks, I/O devices, networks, and operating system images. However, to realize this potential, availability techniques are required. We will illustrate these techniques later in the book, when we discuss how DEC clusters (Section 10.4) and the IBM SP2 (Section 10.3) attempt to achieve HA.

### **2.1.3.4 Cluster Job Management**

Clusters try to achieve high system utilization from traditional workstations or PC nodes that are normally not highly utilized. Job management software is required to provide batching, load balancing, parallel processing, and other functionality. We will study cluster job management systems in [Section 3.4](#). Special software tools are needed to manage multiple jobs simultaneously.

### **2.1.3.5 Internode Communication**

Because of their higher node complexity, cluster nodes cannot be packaged as compactly as MPP nodes. The internode physical wire lengths are longer in a cluster than in an MPP. This is true even for centralized clusters. A long wire implies greater interconnect network latency. But more importantly, longer wires have more problems in terms of reliability, clock skew, and cross talking. These problems call for reliable and secure communication protocols, which increase overhead. Clusters often use commodity networks (e.g., Ethernet) with standard protocols such as TCP/IP.

## 2.1.3.6 Fault Tolerance and Recovery

Clusters of machines can be designed to eliminate all single points of failure. Through redundancy, a cluster can tolerate faulty conditions up to a certain extent. Heartbeat mechanisms can be installed to monitor the running condition of all nodes. In case of a node failure, critical jobs running on the failing nodes can be saved by failing over to the surviving node machines. Rollback recovery schemes restore the computing results through periodic checkpointing.

## 2.1.3.7 Cluster Family Classification

Based on application demand, computer clusters are divided into three classes:

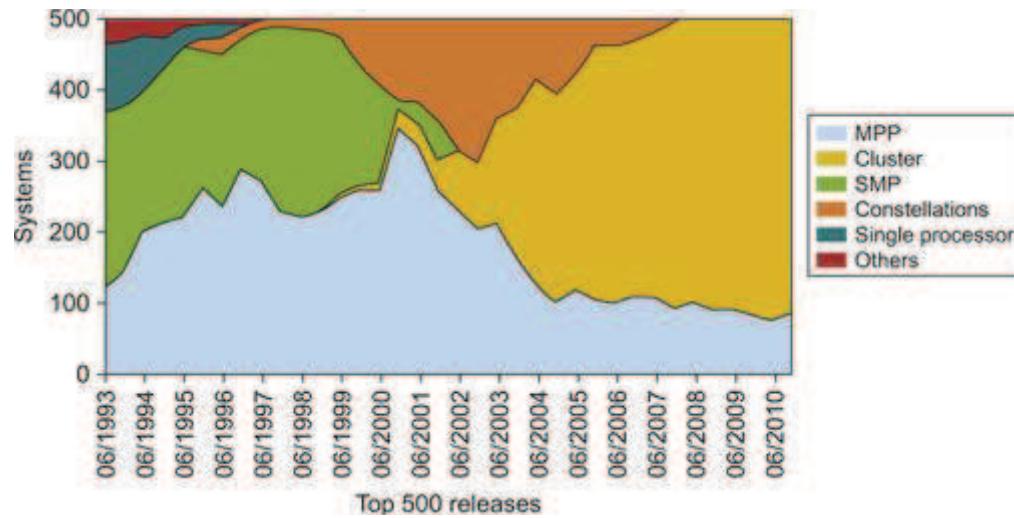
- **Compute clusters** These are clusters designed mainly for collective computation over a single large job. A good example is a cluster dedicated to numerical simulation of weather conditions. The compute clusters do not handle many I/O operations, such as database services. When a single compute job requires frequent communication among the cluster nodes, the cluster must share a dedicated network, and thus the nodes are mostly homogeneous and tightly coupled. This type of clusters is also known as a *Beowulf cluster*.  
When the nodes require internode communication over a small number of heavy-duty nodes, they are essentially known as a *computational grid*. Tightly coupled compute clusters are designed for supercomputing applications. Compute clusters apply middleware such as a message-passing interface (**MPI**) or Parallel Virtual Machine (**PVM**) to port programs to a wide variety of clusters.
- **High-Availability clusters** HA (high-availability) clusters are designed to be fault-tolerant and achieve HA of services. HA clusters operate with many redundant nodes to sustain faults or failures. The simplest HA cluster has only two nodes that can fail over to each other. Of course, high redundancy provides higher availability. HA clusters should be designed to avoid all single points of failure. Many commercial HA clusters are available for various operating systems.
- **Load-balancing clusters** These clusters shoot for higher resource utilization through load balancing among all participating nodes in the cluster. All nodes share the workload or function as a single *virtual machine* (VM). Requests initiated from the user are distributed to all node computers to form a cluster. This results in a balanced workload among different machines, and thus higher resource utilization or higher performance. Middleware is needed to achieve dynamic load balancing by job or process migration among all the cluster nodes.

## 2.1.4 Analysis of the Top 500 Supercomputers

Every six months, the world's Top 500 supercomputers are evaluated by running the Linpack Benchmark program over very large data sets. The ranking varies from year to year, similar to a competition. In this section, we will analyze the historical share in architecture, speed, operating systems, countries, and applications over time. In addition, we will compare the top five fastest systems in 2010.

## 2.1.4.1 Architectural Evolution

It is interesting to observe in Figure 2.1 the architectural evolution of the Top 500 supercomputers over the years. In 1993, 250 systems assumed the SMP architecture and these SMP systems all disappeared in June of 2002. Most SMPs are built with shared memory and shared I/O devices. There were 120 MPP systems built in 1993, MPPs reached the peak of 350 systems in mid-2000, and dropped to less than 100 systems in 2010. The single instruction, multiple data (SIMD) machines disappeared in 1997. The cluster architecture appeared in a few systems in 1999. The cluster systems are now populated in the Top-500 list with more than 400 systems as the dominating architecture class.



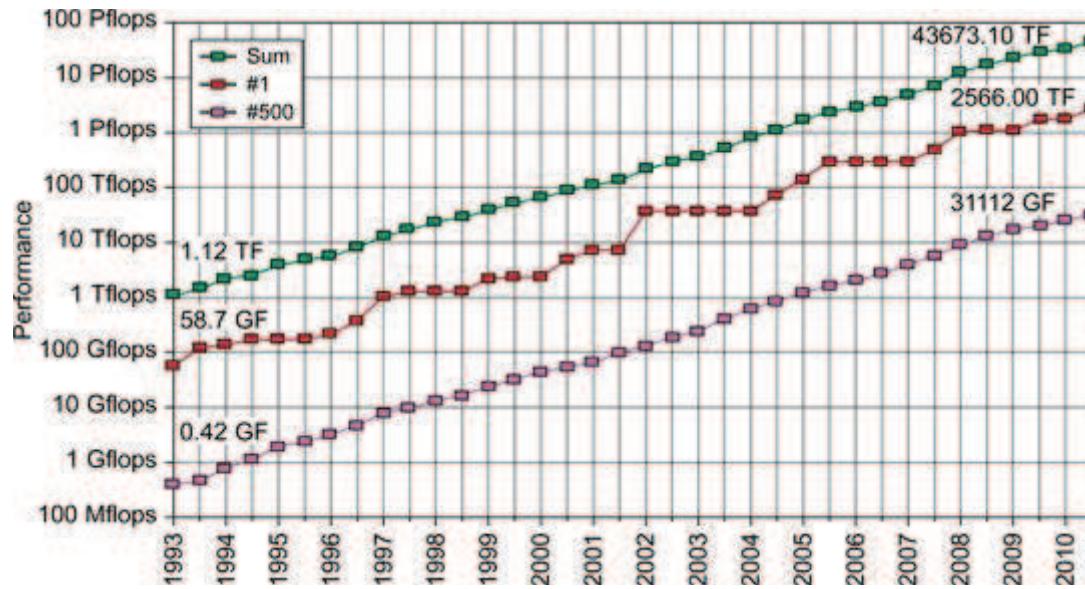
**FIGURE 2.1** Architectural share of the Top 500 systems. Courtesy of [www.top500.org](http://www.top500.org) [25]

In 2010, the Top 500 architecture is dominated by clusters (420 systems) and MPPs (80 systems). The basic distinction between these two classes lies in the components they use to build the systems. Clusters are often built with commodity hardware, software, and network components that are commercially available. MPPs are built with custom-designed compute nodes, boards, modules, and cabinets that are interconnected by special packaging. MPPs demand high bandwidth, low latency, better power efficiency, and high reliability. Cost-wise, clusters are affordable by allowing modular growth with scaling capability. The fact that MPPs appear in a much smaller quantity is due to their high cost. Typically, only a few MPP-based supercomputers are installed in each country.

## 2.1.4.2 Speed Improvement over Time

Figure 2.2 plots the measured performance of the Top 500 fastest computers from 1993 to 2010. The y-axis is scaled by the sustained speed performance in terms of Gflops, Tflops, and Pflops. The middle curve plots the performance of the fastest computers recorded over 17 years; peak performance increases from 58.7 Gflops to 2.566 Pflops. The bottom curve corresponds to the speed of the 500th computer, which increased from 0.42 Gflops in 1993 to 31.1 Tflops in 2010. The top curve plots the speed sum of all 500 computers

over the same period. In 2010, the total speed sum of 43.7 Pflops was achieved by all 500 computers, collectively. It is interesting to observe that the total speed sum increases almost linearly with time.



**FIGURE 2.2** Performance plot of the Top 500 supercomputers from 1993 to 2010. Courtesy of [www.top500.org](http://www.top500.org) [25]

### 2.1.4.3 Operating System Trends in the Top 500

The five most popular operating systems have more than a 10 percent share among the Top 500 computers, according to data released by TOP500.org ([www.top500.org/stats/list/36/os](http://www.top500.org/stats/list/36/os)) in November 2010. According to the data, 410 supercomputers are using Linux with a total processor count exceeding 4.5 million. This constitutes 82 percent of the systems adopting Linux. The IBM AIX/OS is in second place with 17 systems (a 3.4 percent share) and more than 94,288 processors. Third place is represented by the combined use of the SLEs10 with the SGI ProPack5, with 15 systems (3 percent) over 135,200 processors. Fourth place goes to the CNK/SLES9 used by 14 systems (2.8 percent) over 1.13 million processors. Finally, the CNL/OS was used in 10 systems (2 percent) over 178,577 processors. The remaining 34 systems applied 13 other operating systems with a total share of only 6.8 percent. In conclusion, the Linux OS dominates the systems in the Top 500 list.

### 2.1.4.4 The Top Five Systems in 2010

In Table 2.2, we summarize the key architecture features and sustained Linpack Benchmark performance of the top five supercomputers reported in November 2010. The Tianhe-1A was ranked as the fastest MPP in late 2010. This system was built with 86,386 Intel Xeon CPUs and NVIDIA GPUs by the National University of Defense Technology in China. We will present in Section 2.5 some of the top winners: namely the Tianhe-1A, Cray Jaguar, Nebulae, and IBM Roadrunner that were ranked among the top systems from 2008 to 2010. All the top five machines in Table 2.3 have achieved a speed higher

than 1 Pflops. The *sustained speed*,  $R_{max}$ , in Pflops is measured from the execution of the Linpack Benchmark program corresponding to a maximum matrix size.

**Table 2.2**

### The Top Five Supercomputers Evaluated in November 2010

System Name, Site, and URL	System Name, Processors, OS, Topology, and Developer	Linpack Speed ( $R_{max}$ ), Power	Efficiency ( $R_{max}/R_{peak}$ )
1. Tianhe-1A, National Supercomputing Center, Tianjin, China, <a href="http://www.nscc-tj.gov.cn/en/">http://www.nscc-tj.gov.cn/en/</a>	NUDT TH1A with 14,336 Xeon X5670 CPUs (six cores each) plus 7168 NVIDIA Tesla M2050 GPUs (448 CUDA cores each), running Linux, built by National Univ. of Defense Technology, China	2.57 Pflops, 4.02 MW	54.6% (over a peak of 4.7 Pflops)
2. Jaguar, DOE/SC/Oak Ridge National Lab., United States, <a href="http://computing.ornl.gov">http://computing.ornl.gov</a>	Cray XT5-HE: MPP with 224,162 x 6 AMD Opteron, 3D torus network, Linux (CLE), manufactured by Cray, Inc.	1.76 Pflops, 6.95 MW	75.6% (over a peak of 4.7 Pflops)
3. Nebulae at China's National Supercomputer Center, ShenZhen, China <a href="http://www.ict.cas.cas.cn">http://www.ict.cas.cas.cn</a>	TC3600 Blade, 120,640 cores in 55,680 Xeon X5650 plus 64,960 NVIDIA Tesla C2050 GPUs, Linux, InfiniBand, built by Dawning, Inc.	1.27 Pflops, 2.55 MW	42.6% (over a peak of 2.98 Pflops)
4. TSUBAME 2.0, GSIC Center, Tokyo Institute of Technology, Tokyo, Japan, <a href="http://www.gsic.titech.ac.jp/">http://www.gsic.titech.ac.jp/</a>	HP cluster, 3000SL, 73,278 x 6 Xeon X5670 processors, NVIDIA GPU, Linux/SLES 11, built by NEC/HP	1.19 Pflops, 1.8 MW	52.19% (over a peak of 2.3 Pflops)
5. Hopper, DOE/SC/LBNL/ NERSC, Berkeley, CA, USA, <a href="http://www.nersc.gov/">http://www.nersc.gov/</a>	Cray XE6 150,408 x 12 AMD Opteron, Linux (CLE), built by Cray, Inc.	1.05 Pflops, 2.8 MW	78.47% (over a peak of 1.35 Pflops)

**Table 2.3**

### Sample Compute Node Architectures for Large Cluster Construction

Node Architecture	Major Characteristics	Representative Systems
Homogeneous node using the same multicore processors	Multicore processors mounted on the same node with a crossbar connected to shared memory or local disks	The Cray XT5 uses two six-core AMD Opteron processors in each compute node
Hybrid nodes using CPU plus GPU or FLP accelerators	General-purpose CPU for integer operations, with GPUs acting as coprocessors to speed up FLP operations	China's Tianhe-1A uses two Intel Xeon processors plus one NVIDIA GPU per compute node

The *system efficiency* reflects the ratio of the *sustained speed* to the *peak speed*,  $R_{peak}$ , when all computing elements are fully utilized in the system. Among the top five systems, the two U.S.-built systems, the Jaguar and the Hopper, have the highest efficiency, more than 75 percent. The two systems built in China, the Tianhe-1A and the Nebulae, and Japan's

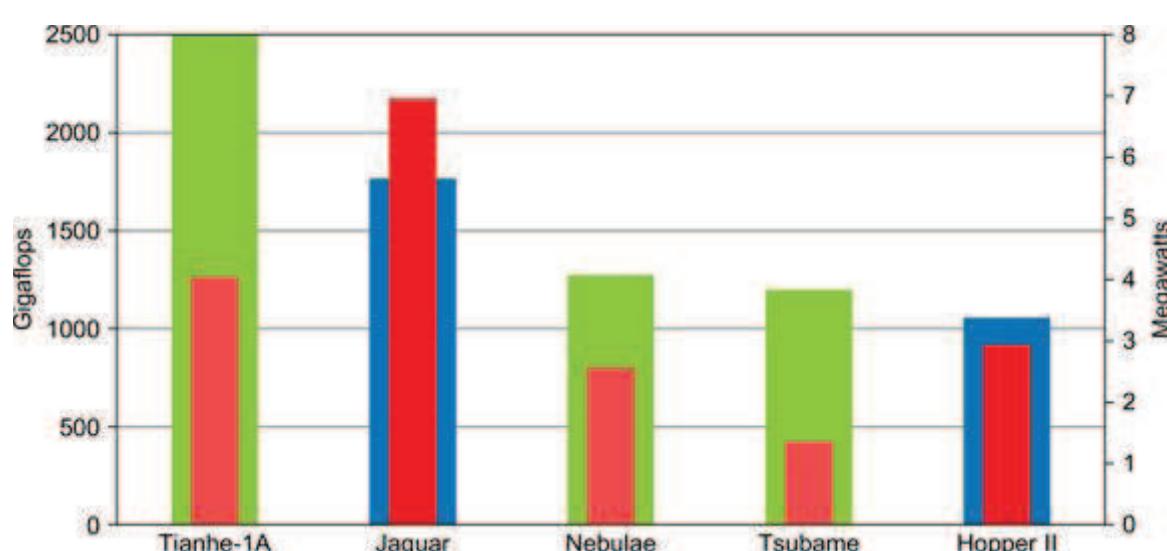
TSUBAME 2.0, are all low in efficiency. In other words, these systems should still have room for improvement in the future. The average power consumption in these 5 systems is 3.22 MW. This implies that excessive power consumption may post the limit to build even faster supercomputers in the future. These top systems all emphasize massive parallelism using up to 250,000 processor cores per system.

#### 2.1.4.5 Country Share and Application Share

In the 2010 Top-500 list, there were 274 supercomputing systems installed in the US, 41 systems in China, and 103 systems in Japan, France, UK, and Germany. The remaining countries have 82 systems. This country share roughly reflects the countries' economic growth over the years. Countries compete in the Top 500 race every six months. Major increases of supercomputer applications are in the areas of database, research, finance, and information services.

#### 2.1.4.6 Power versus Performance in the Top Five in 2010

In Figure 2.3, the top five supercomputers are ranked by their speed (Gflops) on the left side and by their power consumption (MW per system) on the right side. The Tianhe-1A scored the highest with a 2.57 Pflops speed and 4.01 MW power consumption. In second place, the Jaguar consumes the highest power of 6.9 MW. In fourth place, the TSUBAME system consumes the least power, 1.5 MW, and has a speed performance that almost matches that of the Nebulae system. One can define a performance/power ratio to see the trade-off between these two metrics. There is also a Top 500 Green contest that ranks the supercomputers by their power efficiency. This chart shows that all systems using the hybrid CPU/GPU architecture consume much less power.



**FIGURE 2.3** Power and performance of the top 5 supercomputers in November 2010. Courtesy of [www.top500.org](http://www.top500.org) [25] and B. Dally [10]

## 2.2 Computer Clusters and MPP Architectures

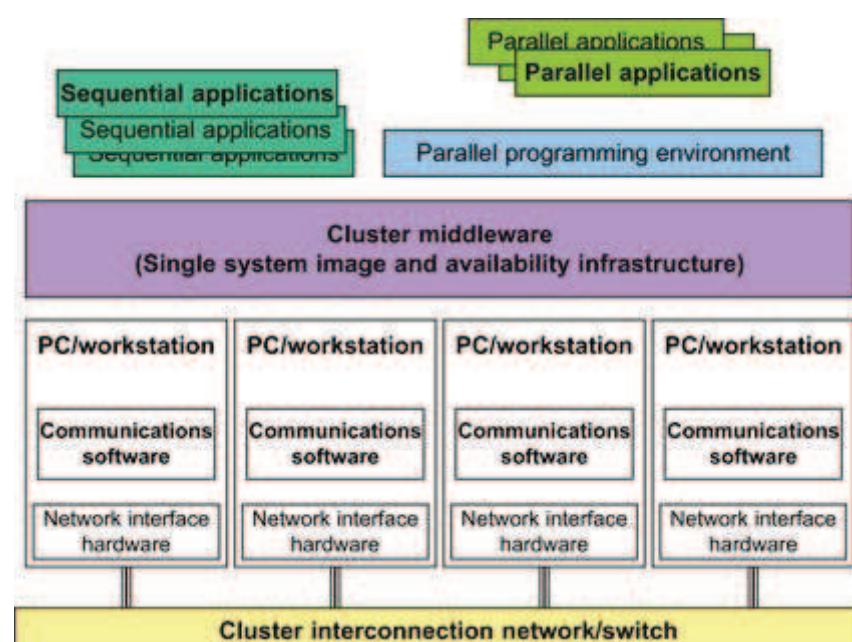
Most clusters emphasize higher availability and scalable performance. Clustered systems evolved from the Microsoft Wolfpack and Berkeley NOW to the SGI Altix Series, IBM SP Series, and IBM Roadrunner. NOW was a UC Berkeley research project designed to explore new mechanisms for clustering of UNIX workstations. Most clusters use commodity networks such as Gigabit Ethernet, Myrinet switches, or InfiniBand networks to interconnect the compute and storage nodes. The clustering trend moves from supporting large rack-size, high-end computer systems to high-volume, desktop or deskside computer systems, matching the downsizing trend in the computer industry.

## 2.2.1 Cluster Organization and Resource Sharing

In this section, we will start by discussing basic, small-scale PC or server clusters. We will discuss how to construct large-scale clusters and MPPs in subsequent sections.

### 2.2.1.1 A Basic Cluster Architecture

Figure 2.4 shows the basic architecture of a computer cluster over PCs or workstations. The figure shows a simple cluster of computers built with commodity components and fully supported with desired SSI features and HA capability. The processing nodes are commodity workstations, PCs, or servers. These commodity nodes are easy to replace or upgrade with new generations of hardware. The node operating systems should be designed for multiuser, multitasking, and multithreaded applications. The nodes are interconnected by one or more fast commodity networks. These networks use standard communication protocols and operate at a speed that should be two orders of magnitude faster than that of the current TCP/IP speed over Ethernet.



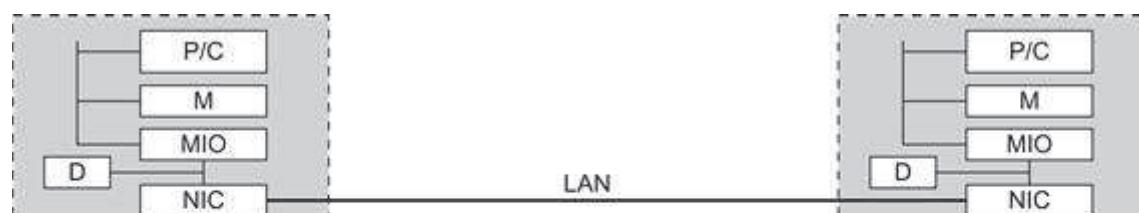
**FIGURE 2.4** The architecture of a computer cluster built with commodity hardware, software, middleware, and network components supporting HA and SSI. Courtesy of M. Baker and R. Buyya, reprinted with Permission [3]

The network interface card is connected to the node's standard I/O bus (e.g., PCI). When the processor or the operating system is changed, only the driver software needs to change. We desire to have a platform-independent *cluster operating system*, sitting on top of the node platforms. But such a cluster OS is not commercially available. Instead, we can deploy some cluster middleware to glue together all node platforms at the user space. An availability middleware offers HA services. An SSI layer provides a single entry point, a single file hierarchy, a single point of control, and a single job management system. Single memory may be realized with the help of the compiler or a runtime library. A single process space is not necessarily supported.

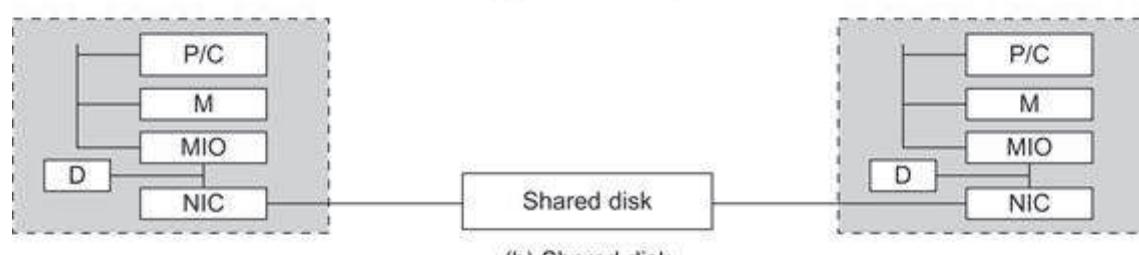
In general, an idealized cluster is supported by three subsystems. First, conventional databases and OLTP monitors offer users a desktop environment in which to use the cluster. In addition to running sequential user programs, the cluster supports parallel programming based on standard languages and communication libraries using PVM, MPI, or OpenMP. The programming environment also includes tools for debugging, profiling, monitoring, and so forth. A user interface subsystem is needed to combine the advantages of the web interface and the Windows GUI. It should also provide user-friendly links to various programming environments, job management tools, hypertext, and search support so that users can easily get help in programming the computer cluster.

### 2.2.1.2 Resource Sharing in Clusters

Supporting clusters of smaller nodes will increase computer sales. Clustering improves both availability and performance. These two clustering goals are not necessarily in conflict. Some HA clusters use hardware redundancy for scalable performance. The nodes of a cluster can be connected in one of three ways, as shown in Figure 2.5. The shared-nothing architecture is used in most clusters, where the nodes are connected through the I/O bus. The shared-disk architecture is in favor of small-scale *availability clusters* in business applications. When one node fails, the other node takes over.



(a) Shared nothing



(b) Shared disk





**FIGURE 2.5** Three ways to connect cluster nodes (P/C: Processor and Cache; M: Memory; D: Disk; NIC: Network Interface Circuitry; MIO: Memory-I/O Bridge.) Courtesy of Hwang and Xu [14]

The shared-nothing configuration in Part (a) simply connects two or more autonomous computers via a LAN such as Ethernet. A shared-disk cluster is shown in Part (b). This is what most business clusters desire so that they can enable recovery support in case of node failure. The shared disk can hold checkpoint files or critical system images to enhance cluster availability. Without shared disks, checkpointing, rollback recovery, failover, and fallback are not possible in a cluster. The shared-memory cluster in Part (c) is much more difficult to realize. The nodes could be connected by a *scalable coherence interface (SCI)* ring, which is connected to the memory bus of each node through an NIC module. In the other two architectures, the interconnect is attached to the I/O bus. The memory bus operates at a higher frequency than the I/O bus.

There is no widely accepted standard for the memory bus. But there are such standards for the I/O buses. One recent, popular standard is the PCI I/O bus standard. So, if you implement an NIC card to attach a faster Ethernet network to the PCI bus you can be assured that this card can be used in other systems that use PCI as the I/O bus. The I/O bus evolves at a much slower rate than the memory bus. Consider a cluster that uses connections through the PCI bus. When the processors are upgraded, the interconnect and the NIC do not have to change, as long as the new system still uses PCI. In a shared-memory cluster, changing the processor implies a redesign of the node board and the NIC card.

## 2.2.2 Node Architectures and MPP Packaging

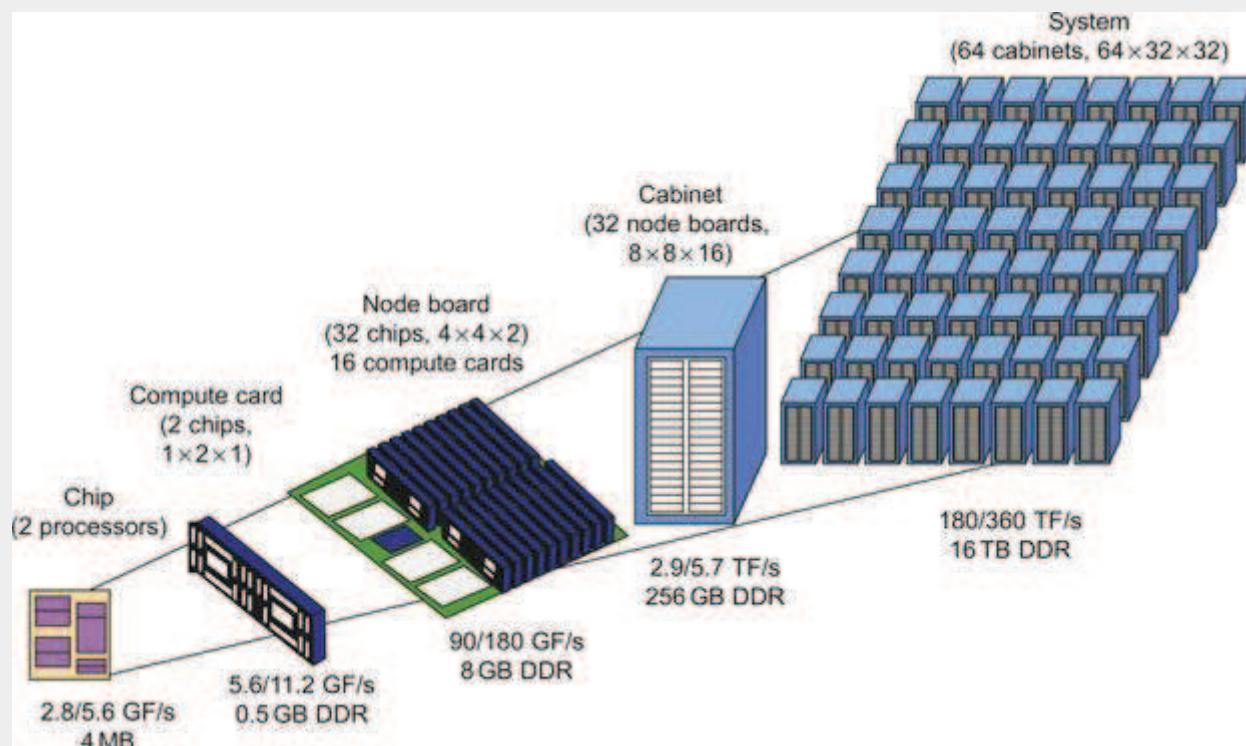
In building large-scale clusters or MPP systems, cluster nodes are classified into two categories: *compute nodes* and *service nodes*. Compute nodes appear in larger quantities mainly used for large-scale searching or parallel floating-point computations. Service nodes could be built with different processors mainly used to handle I/O, file access, and system monitoring. For MPP clusters, the compute nodes dominate in system cost, because we may have 1,000 times more compute nodes than service nodes in a single large clustered system. **Table 2.3** introduces two example compute node architectures: *homogeneous design* and *hybrid node design*.

In the past, most MPPs are built with a homogeneous architecture by interconnecting a large number of the same compute nodes. In 2010, the Cray XT5 Jaguar system was built with 224,162 AMD Opteron processors with six cores each. The Tiahe-1A adopted a hybrid node design using two Xeon CPUs plus two AMD GPUs per each compute node. The GPU could be replaced by special floating-point accelerators. A homogeneous node design makes it easier to program and maintain the system.

## Example 2.1

### Modular Packaging of the IBM Blue Gene/L System

The Blue Gene/L is a supercomputer jointly developed by IBM and Lawrence Livermore National Laboratory. The system became operational in 2005 with a 136 Tflops performance at the No. 1 position in the Top-500 list—toped the Japanese Earth Simulator. The system was upgraded to score a 478 Tflops speed in 2007. By examining the architecture of the Blue Gene series, we reveal the modular construction of a scalable MPP system as shown in [Figure 2.6](#). With modular packaging, the Blue Gene/L system is constructed hierarchically from processor chips to 64 physical racks. This system was built with a total of 65,536 nodes with two PowerPC 449 FP2 processors per node. The 64 racks are interconnected by a huge 3D  $64 \times 32 \times 32$  torus network.



**FIGURE 2.6** The IBM Blue Gene/L architecture built with modular components packaged hierarchically in five levels. Courtesy of N. Adiga, et al., IBM Corp., 2005 [1]

In the lower-left corner, we see a dual-processor chip. Two chips are mounted on a computer card. Sixteen computer cards (32 chips or 64 processors) are mounted on a node board. A cabinet houses 32 node boards with an  $8 \times 8 \times 16$  torus interconnect. Finally, 64 cabinets (racks) form the total system at the upper-right corner. This packaging diagram corresponds to the 2005 configuration. Customers can order any size

to meet their computational needs. The Blue Gene cluster was designed to achieve scalable performance, reliability through built-in testability, resilience by preserving locality of failures and checking mechanisms, and serviceability through partitioning and isolation of fault locations.

## 2.2.3 Cluster System Interconnects

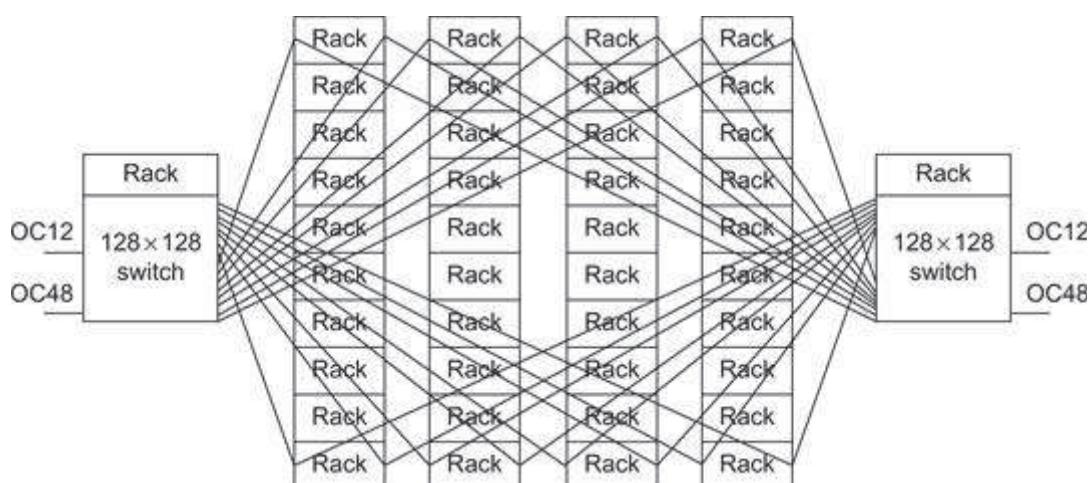
### 2.2.3.1 High-Bandwidth Interconnects

Table 2.4 compares four families of high-bandwidth system interconnects. In 2007, Ethernet used a 1 Gbps link, while the fastest InfiniBand links ran at 30 Gbps. The Myrinet and Quadrics perform in between. The MPI latency represents the state of the art in long-distance message passing. All four technologies can implement any network topology, including crossbar switches, fat trees, and torus networks. The InfiniBand is the most expensive choice with the fastest link speed. The Ethernet is still the most cost-effective choice. We consider two example cluster interconnects over 1,024 nodes in Figure 2.7 and Figure 2.9. The popularity of five cluster interconnects is compared in Figure 2.8.

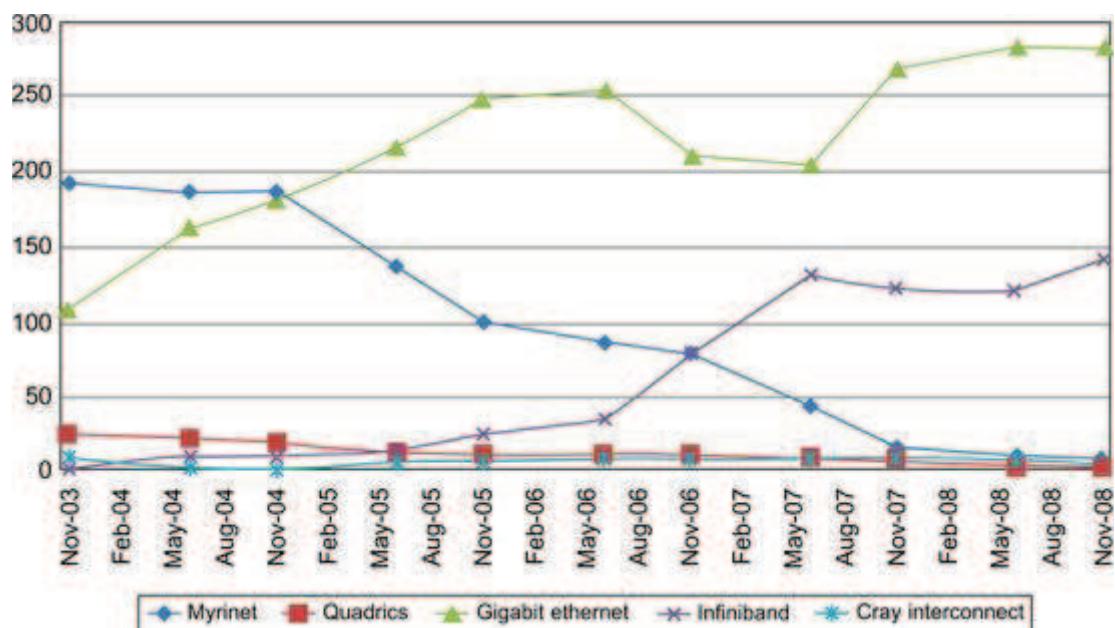
**Table 2.4**

**Comparison of Four Cluster Interconnect Technologies Reported in 2007**

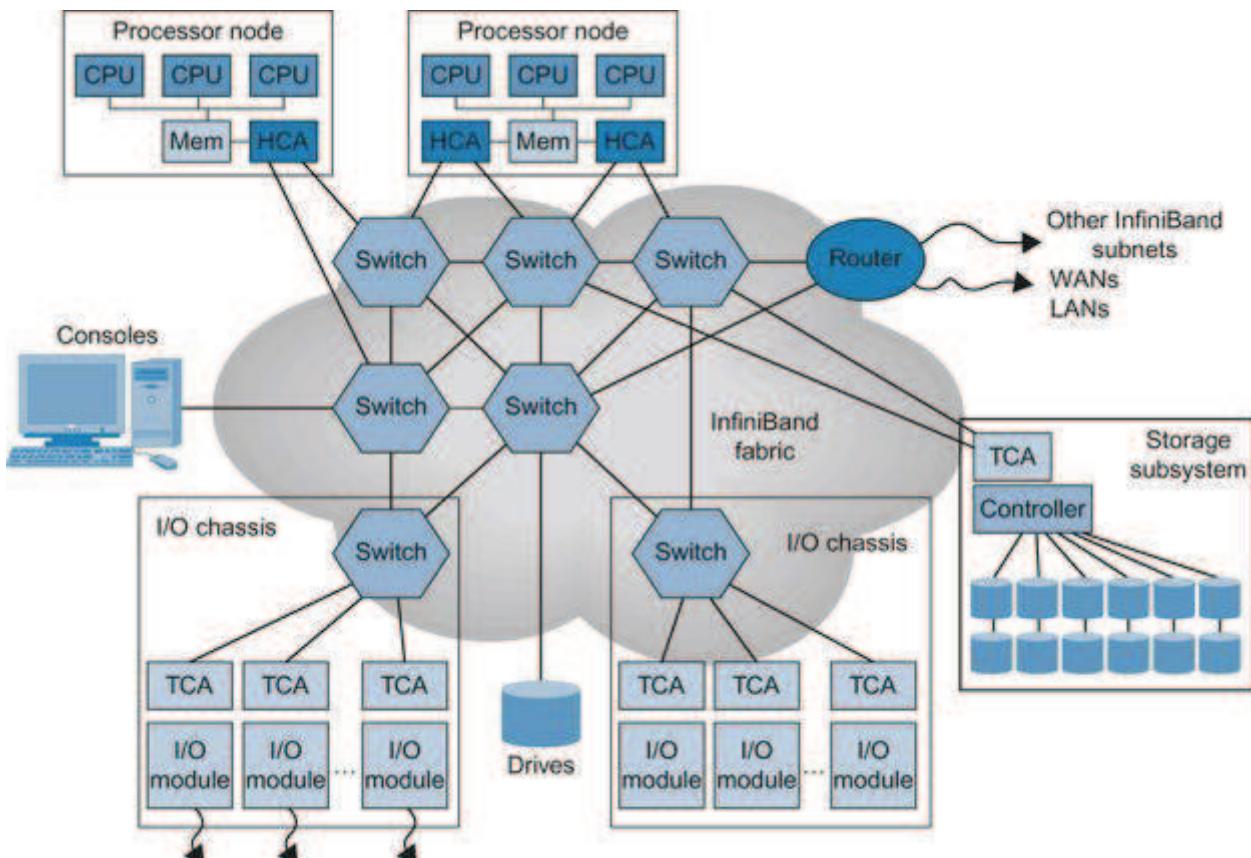
Feature	Myrinet	Quadrics	InfiniBand	Ethernet
Available link speeds	1.28 Gbps ( <i>M-XP</i> ) 10 Gbps ( <i>M-10G</i> )	2.8 Gbps ( <i>QsNet</i> ) 7.2 Gbps ( <i>QsNetII</i> )	2.5 Gbps (1X) 10 Gbps (4X) 30 Gbps (12X)	1 Gbps
MPI latency	~3 us	~3 us	~4.5 us	~40 us
Network processor	Yes	Yes	Yes	No
Topologies	Any	Any	Any	Any
Network topology	Clos	Fat tree	Fat tree	Any
Routing	Source-based, cut-through	Source-based, cut-through	Destination-based	Destination-based
Flow control	Stop and go	Worm-hole	Credit-based	802.3x



**FIGURE 2.7** Google search engine cluster architecture. Courtesy of Google, Inc. [6]



**FIGURE 2.8** Distribution of high-bandwidth interconnects in the Top 500 systems from 2003 to 2008. Courtesy of [www.top500.org](http://www.top500.org) [25]



**FIGURE 2.9** The InfiniBand system fabric built in a typical high-performance computer cluster. Source: O. Celebioglu, et al, "Exploring InfiniBand as an HPC Cluster Interconnect", Dell Power Solutions, Oct.2004 © 2011 Dell Inc. All Rights Reserved

## Example 2.2

### Crossbar Switch in Google Search Engine Cluster

Google has many data centers using clusters of low-cost PC engines. These clusters are mainly used to support Google's web search business. Figure 2.7 shows a Google cluster interconnect of 40 racks of PC engines via two racks of  $128 \times 128$  Ethernet switches. Each Ethernet switch can handle 128 one Gbps Ethernet links. A rack contains 80 PCs. This is an earlier cluster of 3,200 PCs. Google's search engine clusters are built with a lot more nodes. Today's server clusters from Google are installed in data centers with container trucks.

Two switches are used to enhance cluster availability. The cluster works fine even when one switch fails to provide the links among the PCs. The front ends of the switches are connected to the Internet via 2.4 Gbps OC 12 links. The 622 Mbps OC 12 links are connected to nearby data-center networks. In case of failure of the OC 48 links, the cluster is still connected to the outside world via the OC 12 links. Thus, the Google cluster eliminates all single points of failure.

## 2.2.3.2 Share of System Interconnects over Time

Figure 2.8 shows the distribution of large-scale system interconnects in the Top 500 systems from 2003 to 2008. Gigabit Ethernet is the most popular interconnect due to its low cost and market readiness. The InfiniBand network has been chosen in about 150 systems for its high-bandwidth performance. The Cray interconnect is designed for use in Cray systems only. The use of Myrinet and Quadrics networks had declined rapidly in the Top 500 list by 2008.

### Example 2.3

#### Understanding the InfiniBand Architecture[8]

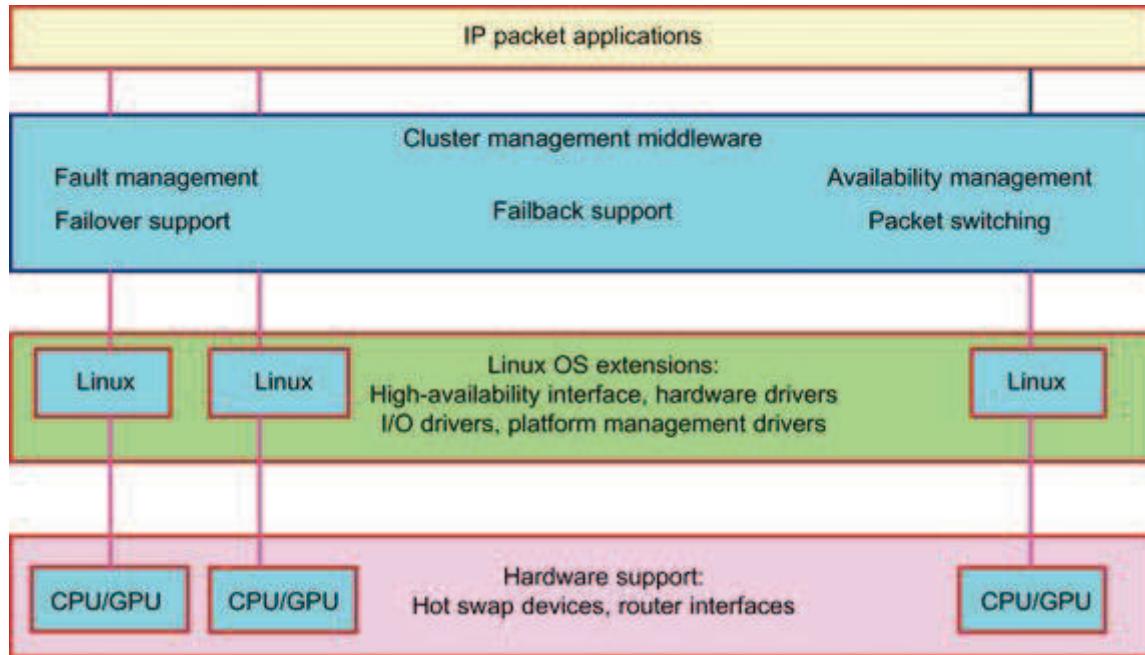
The InfiniBand has a switch-based point-to-point interconnect architecture. A large InfiniBand has a layered architecture. The interconnect supports the virtual interface architecture (VIA) for distributed messaging. The InfiniBand switches and links can make up any topology. Popular ones include crossbars, fat trees, and torus networks. Figure 2.9 shows the layered construction of an InfiniBand network. According to Table 2.5, the InfiniBand provides the highest speed links and the highest bandwidth in reported large-scale systems. However, InfiniBand networks cost the most among the four interconnect technologies.

Each end point can be a storage controller, a network interface card (NIC), or an interface to a host system. A host channel adapter (HCA) connected to the host processor through a standard peripheral component interconnect (PCI), PCI extended (PCI-X), or PCI express bus provides the host interface. Each HCA has more than one InfiniBand port. A target channel adapter (TCA) enables I/O devices to be loaded within the network. The TCA includes an I/O controller that is specific to its particular device's protocol such as SCSI, Fibre Channel, or Ethernet. This architecture can be easily implemented to build very large scale cluster interconnects that connect thousands or more hosts together. Supporting the InfiniBand in cluster applications can be found in [8].

## 2.2.4 Hardware, Software, and Middleware Support

Realistically, SSI and HA features in a cluster are not obtained free of charge. They must be supported by hardware, software, middleware, or OS extensions. Any change in hardware design and OS extensions must be done by the manufacturer. The hardware and OS support could be cost-prohibitive to ordinary users. However, programming level is a big burden to cluster users. Therefore, the middleware support at the application

level costs the least to implement. As an example, we show in [Figure 2.10](#) the middleware, OS extensions, and hardware support needed to achieve HA in a typical Linux cluster system.



**FIGURE 2.10** Middleware, Linux extensions, and hardware support for achieving massive parallelism and HA in a Linux cluster system built with CPUs and GPUs.

Close to the user application end, middleware packages are needed at the cluster management level: one for fault management to support *failover* and *fallback*, to be discussed in [Section 2.3.3](#). Another desired feature is to achieve HA using failure detection and recovery and packet switching. In the middle of [Figure 2.10](#), we need to modify the Linux OS to support HA, and we need special drivers to support HA, I/O, and hardware devices. Toward the bottom, we need special hardware to support hot-swapped devices and provide router interfaces. We will discuss various supporting mechanisms in subsequent sections.

## 2.2.5 GPU Clusters for Massive Parallelism

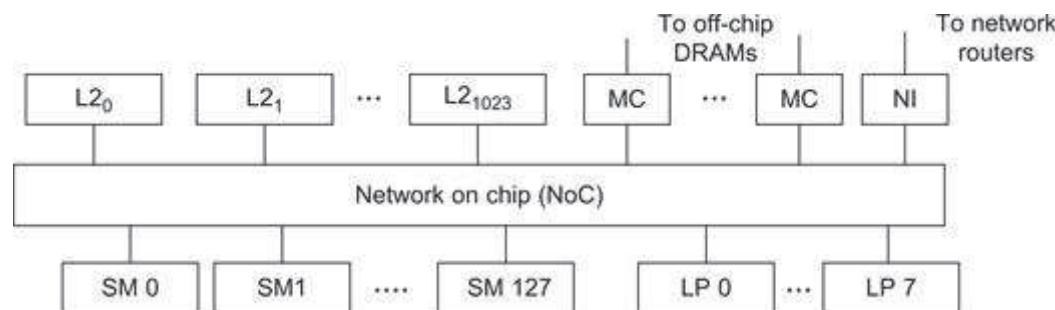
Commodity GPUs are becoming high-performance accelerators for data-parallel computing. Modern GPU chips contain hundreds of processor cores per chip. Based on a 2010 report [19], each GPU chip is capable of achieving up to 1 Tflops for single-precision (SP) arithmetic, and more than 80 Gflops for double-precision (DP) calculations. Recent HPC-optimized GPUs contain up to 4 GB of on-board memory, and are capable of sustaining memory bandwidths exceeding 100 GB/second. GPU clusters are built with a large number of GPU chips. GPU clusters have already demonstrated their capability to achieve Pflops performance in some of the Top 500 systems. Most GPU clusters are structured with homogeneous GPUs of the same hardware class, make, and model. The software used in a GPU cluster includes the OS, GPU drivers, and clustering API such as

an MPI.

The high performance of a GPU cluster is attributed mainly to its massively parallel multicore architecture, high throughput in multithreaded floating-point arithmetic, and significantly reduced time in massive data movement using large on-chip cache memory. In other words, GPU clusters already are more cost-effective than traditional CPU clusters. GPU clusters result in not only a quantum jump in speed performance, but also significantly reduced space, power, and cooling demands. A GPU cluster can operate with a reduced number of operating system images, compared with CPU-based clusters. These reductions in power, environment, and management complexity make GPU clusters very attractive for use in future HPC applications.

### 2.2.5.1 The Echelon GPU Chip Design

Figure 2.11 shows the architecture of a future GPU accelerator that was suggested for use in building a NVIDIA Echelon GPU cluster for Exascale computing. This Echelon project led by Bill Dally at NVIDIA is partially funded by DARPA under the Ubiquitous High-Performance Computing (UHPC) program. This GPU design incorporates 1024 stream cores and 8 latency-optimized CPU-like cores (called latency processor) on a single chip. Eight stream cores form a *stream multiprocessor* (SM) and there are 128 SMs in the Echelon GPU chip.



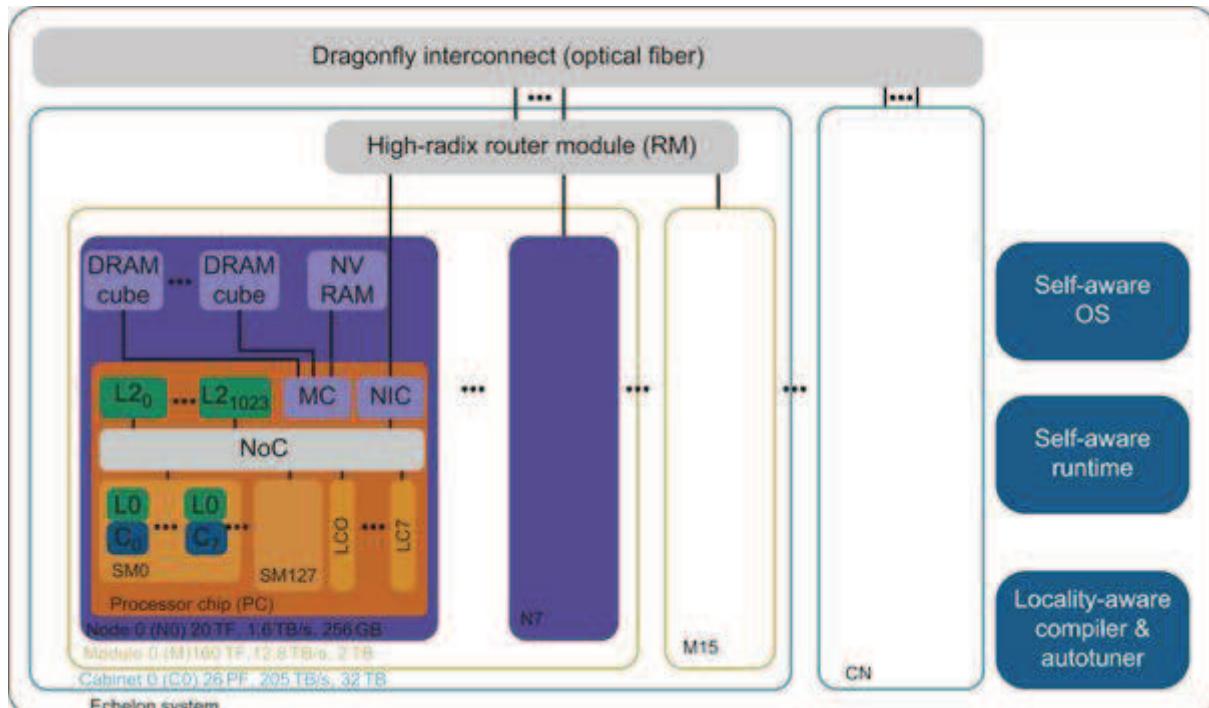
**FIGURE 2.11** The proposed GPU chip design for 20 Tflops performance and 1.6 TB/s memory bandwidth in the Echelon system. Courtesy of Bill Dally, Reprinted with Permission [10]

Each SM is designed with 8 processor cores to yield a 160 Gflops peak speed. With 128 SMs, the chip has a peak speed of 20.48 Tflops. These nodes are interconnected by a NoC (network on chip) to 1,024 SRAM banks (L2 caches). Each cache band has a 256 KB capacity. The MCs (*memory controllers*) are used to connect to off-chip DRAMs and the NI (*network interface*) is to scale the size of the GPU cluster hierarchically, as shown in Figure 2.14. At the time of this writing, the Echelon is only a research project. With permission from Bill Dally, we present the design for academic interest to illustrate how one can explore the many-core GPU technology to achieve Exascale computing in the future PU technology to achieve Exascale computing in the future.

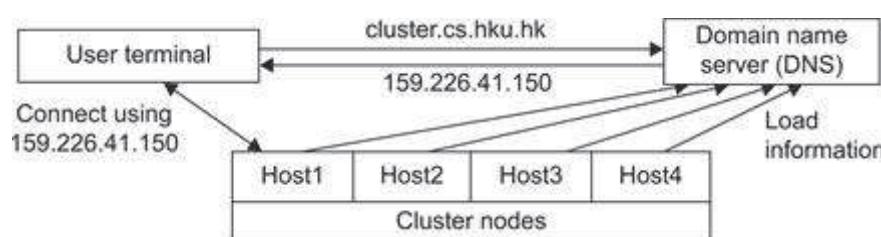
### 2.2.5.2 GPU Cluster Components

A GPU cluster is often built as a heterogeneous system consisting of three major

components: the CPU host nodes, the GPU nodes and the cluster interconnect between them. The GPU nodes are formed with general-purpose GPUs, known as GPGPUs, to carry out numerical calculations. The host node controls program execution. The cluster interconnect handles inter-node communications. To guarantee the performance, multiple GPUs must be fully supplied with data streams over high-bandwidth network and memory. Host memory should be optimized to match with the on-chip cache bandwidths on the GPUs. Figure 2.12 shows the proposed Echelon GPU clusters using the GPU chips shown in Figure 2.13 as building blocks interconnected by a hierarchically constructed network.



**FIGURE 2.12** The architecture of NVIDIA Echelon system built with a hierarchical network of GPUs that can deliver 2.6 Pflops per cabinet, and takes at least  $N = 400$  cabinets to achieve the desired Eflops performance. Courtesy of Bill Dally, Reprinted with Permission [10]



**FIGURE 2.13** Realizing a single entry point using a load-balancing domain name system (DNS). Courtesy of Hwang and Xu [14]

### 2.2.5.3 Echelon GPU Cluster Architecture

The Echelon system architecture is shown in Figure 2.11, hierarchically. The entire

Echelon system is built with  $N$  cabinets, labeled C0, C1, ..., CN. Each cabinet is built with 16 compute module labeled as M0, M1, ..., M15. Each compute module is built with 8 GPU nodes labeled as N0, N1, ..., N7. Each GPU node is the innermost block labeled as PC in [Figure 2.12](#) (also detailed in [Figure 2.11](#)). Each compute module features a performance of 160 Tflops and 12.8 TB/s over 2 TB of memory. Thus, a single cabinet can house 128 GPU nodes or 16,000 processor cores. Each cabinet has the potential to deliver 2.6 Pflops over 32 TB memory and 205 TB/s bandwidth. The  $N$  cabinets are interconnected by a Dragonfly network with optical fiber.

To achieve Eflops performance, we need to use at least  $N = 400$  cabinets. In total, an Exascale system needs 327,680 processor cores in 400 cabinets. The Echelon system is supported by a self-aware OS and runtime system. The Echelon system is also designed to preserve locality with the support of compiler and autotuner. At present, NVIDIA Fermi (GF110) chip has 512 stream processors. Thus the Echelon design is about 25 times faster. It is highly likely that the Echelon will employ post-Maxwell NVIDIA GPU planned to appear in 2013 ~ 2014 time frame.

#### 2.2.5.4 CUDA Parallel Programming

CUDA (*Compute Unified Device Architecture*) offers a parallel computing architecture developed by NVIDIA. CUDA is the computing engine in NVIDIA GPUs. This software is accessible to developers through standard programming languages. Programmers use C for CUDA C with NVIDIA extensions and certain restrictions. This CUDA C is compiled through a PathScale Open64 C compiler for parallel execution on a large number of GPU cores. [Example 2.4](#) shows the advantage of using CUDA C in parallel processing.

#### Example 2.4

#### Parallel SAXPY Execution Using CUDA C Code on GPUs

SAXPY is a kernel operation frequently performed in matrix multiplication. It essentially performs repeated *multiply and add* operations to generate the dot product of two long vectors. The following `saxpy_serial` routine is written in standard C code. This code is only suitable for sequential execution on a single processor core.

```
Void saxpy_serial (int n, float a, float*x, float *y
{ for (int i = 0; i < n; ++i), y[i] = a*x[i] + y[i] }
// Invoke the serial SAXPY kernel
```

```
saxpy_serial (n, 2.0, x, y);
```

The following **saxpy\_parallel** routine is written in CUDA C code for parallel execution by 256 threads/block on many processing cores on the GPU chip. Note that `n` blocks are handled by `n` processing cores, where `n` could be on the order of hundreds of blocks.

```
_global__void saxpy_parallel (int n, float a, float*x, float *y)  
{ Int i = blockIdx.x*blockDim.x + threadIdx.x; if (i < n) y [i] = a*x[i] + y  
// Invoke the parallel SAXPY kernel with 256 threads/block int nblocks = (n + 256)/256;  
saxpy_parallel <<< nblocks, 256 >>> (n, 2.0, x, y);
```

This is a good example of using CUDA C to exploit massive parallelism on a cluster of multicore and multithreaded processors using the CODA GPGPUs as building blocks.

### 2.2.5.5 CUDA Programming Interfaces

CUDA architecture shares a range of computational interfaces with two competitors: the *Khronos Group's Open Computing Language* and Microsoft's *DirectCompute*. Third-party wrappers are also available for using Python, Perl, FORTRAN, Java, Ruby, Lua, MATLAB, and IDL. CUDA has been used to accelerate nongraphical applications in computational biology, cryptography, and other fields by an order of magnitude or more. A good example is the BOINC distributed computing client. CUDA provides both a low-level API and a higher-level API. CUDA works with all NVIDIA GPUs from the G8X series onward, including the GeForce, Quadro, and Tesla lines. NVIDIA states that programs developed for the GeForce 8 series will also work without modification on all future NVIDIA video cards due to binary compatibility.

### 2.2.5.6 Trends in CUDA Usage

Tesla and Fermi are two generations of CUDA architecture released by NVIDIA in 2007 and 2010, respectively. The CUDA version 3.2 is used for using a single GPU module in 2010. A newer CUDA version 4.0 will allow multiple GPUs to address use an unified virtual address space of shared memory. The next NVIDIA GPUs will be Kepler-designed to support C++. The Fermi has eight times the peak double-precision floating-point performance of the Tesla GPU (5.8 Gflops/W versus 0.7 Gflops/W). Currently, the Fermi has up to 512 CUDA cores on 3 billion transistors.

Future applications of the CUDA GPUs and the Echelon system may include the following:

- The search for extraterrestrial intelligence (SETI@Home)
- Distributed calculations to predict the native conformation of proteins
- Medical analysis simulations based on CT and MRI scan images
- Physical simulations in fluid dynamics and environment statistics
- Accelerated 3D graphics, cryptography, compression, and interconversion of video file formats

- Building the *single-chip cloud computer* (SCC) through virtualization in many-core architecture.

## 2.3 Design Principles of Computer Clusters

Clusters should be designed for scalability and availability. In this section, we will cover the design principles of SSI, HA, fault tolerance, and rollback recovery in general-purpose computers and clusters of cooperative computers.

### 2.3.1 Single-System Image Features

SSI does not mean a single copy of an operating system image residing in memory, as in an SMP or a workstation. Rather, it means the *illusion* of a single system, single control, symmetry, and transparency as characterized in the following list:

- **Single system** The entire cluster is viewed by users as one system that has multiple processors. The user could say, “Execute my application using five processors.” This is different from a distributed system.
- **Single control** Logically, an end user or system user utilizes services from one place with a single interface. For instance, a user submits batch jobs to one set of queues; a system administrator configures all the hardware and software components of the cluster from one control point.
- **Symmetry** A user can use a cluster service from any node. In other words, all cluster services and functionalities are symmetric to all nodes and all users, except those protected by access rights.
- **Location-transparent** The user is not aware of the whereabouts of the physical device that eventually provides a service. For instance, the user can use a tape drive attached to any cluster node as though it were physically attached to the local node.

The main motivation to have SSI is that it allows a cluster to be used, controlled, and maintained as a familiar workstation is. The word “single” in “single-system image” is sometimes synonymous with “global” or “central.” For instance, a global file system means a single file hierarchy, which a user can access from any node. A single point of control allows an operator to monitor and configure the cluster system. Although there is an illusion of a single system, a cluster service or functionality is often realized in a distributed manner through the cooperation of multiple components. A main requirement (and advantage) of SSI techniques is that they provide both the performance benefits of distributed implementation and the usability benefits of a single image.

From the viewpoint of a process  $P$ , cluster nodes can be classified into three types. The *home node* of a process  $P$  is the node where  $P$  resided when it was created. The *local node* of a process  $P$  is the node where  $P$  currently resides. All other nodes are *remote nodes* to  $P$ . Cluster nodes can be configured to suit different needs. A *host node* serves user logins through Telnet, rlogin, or even FTP and HTTP. A *compute node* is one that performs computational jobs. An *I/O node* is one that serves file I/O requests. If a cluster has large shared disks and tape units, they are normally physically attached to I/O nodes.

There is one home node for each process, which is fixed throughout the life of the process. At any time, there is only one local node, which may or may not be the host node. The local node and remote nodes of a process may change when the process migrates. A node can be configured to provide multiple functionalities. For instance, a node can be designated as a host, an I/O node, and a compute node at the same time. The illusion of an SSI can be obtained at several layers, three of which are discussed in the following list. Note that these layers may overlap with one another.

- **Application software layer** Two examples are parallel web servers and various parallel databases. The user sees an SSI through the application and is not even aware that he is using a cluster. This approach demands the modification of workstation or SMP applications for clusters.
- **Hardware or kernel layer** Ideally, SSI should be provided by the operating system or by the hardware. Unfortunately, this is not a reality yet. Furthermore, it is extremely difficult to provide an SSI over heterogeneous clusters. With most hardware architectures and operating systems being proprietary, only the manufacturer can use this approach.
- **Middleware layer** The most viable approach is to construct an SSI layer just above the OS kernel. This approach is promising because it is platform-independent and does not require application modification. Many cluster job management systems have already adopted this approach.

Each computer in a cluster has its own operating system image. Thus, a cluster may display multiple system images due to the stand-alone operations of all participating node computers. Determining how to merge the multiple system images in a cluster is as difficult as regulating many individual personalities in a community to a single personality. With different degrees of resource sharing, multiple systems could be integrated to achieve SSI at various operational levels.

### **2.3.1.1 Single Entry Point**

Single-system image (SSI) is a very rich concept, consisting of single entry point, single file hierarchy, single I/O space, single networking scheme, single control point, single job management system, single memory space, and single process space. The single entry point enables users to log in (e.g., through Telnet, rlogin, or HTTP) to a cluster as one virtual host, although the cluster may have multiple physical host nodes to serve the login sessions. The system transparently distributes the user's login and connection requests to different physical hosts to balance the load. Clusters could substitute for mainframes and supercomputers. Also, in an Internet cluster server, thousands of HTTP or FTP requests may come simultaneously. Establishing a single entry point with multiple hosts is not a trivial matter. Many issues must be resolved. The following is just a partial list:

- **Home directory** Where do you put the user's home directory?
- **Authentication** How do you authenticate user logins?
- **Multiple connections** What if the same user opens several sessions to the same user account?

- **Host failure** How do you deal with the failure of one or more hosts?

### Example 2.5

## Realizing a Single Entry Point in a Cluster of Computers

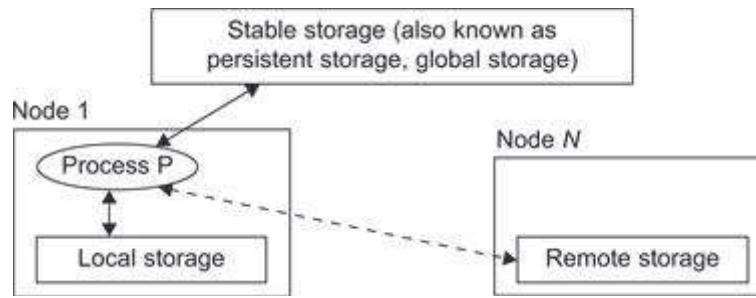
Figure 2.13 illustrates how to realize a single entry point. Four nodes of a cluster are used as host nodes to receive users' login requests. Although only one user is shown, thousands of users can connect to the cluster in the same fashion. When a user logs into the cluster, he issues a standard UNIX command such as *telnet cluster.cs.hku.hk*, using the symbolic name of the cluster system.

The DNS translates the symbolic name and returns the IP address 159.226.41.150 of the least-loaded node, which happens to be node Host1. The user then logs in using this IP address. The DNS periodically receives load information from the host nodes to make load-balancing translation decisions. In the ideal case, if 200 users simultaneously log in, the login sessions are evenly distributed among our hosts with 50 users each. This allows a single host to be four times more powerful.

### 2.3.1.2 Single File Hierarchy

We use the term "single file hierarchy" in this book to mean the illusion of a single, huge file system image that transparently integrates local and global disks and other file devices (e.g., tapes). In other words, all files a user needs are stored in some subdirectories of the root directory /, and they can be accessed through ordinary UNIX calls such as *open*, *read*, and so on. This should not be confused with the fact that multiple file systems can exist in a workstation as subdirectories of the root directory.

The functionalities of a single file hierarchy have already been partially provided by existing distributed file systems such as *Network File System (NFS)* and *Andrew File System (AFS)*. From the viewpoint of any process, files can reside on three types of locations in a cluster, as shown in Figure 2.14.



**FIGURE 2.14** Three types of storage in a single file hierarchy. Solid lines show what process P can access and the dashed line shows what P may be able to access. Courtesy of Hwang and Xu [14]

*Local storage* is the disk on the local node of a process. The disks on remote nodes are *remote storage*. A *stable storage* requires two aspects: It is *persistent*, which means data, once written to the stable storage, will stay there for a sufficiently long time (e.g., a week), even after the cluster shuts down; and it is fault-tolerant to some degree, by using redundancy and periodic backup to tapes.

Figure 2.14 uses stable storage. Files in stable storage are called *global files*, those in local storage *local files*, and those in remote storage *remote files*. Stable storage could be implemented as one centralized, large RAID disk. But it could also be distributed using local disks of cluster nodes. The first approach uses a large disk, which is a single point of failure and a potential performance bottleneck. The latter approach is more difficult to implement, but it is potentially more economical, more efficient, and more available. On many cluster systems, it is customary for the system to make visible to the user processes the following directories in a single file hierarchy: the usual *system directories* as in a traditional UNIX workstation, such as /usr and /usr/local; and the user's *home directory* ~/ that has a small disk quota (1–20 MB). The user stores his code files and other files here. But large data files must be stored elsewhere.

- A *global directory* is shared by all users and all processes. This directory has a large disk space of multiple gigabytes. Users can store their large data files here.
- On a cluster system, a process can access a special directory on the local disk. This directory has medium capacity and is faster to access than the global directory.

### 2.3.1.3 Visibility of Files

The term “visibility” here means a process can use traditional UNIX system or library calls such as *fopen*, *fread*, and *fwrite* to access files. Note that there are multiple local scratch directories in a cluster. The local scratch directories in remote nodes are not in the single file hierarchy, and are not directly visible to the process. A user process can still access them with commands such as *rcp* or some special library functions, by specifying both the node name and the filename.

The name “scratch” indicates that the storage is meant to act as a scratch pad for temporary information storage. Information in the local scratch space could be lost once the user logs out. Files in the global scratch space will normally persist even after the user logs out, but will be deleted by the system if not accessed in a predetermined time period. This is to free disk space for other users. The length of the period can be set by

the system administrator, and usually ranges from one day to several weeks. Some systems back up the global scratch space to tapes periodically or before deleting any files.

### 2.3.1.4 Support of Single-File Hierarchy

It is desired that a single file hierarchy have the SSI properties discussed, which are reiterated for file systems as follows:

- **Single system** There is just one file hierarchy from the user's viewpoint.
- **Symmetry** A user can access the global storage (e.g., `/scratch`) using a cluster service from any node. In other words, all file services and functionalities are symmetric to all nodes and all users, except those protected by access rights.
- **Location-transparent** The user is not aware of the whereabouts of the physical device that eventually provides a service. For instance, the user can use a RAID attached to any cluster node as though it were physically attached to the local node. There may be some performance differences, though.

A cluster file system should maintain *UNIX semantics*: Every file operation (`fopen`, `fread`, `fwrite`, `fclose`, etc.) is a transaction. When an `fread` accesses a file after an `fwrite` modifies the same file, the `fread` should get the updated value. However, existing distributed file systems do not completely follow UNIX semantics. Some of them update a file only at close or flush. A number of alternatives have been suggested to organize the global storage in a cluster. One extreme is to use a single file server that hosts a big RAID. This solution is simple and can be easily implemented with current software (e.g., NFS). But the file server becomes both a performance bottleneck and a single point of failure. Another extreme is to utilize the local disks in all nodes to form global storage. This could solve the performance and availability problems of a single file server.

### 2.3.1.5 Single I/O, Networking, and Memory Space

To achieve SSI, we desire a single control point, a single address space, a single job management system, a single user interface, and a single process control, as depicted in [Figure 2.17](#). In this example, each node has exactly one network connection. Two of the four nodes each have two I/O devices attached.

**Single Networking:** A properly designed cluster should behave as one system (the shaded area). In other words, it is like a big workstation with four network connections and four I/O devices attached. Any process on any node can use any network and I/O device as though it were attached to the local node. Single networking means any node can access any network connection.

**Single Point of Control:** The system administrator should be able to configure, monitor, test, and control the entire cluster and each individual node from a single point. Many clusters help with this through a system console that is connected to all nodes of the cluster. The system console is normally connected to an external LAN (not shown in [Figure 2.15](#)) so that the administrator can log in remotely to the system console from anywhere in the LAN to perform administration work.

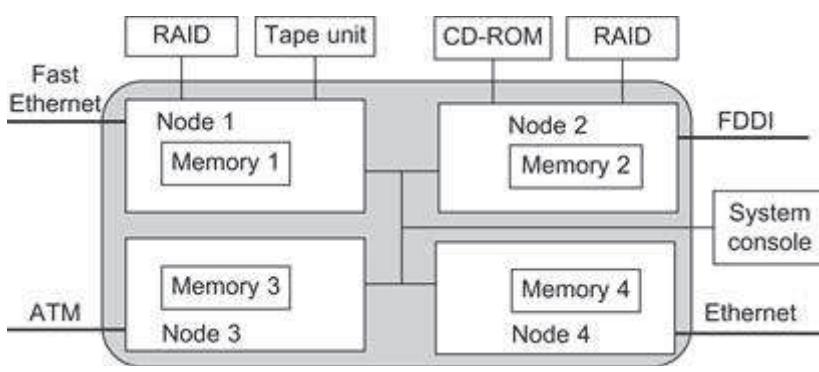
Note that single point of control does not mean all system administration work should be carried out solely by the system console. In reality, many administrative functions are distributed across the cluster. It means that controlling a cluster should be no more difficult than administering an SMP or a mainframe. It implies that administration-related system information (such as various configuration files) should be kept in one logical place. The administrator monitors the cluster with one graphics tool, which shows the entire picture of the cluster, and the administrator can zoom in and out at will.

Single point of control (or *single point of management*) is one of the most challenging issues in constructing a cluster system. Techniques from distributed and networked system management can be transferred to clusters. Several de facto standards have already been developed for network management. An example is *Simple Network Management Protocol (SNMP)*. It demands an efficient cluster management package that integrates with the availability support system, the file system, and the job management system.

**Single Memory Space:** *Single memory space* gives users the illusion of a big, centralized main memory, which in reality may be a set of distributed local memory spaces. PVPs, SMPs, and DSMs have an edge over MPPs and clusters in this respect, because they allow a program to utilize all global or local memory space. A good way to test if a cluster has a single memory space is to run a *sequential* program that needs a memory space larger than any single node can provide.

Suppose each node in [Figure 2.15](#) has 2 GB of memory available to users. An ideal single memory image would allow the cluster to execute a sequential program that needs 8 GB of memory. This would enable a cluster to operate like an SMP system. Several approaches have been attempted to achieve a single memory space on clusters. Another approach is to let the compiler distribute the data structures of an application across multiple nodes. It is still a challenging task to develop a single memory scheme that is efficient, platform-independent, and able to support sequential binary codes.

**Single I/O Address Space:** Assume the cluster is used as a web server. The web information database is distributed between the two RAIDs. An HTTP daemon is started on each node to handle web requests, which come from all four network connections. A single I/O space implies that any node can access the two RAIDs. Suppose most requests come from the ATM network. It would be beneficial if the functions of the HTTP on node 3 could be distributed to all four nodes. The following example shows a distributed RAID-x architecture for I/O-centric cluster computing [9].

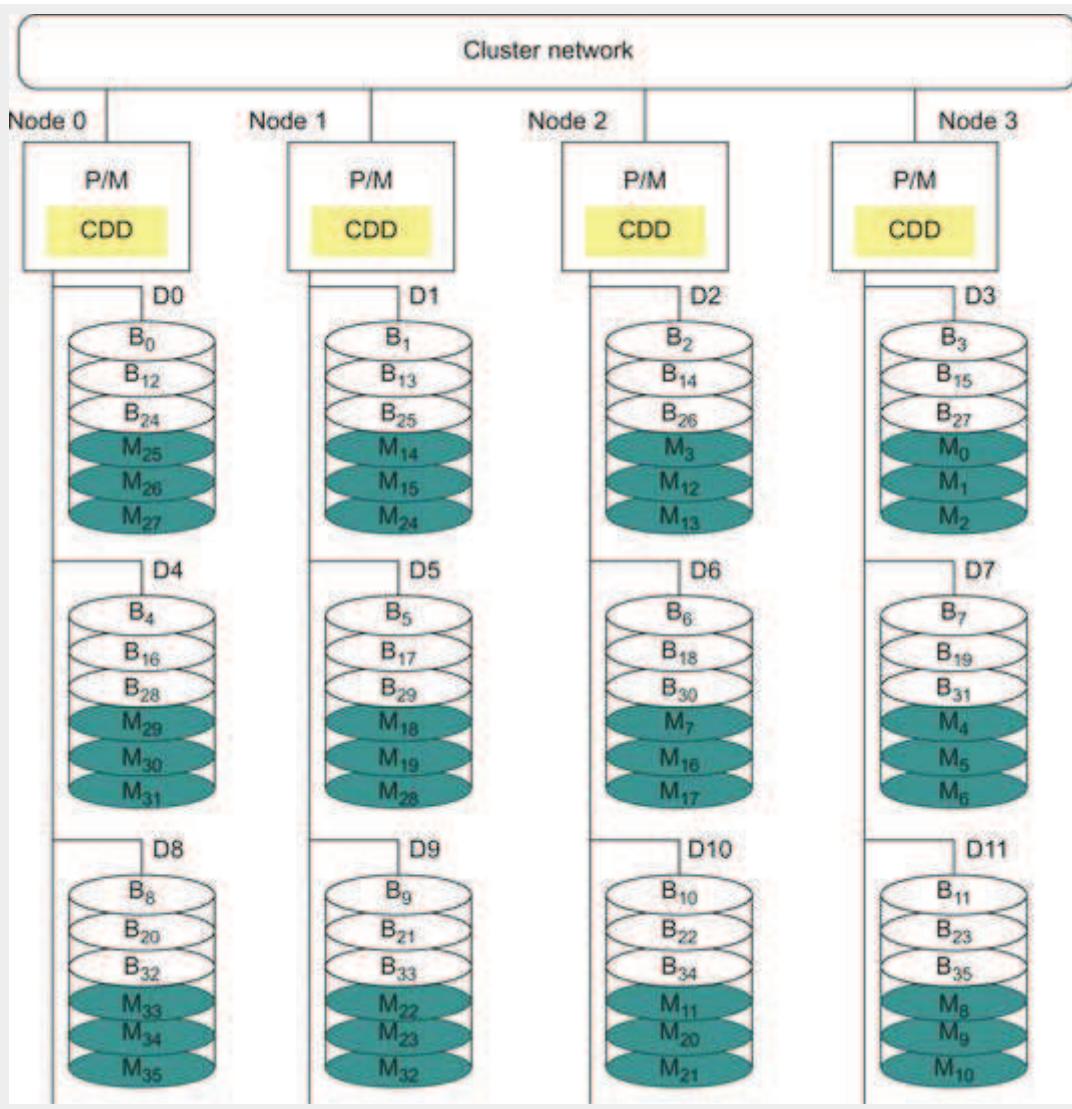


**FIGURE 2.15** A cluster with single networking, single I/O space, single memory, and single point of control.  
Courtesy of Hwang and Xu [14]

## Example 2.6

### Single I/O Space over Distributed RAID for I/O-Centric Clusters

A distributed disk array architecture was proposed by Hwang, et al. [9] for establishing a single I/O space in I/O-centric cluster applications. Figure 2.16 shows the architecture for a four-node Linux PC cluster, in which three disks are attached to the SCSI bus of each host node. All 12 disks form an integrated RAID-x with a single address space. In other words, all PCs can access both local and remote disks. The addressing scheme for all disk blocks is interleaved horizontally. Orthogonal stripping and mirroring make it possible to have a RAID-1 equivalent capability in the system.



**FIGURE 2.16** Distributed RAID architecture with a single I/O space over 12 distributed disks attached to 4 host computers in the cluster (Di stands for Disk i, Bj for disk block j, Mj an image for blocks Bj, P/M for processor/memory node, and CDD for cooperative disk driver.) Courtesy of Hwang, Jin, and Ho [13]

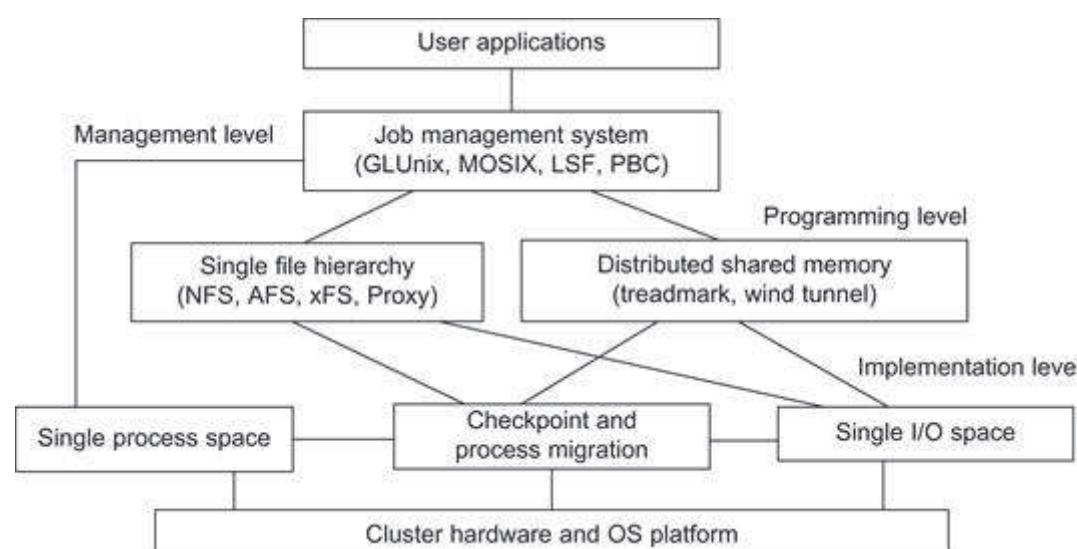
The shaded blocks are images of the blank blocks. A disk block and its image will be mapped on different physical disks in an orthogonal manner. For example, the block  $B_0$  is located on disk D0. The image block  $M_0$  of block  $B_0$  is located on disk D3. The four disks D0, D1, D2, and D3 are attached to four servers, and thus can be accessed in parallel. Any single disk failure will not lose the data block, because its image is available in recovery. All disk blocks are labeled to show image mapping. Benchmark experiments show that this RAID-x is scalable and can restore data after any single disk failure. The distributed RAID-x has improved aggregate I/O bandwidth in both parallel read and write operations over all physical disks in the cluster.

### 2.3.1.6 Other Desired SSI Features

The ultimate goal of SSI is for a cluster to be as easy to use as a desktop computer. Here

are additional types of SSI, which are present in SMP servers:

- **Single job management system** All cluster jobs can be submitted from any node to a single job management system.
- **Single user interface** The users use the cluster through a single graphical interface. Such an interface is available for workstations and PCs. A good direction to take in developing a cluster GUI is to utilize web technology.
- **Single process space** All user processes created on various nodes form a single process space and share a uniform process identification scheme. A process on any node can create (e.g., through a UNIX fork) or communicate with (e.g., through signals, pipes, etc.) processes on remote nodes.
- **Middleware support for SSI clustering** As shown in Figure 2.17, various SSI features are supported by middleware developed at three cluster application levels:
- **Management level** This level handles user applications and provides a job management system such as GLUnix, MOSIX, *Load Sharing Facility (LSF)*, or Codine.
- **Programming level** This level provides single file hierarchy (NFS, xFS, AFS, Proxy) and distributed shared memory (TreadMark, Wind Tunnel).
- **Implementation level** This level supports a single process space, checkpointing, process migration, and a single I/O space. These features must interface with the cluster hardware and OS platform. The distributed disk array, RAID-x, in Example 2.6 implements a single I/O space.



**FIGURE 2.17** Relationship among clustering middleware at the job management, programming, and implementation levels. Courtesy of K. Hwang, H. Jin, C.L. Wang and Z. Xu [16]

## 2.3.2 High Availability through Redundancy

When designing robust, highly available systems three terms are often used together: *reliability*, *availability*, and *serviceability* (RAS). Availability is the most interesting measure since it combines the concepts of reliability and serviceability as defined here:

- *Reliability* measures how long a system can operate without a breakdown.

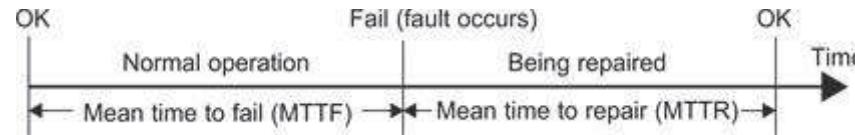
- *Availability* indicates the percentage of time that a system is available to the user, that is, the percentage of system uptime.
- *Serviceability* refers to how easy it is to service the system, including hardware and software maintenance, repair, upgrades, and so on.

The demand for RAS is driven by practical market needs. A recent Find/SVP survey found the following figures among Fortune 1000 companies: An average computer is down nine times per year with an average downtime of four hours. The average loss of revenue per hour of downtime is \$82,500. With such a hefty penalty for downtime, many companies are striving for systems that offer 24/365 *availability*, meaning the system is available 24 hours per day, 365 days per year.

### 2.3.2.1 Availability and Failure Rate

As Figure 2.18 shows, a computer system operates normally for a period of time before it fails. The failed system is then repaired, and the system returns to normal operation. This operate-repair cycle then repeats. A system's reliability is measured by the *mean time to failure* (MTTF), which is the average time of normal operation before the system (or a component of the system) fails. The metric for serviceability is the *mean time to repair* (MTTR), which is the average time it takes to repair the system and restore it to working condition after it fails. The *availability* of a system is defined by:

(2.1)



**FIGURE 2.18** The operate-repair cycle of a computer system.

### 2.3.2.2 Planned versus Unplanned Failure

When studying RAS, we call any event that prevents the system from normal operation a *failure*. This includes:

- **Unplanned failures** The system breaks, due to an operating system crash, a hardware failure, a network disconnection, human operation errors, a power outage, and so on. All these are simply called failures. The system must be repaired to correct the failure.
- **Planned shutdowns** The system is not broken, but is periodically taken off normal operation for upgrades, reconfiguration, and maintenance. A system may also be shut down for weekends or holidays. The MTTR in Figure 2.18 for this type of failure is the planned downtime.

Table 2.5 shows the availability values of several representative systems. For instance, a conventional workstation has an availability of 99 percent, meaning it is up and running

99 percent of the time or it has a downtime of 3.6 days per year. An optimistic definition of availability does not consider planned downtime, which may be significant. For instance, many supercomputer installations have a planned downtime of several hours per week, while a telephone system cannot tolerate a downtime of a few minutes per year.

**Table 2.5**  
**Availability of Computer System Types**

System Type	Availability (%)	Downtime in a Year
Conventional workstation	99	3.6 days
HA system	99.9	8.5 hours
Fault-resilient system	99.99	1 hour
Fault-tolerant system	99.999	5 minutes

### 2.3.2.3 Transient versus Permanent Failures

A lot of failures are *transient* in that they occur temporarily and then disappear. They can be dealt with without replacing any components. A standard approach is to roll back the system to a known state and start over. For instance, we all have rebooted our PC to take care of transient failures such as a frozen keyboard or window. *Permanent failures* cannot be corrected by rebooting. Some hardware or software component must be repaired or replaced. For instance, rebooting will not work if the system hard disk is broken.

### 2.3.2.4 Partial versus Total Failures

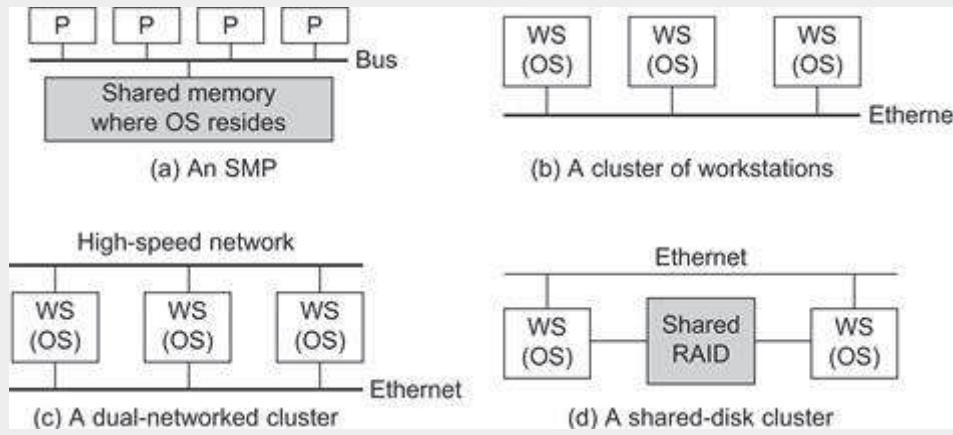
A failure that renders the entire system unusable is called a *total failure*. A failure that only affects part of the system is called a *partial failure* if the system is still usable, even at a reduced capacity. A key approach to enhancing availability is to make as many failures as possible partial failures, by systematically removing *single points of failure*, which are hardware or software components whose failure will bring down the entire system.

#### Example 2.7

#### Single Points of Failure in an SMP and in Clusters of Computers

In an SMP ([Figure 2.19\(a\)](#)), the shared memory, the OS image, and the memory bus are all single points of failure. On the other hand, the processors are not forming a single point of failure. In a cluster of workstations ([Figure 2.19\(b\)](#)), interconnected by Ethernet, there are multiple OS images, each residing in a workstation. This avoids the single point of failure caused by the

OS as in the SMP case. However, the Ethernet network now becomes a single point of failure, which is eliminated in [Figure 2.21\(c\)](#), where a high-speed network is added to provide two paths for communication.



**FIGURE 2.19** Single points of failure (SPF) in an SMP and in three clusters, where greater redundancy eliminates more SPFs in systems from (a) to (d). Courtesy of Hwang and Xu [14]

When a node fails in the clusters in [Figure 2.19\(b\)](#) and [Figure 2.19\(c\)](#), not only will the node applications all fail, but also all node data cannot be used until the node is repaired. The shared disk cluster in [Figure 2.19\(d\)](#) provides a remedy. The system stores persistent data on the shared disk, and periodically *checkpoints* to save intermediate results. When one WS node fails, the data will not be lost in this shared-disk cluster.

### 2.3.2.5 Redundancy Techniques

Consider the cluster in [Figure 2.19\(d\)](#). Assume only the nodes can fail. The rest of the system (e.g., interconnect and the shared RAID disk) is 100 percent available. Also assume that when a node fails, its workload is switched over to the other node in zero time. We ask, what is the availability of the cluster if planned downtime is ignored? What is the availability if the cluster needs one hour/week for maintenance? What is the availability if it is shut down one hour/week, one node at a time?

According to [Table 2.4](#), a workstation is available 99 percent of the time. The time both nodes are down is only 0.01 percent. Thus, the availability is 99.99 percent. It is now a fault-resilient system, with only one hour of downtime per year. The planned downtime is 52 hours per year, that is,  $52 / (365 \times 24) = 0.0059$ . The total downtime is now 0.59 percent + 0.01 percent = 0.6 percent. The availability of the cluster becomes 99.4 percent. Suppose we ignore the unlikely situation in which the other node fails while one node is maintained. Then the availability is 99.99 percent.

There are basically two ways to increase the availability of a system: increasing MTTF or reducing MTTR. Increasing MTTF amounts to increasing the reliability of the system.

The computer industry has strived to make reliable systems, and today's workstations have MTTFs in the range of hundreds to thousands of hours. However, to further improve MTTF is very difficult and costly. Clusters offer an HA solution based on reducing the MTTR of the system. A multinode cluster has a lower MTTF (thus lower reliability) than a workstation. However, the failures are taken care of quickly to deliver higher availability. We consider several redundancy techniques used in cluster design.

### **2.3.2.6 Isolated Redundancy**

A key technique to improve availability in any system is to use redundant components. When a component (the *primary* component) fails, the service it provided is taken over by another component (the *backup* component). Furthermore, the primary and the backup components should be *isolated* from each other, meaning they should not be subject to the same cause of failure. Clusters provide HA with redundancy in power supplies, fans, processors, memories, disks, I/O devices, networks, operating system images, and so on. In a carefully designed cluster, redundancy is also isolated. Isolated redundancy provides several benefits:

- First, a component designed with isolated redundancy is not a single point of failure, and the failure of that component will not cause a total system failure.
- Second, the failed component can be repaired while the rest of the system is still working.
- Third, the primary and the backup components can mutually test and debug each other.

The IBM SP2 communication subsystem is a good example of isolated-redundancy design. All nodes are connected by two networks: an Ethernet network and a high-performance switch. Each node uses two separate interface cards to connect to these networks. There are two communication protocols: a standard IP and a *user-space* (US) protocol; each can run on either network. If either network or protocol fails, the other network or protocol can take over.

### **2.3.2.7 N-Version Programming to Enhance Software Reliability**

A common isolated-redundancy approach to constructing a mission-critical software system is called *N-version programming*. The software is implemented by  $N$  isolated teams who may not even know the others exist. Different teams are asked to implement the software using different algorithms, programming languages, environment tools, and even platforms. In a fault-tolerant system, the  $N$  versions all run simultaneously and their results are constantly compared. If the results differ, the system is notified that a fault has occurred. But because of isolated redundancy, it is extremely unlikely that the fault will cause a majority of the  $N$  versions to fail at the same time. So the system continues working, with the correct result generated by majority voting. In a highly available but less mission-critical system, only one version needs to run at a time. Each version has a built-in self-test capability. When one version fails, another version can take over.

## 2.3.3 Fault-Tolerant Cluster Configurations

The cluster solution was targeted to provide availability support for two server nodes with three ascending levels of availability: *hot standby*, *active takeover*, and *fault-tolerant*. In this section, we will consider the *recovery time*, *failback feature*, and *node activeness*. The level of availability increases from standby to active and fault-tolerant cluster configurations. The shorter is the recovery time, the higher is the cluster availability. *Failback* refers to the ability of a recovered node returning to normal operation after repair or maintenance. *Activeness* refers to whether the node is used in active work during normal operation.

- **Hot standby server clusters** In a *hot standby* cluster, only the primary node is actively doing all the useful work normally. The standby node is powered on (hot) and running some monitoring programs to communicate heartbeat signals to check the status of the primary node, but is not actively running other useful workloads. The primary node must mirror any data to shared disk storage, which is accessible by the standby node. The standby node requires a second copy of data.
- **Active-takeover clusters** In this case, the architecture is symmetric among multiple server nodes. Both servers are primary, doing useful work normally. Both failover and failback are often supported on both server nodes. When a node fails, the user applications fail over to the available node in the cluster. Depending on the time required to implement the failover, users may experience some delays or may lose some data that was not saved in the last checkpoint.
- **Failover cluster** This is probably the most important feature demanded in current clusters for commercial applications. When a component fails, this technique allows the remaining system to take over the services originally provided by the failed component. A failover mechanism must provide several functions, such as *failure diagnosis*, *failure notification*, and *failure recovery*. Failure diagnosis refers to the detection of a failure and the location of the failed component that caused the failure. A commonly used technique is *heartbeat*, whereby the cluster nodes send out a stream of heartbeat messages to one another. If the system does not receive the stream of heartbeat messages from a node, it can conclude that either the node or the network connection has failed.

### Example 2.8

#### Failure Diagnosis and Recovery in a Dual-Network Cluster

A cluster uses two networks to connect its nodes. One node is designated as the *master node*. Each node has a *heartbeat daemon* that periodically (every 10 seconds) sends a heartbeat message to the master node through both networks. The master node will detect a failure if it does not receive messages

for a beat (10 seconds) from a node and will make the following diagnoses:

- A node's connection to one of the two networks failed if the master receives a heartbeat from the node through one network but not the other.
- The node failed if the master does not receive a heartbeat through either network. It is assumed that the chance of both networks failing at the same time is negligible.

The failure diagnosis in this example is simple, but it has several pitfalls. What if the master node fails? Is the 10-second heartbeat period too long or too short? What if the heartbeat messages are dropped by the network (e.g., due to network congestion)? Can this scheme accommodate hundreds of nodes? Practical HA systems must address these issues. A popular trick is to use the heartbeat messages to carry load information so that when the master receives the heartbeat from a node, it knows not only that the node is alive, but also the resource utilization status of the node. Such load information is useful for load balancing and job management.

Once a failure is diagnosed, the system notifies the components that need to know the failure event. Failure notification is needed because the master node is not the only one that needs to have this information. For instance, in case of the failure of a node, the DNS needs to be told so that it will not connect more users to that node. The resource manager needs to reassign the workload and to take over the remaining workload on that node. The system administrator needs to be alerted so that she can initiate proper actions to repair the node.

### 2.3.3.1 Recovery Schemes

Failure recovery refers to the actions needed to take over the workload of a failed component. There are two types of recovery techniques. In *backward recovery*, the processes running on a cluster periodically save a consistent state (called a *checkpoint*) to a stable storage. After a failure, the system is reconfigured to isolate the failed component, restores the previous checkpoint, and resumes normal operation. This is called *rollback*.

Backward recovery is relatively easy to implement in an application-independent, portable fashion, and has been widely used. However, rollback implies wasted execution. If execution time is crucial, such as in real-time systems where the rollback time cannot be tolerated, a *forward recovery* scheme should be used. With such a scheme, the system is not rolled back to the previous checkpoint upon a failure. Instead, the system utilizes the failure diagnosis information to reconstruct a valid system state and continues execution. Forward recovery is application-dependent and may need extra hardware.

## Example 2.9

### MTTF, MTTR, and Failure Cost Analysis

Consider a cluster that has little availability support. Upon a node failure, the following sequence of events takes place:

1. The entire system is shut down and powered off.
2. The faulty node is replaced if the failure is in hardware.
3. The system is powered on and rebooted.
4. The user application is reloaded and rerun from the start.

Assume one of the cluster nodes fails every 100 hours. Other parts of the cluster never fail. Steps 1 through 3 take two hours. On average, the mean time for step 4 is two hours. What is the availability of the cluster? What is the yearly failure cost if each one-hour downtime costs \$82,500?

**Solution:** The cluster's MTTF is 100 hours; the MTTR is  $2 + 2 = 4$  hours. According to Table 2.5, the availability is  $100/104 = 96.15$  percent. This corresponds to 337 hours of downtime in a year, and the failure cost is  $\$82500 \times 337$ , that is, more than \$27 million.

## Example 2.10

### Availability and Cost Analysis of a Cluster of Computers

Repeat Example 2.9, but assume that the cluster now has much increased availability support. Upon a node failure, its workload automatically fails over to other nodes. The failover time is only six minutes. Meanwhile, the cluster has *hot swap* capability: The faulty node is taken off the cluster, repaired, replugged, and rebooted, and it rejoins the cluster, all without impacting the rest of the cluster. What is the availability of this ideal cluster, and what is the yearly failure cost?

**Solution:** The cluster's MTTF is still 100 hours, but the MTTR is reduced to 0.1 hours, as the cluster is available while the failed node is being repaired. From to Table 2.5, the availability is  $100/100.5 = 99.9$  percent. This corresponds to 8.75 hours of downtime per year, and the failure cost is \$82,500, a  $27M/722K = 38$  times reduction in failure cost from the design in Example 3.8.

## 2.3.4 Checkpointing and Recovery Techniques

Checkpointing and recovery are two techniques that must be developed hand in hand to enhance the availability of a cluster system. We will start with the basic concept of checkpointing. This is the process of periodically saving the state of an executing program to stable storage, from which the system can recover after a failure. Each program state saved is called a *checkpoint*. The disk file that contains the saved state is called the *checkpoint file*. Although all current checkpointing software saves program states in a disk, research is underway to use node memories in place of stable storage in order to improve performance.

Checkpointing techniques are useful not only for availability, but also for program debugging, process migration, and load balancing. Many job management systems and some operating systems support checkpointing to a certain degree. The Web Resource contains pointers to numerous checkpoint-related web sites, including some public domain software such as Condor and Libckpt. Here we will present the important issues for the designer and the user of checkpoint software. We will first consider the issues that are common to both sequential and parallel programs, and then we will discuss the issues pertaining to parallel programs.

#### **2.3.4.1 Kernel, Library, and Application Levels**

Checkpointing can be realized by the operating system at the *kernel level*, where the OS transparently checkpoints and restarts processes. This is ideal for users. However, checkpointing is not supported in most operating systems, especially for parallel programs. A less transparent approach links the user code with a checkpointing library in the user space. Checkpointing and restarting are handled by this runtime support. This approach is used widely because it has the advantage that user applications do not have to be modified.

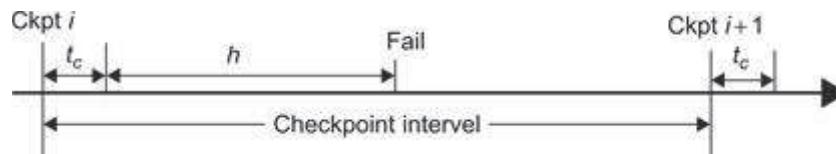
A main problem is that most current checkpointing libraries are static, meaning the application source code (or at least the object code) must be available. It does not work if the application is in the form of executable code. A third approach requires the user (or the compiler) to insert checkpointing functions in the application; thus, the application has to be modified, and the transparency is lost. However, it has the advantage that the user can specify where to checkpoint. This is helpful to reduce checkpointing overhead. Checkpointing incurs both time and storage overheads.

#### **2.3.4.2 Checkpoint Overheads**

During a program's execution, its states may be saved many times. This is denoted by the time consumed to save one checkpoint. The storage overhead is the extra memory and disk space required for checkpointing. Both time and storage overheads depend on the size of the checkpoint file. The overheads can be substantial, especially for applications that require a large memory space. A number of techniques have been suggested to reduce these overheads.

#### **2.3.4.3 Choosing an Optimal Checkpoint Interval**

The time period between two checkpoints is called the *checkpoint interval*. Making the interval larger can reduce checkpoint time overhead. However, this implies a longer computation time after a failure. Wong and Franklin [28] derived an expression for optimal checkpoint interval as illustrated in Figure 2.20.



**FIGURE 2.20** Time parameters between two checkpoints. Courtesy of Hwang and Xu [14]

(2.2)

Here, MTTF is the system's *mean time to failure*. This MTTF accounts the time consumed to save one checkpoint, and  $h$  is the average percentage of normal computation performed in a checkpoint interval before the system fails. The parameter  $h$  is always in the range. After a system is restored, it needs to spend  $h \times (\text{checkpoint interval})$  time to recompute.

#### 2.3.4.4 Incremental Checkpoint

Instead of saving the full state at each checkpoint, an *incremental checkpoint* scheme saves only the portion of the state that is changed from the previous checkpoint. However, care must be taken regarding *old* checkpoint files. In full-state checkpointing, only one checkpoint file needs to be kept on disk. Subsequent checkpoints simply overwrite this file. With incremental checkpointing, old files needed to be kept, because a state may span many files. Thus, the total storage requirement is larger.

#### 2.3.4.5 Forked Checkpointing

Most checkpoint schemes are blocking in that the normal computation is stopped while checkpointing is in progress. With enough memory, checkpoint overhead can be reduced by making a copy of the program state in memory and invoking another asynchronous thread to perform the checkpointing concurrently. A simple way to overlap checkpointing with computation is to use the UNIX *fork()* system call. The forked child process duplicates the parent process's address space and checkpoints it. Meanwhile, the parent process continues execution. Overlapping is achieved since checkpointing is disk-I/O intensive. A further optimization is to use the copy-on-write mechanism.

#### 2.3.4.6 User-Directed Checkpointing

The checkpoint overheads can sometimes be substantially reduced if the user inserts code (e.g., library or system calls) to tell the system when to save, what to save, and what

not to save. What should be the exact contents of a checkpoint? It should contain just enough information to allow a system to recover. The state of a process includes its data state and control state. For a UNIX process, these states are stored in its address space, including the text (code), the data, the stack segments, and the process descriptor. Saving and restoring the full state is expensive and sometimes impossible.

For instance, the process ID and the parent process ID are not restorable, nor do they need to be saved in many applications. Most checkpointing systems save a partial state. For instance, the code segment is usually not saved, as it does not change in most applications. What kinds of applications can be checkpointed? Current checkpoint schemes require programs to be *well behaved*, the exact meaning of which differs in different schemes. At a minimum, a well-behaved program should not need the exact contents of state information that is not restorable, such as the numeric value of a process ID.

### 2.3.4.7 Checkpointing Parallel Programs

We now turn to checkpointing parallel programs. The state of a parallel program is usually much larger than that of a sequential program, as it consists of the set of the states of individual processes, plus the state of the communication network. Parallelism also introduces various timing and consistency problems.

#### Example 2.11

#### Checkpointing a Parallel Program

Figure 2.21 illustrates checkpointing of a three-process parallel program. The arrows labeled  $x$ ,  $y$ , and  $z$  represent point-to-point communication among the processes. The three thick lines labeled  $a$ ,  $b$ , and  $c$  represent three *global snapshots* (or simply *snapshots*), where a global snapshot is a set of checkpoints (represented by dots), one from every process. In addition, some communication states may need to be saved. The intersection of a snapshot line with a process's time line indicates where the process should take a (local) checkpoint. Thus, the program's snapshot  $c$  consists of three local checkpoints:  $s$ ,  $t$ ,  $u$  for processes  $P$ ,  $Q$ , and  $R$ , respectively, plus saving the communication  $y$ .

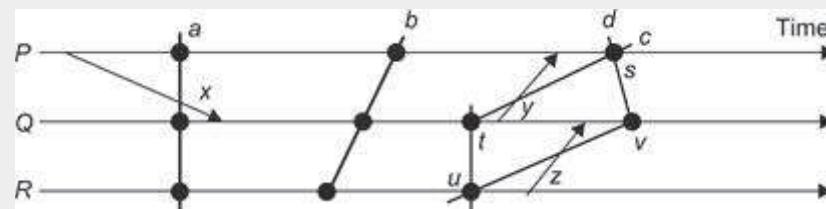


FIGURE 2.21 Consistent and inconsistent checkpoints in a parallel program. Courtesy of Hwang and Xu [14]

#### 2.3.4.8 Consistent Snapshot

A global snapshot is called *consistent* if there is no message that is received by the checkpoint of one process, but not yet sent by another process. Graphically, this corresponds to the case that no arrow crosses a snapshot line from right to left. Thus, snapshot *a* is consistent, because arrow *x* is from left to right. But snapshot *c* is inconsistent, as *y* goes from right to left. To be consistent, there should not be any *zigzag path* between two checkpoints [20]. For instance, checkpoints *u* and *s* cannot belong to a consistent global snapshot. A stronger consistency requires that no arrows cross the snapshot. Thus, only snapshot *b* is consistent in Figure 2.23.

#### 2.3.4.9 Coordinated versus Independent Checkpointing

Checkpointing schemes for parallel programs can be classified into two types. In *coordinated checkpointing* (also called *consistent checkpointing*), the parallel program is frozen, and all processes are checkpointed at the same time. In *independent checkpointing*, the processes are checkpointed independent of one another. These two types can be combined in various ways. Coordinated checkpointing is difficult to implement and tends to incur a large overhead. Independent checkpointing has a small overhead and can utilize existing checkpointing schemes for sequential programs.

### 2.4 Cluster Job and Resource Management

This section covers various scheduling methods for executing multiple jobs on a clustered system. The LSF is described as middleware for cluster computing. MOSIX is introduced as a distributed OS for managing resources in large-scale clusters or in clouds.

#### 2.4.1 Cluster Job Scheduling Methods

Cluster jobs may be scheduled to run at a specific time (*calendar scheduling*) or when a particular event happens (*event scheduling*). Table 2.6 summarizes various schemes to resolve job scheduling issues on a cluster. Jobs are scheduled according to priorities based on submission time, resource nodes, execution time, memory, disk, job type, and user identity. With *static priority*, jobs are assigned priorities according to a predetermined, fixed scheme. A simple scheme is to schedule jobs in a first-come, first-serve fashion. Another scheme is to assign different priorities to users. With *dynamic priority*, the priority of a job may change over time.

---

**Table 2.6**

# Job Scheduling Issues and Schemes for Cluster Nodes

Issue	Scheme	Key Problems
Job priority	Nonpreemptive	Delay of high-priority jobs
	Preemptive	Overhead, implementation
Resource required	Static	Load imbalance
	Dynamic	Overhead, implementation
Resource sharing	Dedicated	Poor utilization
	Space sharing	Tiling, large job
Scheduling	Time sharing	Process-based job control with context switch overhead
	Independent	Severe slowdown
	Gang scheduling	Implementation difficulty
Competing with foreign (local) jobs	Stay	Local job slowdown
	Migrate	Migration threshold, migration overhead

Three schemes are used to share cluster nodes. In the *dedicated mode*, only one job runs in the cluster at a time, and at most, one process of the job is assigned to a node at a time. The single job runs until completion before it releases the cluster to run other jobs. Note that even in the dedicated mode, some nodes may be reserved for system use and not be open to the user job. Other than that, all cluster resources are devoted to run a single job. This may lead to poor system utilization. The job resource requirement can be *static* or *dynamic*. Static scheme fixes the number of nodes for a single job for its entire period. Static scheme may underutilize the cluster resource. It cannot handle the situation when the needed nodes become unavailable, such as when the workstation owner shuts down the machine.

Dynamic resource allows a job to acquire or release nodes during execution. However, it is much more difficult to implement, requiring cooperation between a running job and the Java Message Service (JMS). The jobs make asynchronous requests to the JMS to add/delete resources. The JMS needs to notify the job when resources become available. The synchrony means that a job should not be delayed (blocked) by the request/notification. Cooperation between jobs and the JMS requires modification of the programming languages/libraries. A primitive mechanism for such cooperation exists in PVM and MPI.

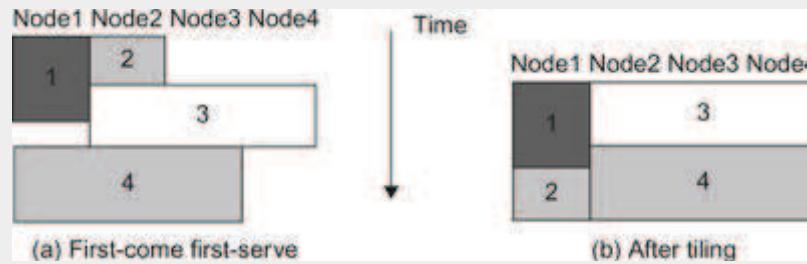
## 2.4.1.1 Space Sharing

A common scheme is to assign higher priorities to short, interactive jobs in daytime and during evening hours using *tiling*. In this *space-sharing mode*, multiple jobs can run on disjointed partitions (groups) of nodes simultaneously. At most, one process is assigned to a node at a time. Although a partition of nodes is dedicated to a job, the interconnect and the I/O subsystem may be shared by all jobs. Space sharing must solve the tiling problem and the large-job problem.

## Example 2.12

### Job Scheduling by Tiling over Cluster Nodes

Figure 2.22 illustrates the *tiling technique*. In Part (a), the JMS schedules four jobs in a first-come first-serve fashion on four nodes. Jobs 1 and 2 are small and thus assigned to nodes 1 and 2. Jobs 3 and 4 are parallel; each needs three nodes. When job 3 comes, it cannot run immediately. It must wait until job 2 finishes to free up the needed nodes. Tiling will increase the utilization of the nodes as shown in Figure 2.22(b). The overall execution time of the four jobs is reduced after repacking the jobs over the available nodes. This problem cannot be solved in dedicated or space-sharing modes. However, it can be alleviated by timesharing.



**FIGURE 2.22** The tiling technique for scheduling more jobs to cluster nodes to shorten the total makespan and thus increase the job throughput. Courtesy of Hwang and Xu [14]

#### 2.4.1.2 Time Sharing

In the dedicated or space-sharing model, only one user process is allocated to a node. However, the system processes or daemons are still running on the same node. In the *time-sharing* mode, multiple user processes are assigned to the same node. Time sharing introduces the following parallel scheduling policies:

1. **Independent scheduling** The most straightforward implementation of time sharing is to use the operating system of each cluster node to schedule different processes as in a traditional workstation. This is called *local scheduling* or *independent scheduling*. However, the performance of parallel jobs could be significantly degraded. Processes of a parallel job need to interact. For instance, when one process wants to barrier-synchronize with another, the latter may be scheduled out. So the first process has to wait. As the second process is rescheduled, the first process may be swapped out.
2. **Gang scheduling** The *gang scheduling* scheme schedules all processes of a parallel job together. When one process is active, all processes are active. The cluster nodes are not perfectly clock-synchronized. In fact, most clusters are asynchronous systems, and are not driven by the same clock. Although we say, “All processes are scheduled to run at the same time,” they do not start exactly at the same time. *Gang-scheduling skew* is the

maximum difference between the time the first process starts and the time the last process starts. The execution time of a parallel job increases as the gang-scheduling skew becomes larger, leading to longer execution time. We should use a homogeneous cluster, where gang scheduling is more effective. However, gang scheduling is not yet realized in most clusters, because of implementation difficulties.

3. **Competition with foreign (local) jobs** Scheduling becomes more complicated when both cluster jobs and local jobs are running. Local jobs should have priority over cluster jobs. With one keystroke, the owner wants command of all workstation resources. There are basically two ways to deal with this situation: The cluster job can either stay in the workstation node or migrate to another idle node. A *stay scheme* has the advantage of avoiding migration cost. The cluster process can be run at the lowest priority. The workstation's cycles can be divided into three portions, for kernel processes, local processes, and cluster processes. However, to stay slows down both the local and the cluster jobs, especially when the cluster job is a load-balanced parallel job that needs frequent synchronization and communication. This leads to the migration approach to flow the jobs around available nodes, mainly for balancing the workload.

## 2.4.2 Cluster Job Management Systems

Job management is also known as *workload management*, *load sharing*, or *load management*. We will first discuss the basic issues facing a job management system and summarize the available software packages. A *Job Management System (JMS)* should have three parts:

- A *user server* lets the user submit jobs to one or more queues, specify resource requirements for each job, delete a job from a queue, and inquire about the status of a job or a queue.
- A *job scheduler* performs job scheduling and queuing according to job types, resource requirements, resource availability, and scheduling policies.
- A *resource manager* allocates and monitors resources, enforces scheduling policies, and collects accounting information.

### 2.4.2.1 JMS Administration

The functionality of a JMS is often distributed. For instance, a user server may reside in each host node, and the resource manager may span all cluster nodes. However, the administration of a JMS should be *centralized*. All configuration and log files should be maintained in one location. There should be a single user interface to use the JMS. It is undesirable to force the user to run PVM jobs through one software package, MPI jobs through another, and HPF jobs through yet another.

The JMS should be able to dynamically reconfigure the cluster with minimal impact on the running jobs. The administrator's prologue and epilogue scripts should be able to run before and after each job for security checking, accounting, and cleanup. Users should be able to cleanly kill their own jobs. The administrator or the JMS should be able to cleanly suspend or kill any job. *Clean* means that when a job is suspended or killed, all

its processes must be included. Otherwise, some “orphan” processes are left in the system, which wastes cluster resources and may eventually render the system unusable.

### 2.4.2.2 Cluster Job Types

Several types of jobs execute on a cluster. *Serial jobs* run on a single node. *Parallel jobs* use multiple nodes. *Interactive jobs* are those that require fast turnaround time, and their input/output is directed to a terminal. These jobs do not need large resources, and users expect them to execute immediately, not to wait in a queue. *Batch jobs* normally need more resources, such as large memory space and long CPU time. But they do not need immediate responses. They are submitted to a job queue to be scheduled to run when the resource becomes available (e.g., during off hours).

While both interactive and batch jobs are managed by the JMS, *foreign jobs* are created outside the JMS. For instance, when a network of workstations is used as a cluster, users can submit interactive or batch jobs to the JMS. Meanwhile, the owner of a workstation can start a foreign job at any time, which is not submitted through the JMS. Such a job is also called a *local job*, as opposed to *cluster jobs* (interactive or batch, parallel or serial) that are submitted through the JMS of the cluster. The characteristic of a local job is fast response time. The owner wants all resources to execute his job, as though the cluster jobs did not exist.

### 2.4.2.3 Characteristics of a Cluster Workload

To realistically address job management issues, we must understand the workload behavior of clusters. It may seem ideal to characterize workload based on long-time operation data on real clusters. The parallel workload traces include both development and production jobs. These traces are then fed to a simulator to generate various statistical and performance results, based on different sequential and parallel workload combinations, resource allocations, and scheduling policies. The following workload characteristics are based on a NAS benchmark experiment. Of course, different workloads may have variable statistics.

- Roughly half of parallel jobs are submitted during regular working hours. Almost 80 percent of parallel jobs run for three minutes or less. Parallel jobs running longer than 90 minutes account for 50 percent of the total time.
- The sequential workload shows that 60 percent to 70 percent of workstations are available to execute parallel jobs at any time, even during peak daytime hours.
- On a workstation, 53 percent of all idle periods are three minutes or less, but 95 percent of idle time is spent in periods of time that are 10 minutes or longer.
- A 2:1 rule applies, which says that a network of 64 workstations, with proper JMS software, can sustain a 32-node parallel workload in addition to the original sequential workload. In other words, clustering gives a supercomputer half of the cluster size for free!

### 2.4.2.4 Migration Schemes

A *migration scheme* must consider the following three issues:

- **Node availability** This refers to node availability for job migration. The Berkeley NOW project has reported such opportunity does exist in university campus environment. Even during peak hours, 60 percent of workstations in a cluster are available at Berkeley campus.
- **Migration overhead** What is the effect of the migration overhead? The migration time can significantly slow down a parallel job. It is important to reduce the migration overhead (e.g., by improving the communication subsystem) or to migrate only rarely. The slowdown is significantly reduced if a parallel job is run on a cluster of twice the size. For instance, for a 32-node parallel job run on a 60-node cluster, the slowdown caused by migration is no more than 20 percent, even when the migration time is as long as three minutes. This is because more nodes are available, and thus the migration demand is less frequent.
- **Recruitment threshold** What should be the recruitment threshold? In the worst scenario, right after a process migrates to a node, the node is immediately claimed by its owner. Thus, the process has to migrate again, and the cycle continues. The recruitment threshold is the amount of time a workstation stays unused before the cluster considers it an idle node.

#### 2.4.2.5 Desired Features in A JMS

Here are some features that have been built in some commercial JMSes in cluster computing applications:

- Most support heterogeneous Linux clusters. All support parallel and batch jobs. However, Connect:Queue does not support interactive jobs.
- Enterprise cluster jobs are managed by the JMS. They will impact the owner of a workstation in running local jobs. However, NQE and PBS allow the impact to be adjusted. In DQS, the impact can be configured to be minimal.
- All packages offer some kind of load-balancing mechanism to efficiently utilize cluster resources. Some packages support checkpointing.
- Most packages cannot support dynamic process migration. They support static migration: A process can be dispatched to execute on a remote node when the process is first created. However, once it starts execution, it stays in that node. A package that does support dynamic process migration is Condor.
- All packages allow dynamic suspension and resumption of a user job by the user or by the administrator. All packages allow resources (e.g., nodes) to be dynamically added to or deleted.
- Most packages provide both a command-line interface and a graphical user interface. Besides UNIX security mechanisms, most packages use the Kerberos authentication system.

#### 2.4.3 Load Sharing Facility (LSF) for Cluster Computing

LSF is a commercial workload management system from Platform Computing [29]. LSF

emphasizes job management and load sharing on both parallel and sequential jobs. In addition, it supports checkpointing, availability, load migration, and SSI. LSF is highly scalable and can support a cluster of thousands of nodes. LSF has been implemented for various UNIX and Windows/NT platforms. Currently, LSF is being used not only in clusters but also in grids and clouds.

### 2.4.3.1 LSF Architecture

LSF supports most UNIX platforms and uses the standard IP for JMS communication. Because of this, it can convert a heterogeneous network of UNIX computers into a cluster. There is no need to change the underlying OS kernel. The end user utilizes the LSF functionalities through a set of utility commands. PVM and MPI are supported. Both a command-line interface and a GUI are provided. LSF also offers skilled users an API that is a runtime library called *LSLIB* (*load sharing library*). Using LSLIB explicitly requires the user to modify the application code, whereas using the utility commands does not. Two LSF daemons are used on each server in the cluster. The *load information managers* (*LIMs*) periodically exchange load information. The *remote execution server* (*RES*) executes remote tasks.

### 2.4.3.2 LSF Utility Commands

A cluster node may be a single-processor host or an SMP node with multiple processors, but always runs with only a single copy of the operating system on the node. Here are interesting features built into the LSF facilities:

- LSF supports all four combinations of interactive, batch, sequential, and parallel jobs. A job that is not executed through LSF is called a *foreign job*. A *server node* is one which can execute LSF jobs. A *client node* is one that can initiate or submit LSF jobs but cannot execute them. Only the resources on the server nodes can be shared. Server nodes can also initiate or submit LSF jobs.
- LSF offers a set of tools (*lstoools*) to get information from LSF and to run jobs remotely. For instance, *lshosts* lists the static resources (discussed shortly) of every server node in the cluster. The command *lsrun* executes a program on a remote node.
- When a user types the command line `%lsrun-R 'swp>100' myjob` at a client node, the application *myjob* will be automatically executed on the most lightly loaded server node that has an available swap space greater than 100 MB.
- The *lsbatch* utility allows users to submit, monitor, and execute batch jobs through LSF. This utility is a load-sharing version of the popular UNIX command interpreter *tcsh*. Once a user enters the *lstcsh* shell, every command issued will be automatically executed on a suitable node. This is done transparently: The user sees a shell exactly like a *tcsh* running on the local node.
- The *lsmake* utility is a parallel version of the UNIX make utility, allowing a makefile to be processed in multiple nodes simultaneously.

## Example 2.13

### Application of the LSF on a Cluster of Computers

Suppose a cluster consists of eight expensive server nodes and 100 inexpensive client nodes (workstations or PCs). The server nodes are expensive due to better hardware and software, including application software. A license is available to install a FORTRAN compiler and a CAD simulation package, both valid for up to four users. Using a JMS such as LSF, all the hardware and software resources of the server nodes are made available to the clients transparently.

A user sitting in front of a client's terminal feels as though the client node has all the software and speed of the servers locally. By typing *lsmake my.makefile*, the user can compile his source code on up to four servers. LSF selects the nodes with the least amount of load. Using LSF also benefits resource utilization. For instance, a user wanting to run a CAD simulation can submit a batch job. LSF will schedule the job as soon as the software becomes available.

## 2.4.4 MOSIX: An OS for Linux Clusters and Clouds

MOSIX is a distributed operating system that was developed at Hebrew University in 1977. Initially, the system extended BSD/OS system calls for resource sharing in Pentium clusters. In 1999, the system was redesigned to run on Linux clusters built with x86 platforms. The MOSIX project is still active as of 2011, with 10 versions released over the years. The latest version, MOSIX2, is compatible with Linux 2.6.

### 2.4.4.1 MOXIS2 for Linux Clusters

MOSIX2 runs as a virtualization layer in the Linux environment. This layer provides SSI to users and applications along with runtime Linux support. The system runs applications in remote nodes as though they were run locally. It supports both sequential and parallel applications, and can discover resources and migrate software processes transparently and automatically among Linux nodes. MOSIX2 can also manage a Linux cluster or a grid of multiple clusters.

Flexible management of a grid allows owners of clusters to share their computational resources among multiple cluster owners. Each cluster can still preserve its autonomy over its own clusters and its ability to disconnect its nodes from the grid at any time. This can be done without disrupting the running programs. A MOSIX-enabled grid can extend indefinitely as long as trust exists among the cluster owners. The condition is to guarantee that guest applications cannot be modified while running in remote clusters. Hostile computers are not allowed to connect to the local network.

## 2.4.4.2 SSI Features in MOSIX2

The system can run in *native mode* or as a VM. In native mode, the performance is better, but it requires that you modify the base Linux kernel, whereas a VM can run on top of any unmodified OS that supports virtualization, including Microsoft Windows, Linux, and Mac OS X. The system is most suitable for running compute-intensive applications with low to moderate amounts of I/O. Tests of MOSIX2 show that the performance of several such applications over a 1 GB/second campus grid is nearly identical to that of a single cluster. Here are some interesting features of MOSIX2:

- Users can log in on any node and do not need to know where their programs run.
- There is no need to modify or link applications with special libraries.
- There is no need to copy files to remote nodes, thanks to automatic resource discovery and workload distribution by process migration.
- Users can load-balance and migrate processes from slower to faster nodes and from nodes that run out of free memory.
- Sockets are migratable for direct communication among migrated processes.
- The system features a secure runtime environment (sandbox) for guest processes.
- The system can run batch jobs with checkpoint recovery along with tools for automatic installation and configuration scripts.

## 2.4.4.3 Applications of MOSIX for HPC

The MOSIX is a research OS for HPC cluster, grid, and cloud computing. The system's designers claim that MOSIX offers efficient utilization of wide-area grid resources through automatic resource discovery and load balancing. The system can run applications with unpredictable resource requirements or runtimes by running long processes, which are automatically sent to grid nodes. The system can also combine nodes of different capacities by migrating processes among nodes based on their load index and available memory.

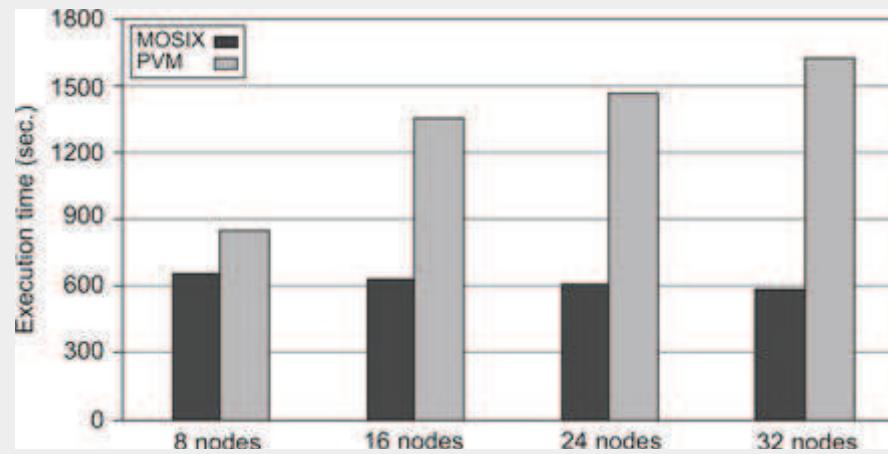
MOSIX became proprietary software in 2001. Application examples include scientific computations for genomic sequence analysis, molecular dynamics, quantum dynamics, nanotechnology and other parallel HPC applications; engineering applications including CFD, weather forecasting, crash simulations, oil industry simulations, ASIC design, and pharmaceutical design; and cloud applications such as for financial modeling, rendering farms, and compilation farms.

### Example 2.14

#### Memory-Ushering Algorithm Using MOSIX versus PVM

Memory ushering is practiced to borrow the main memory of a remote cluster node, when the main memory on a local node is exhausted. The remote memory access is done by process migration instead of paging or swapping to local disks. The ushering process can be implemented with PVM

commands or it can use MOSIX process migration. In each execution, an average memory chunk can be assigned to the nodes using PVM. Figure 2.23 shows the execution time of the memory algorithm using PVM compared with the use of the MOSIX routine.



**FIGURE 2.23** Performance of the memory-ushering algorithm using MOSIX versus PVM. Courtesy of A. Barak and O. La'adan [5]

For a small cluster of eight nodes, the execution times are closer. When the cluster scales to 32 nodes, the MOSIX routine shows a 60 percent reduction in ushering time. Furthermore, MOSIX performs almost the same when the cluster size increases. The PVM ushering time increases monotonically by an average of 3.8 percent per node increase, while MOSIX consistently decreases by 0.4 percent per node increase. The reduction in time results from the fact that the memory and load-balancing algorithms of MOSIX are more scalable than PVM.

## 2.5 Case Studies of Top Supercomputer Systems

This section reviews three top supercomputers that have been singled out as winners in the Top 500 List for the years 2008–2010. The IBM Roadrunner was the world's first petaflops computer, ranked No. 1 in 2008. Subsequently, the Cray XT5 Jaguar became the top system in 2009. In November 2010, China's Tianhe-1A became the fastest system in the world. All three systems are Linux cluster-structured with massive parallelism in term of large number of compute nodes that can execute concurrently.

### 2.5.1 Tianhe-1A: The World Fastest Supercomputer in 2010

In November 2010, the Tianhe-1A was unveiled as a hybrid supercomputer at the 2010 ACM Supercomputing Conference. This system demonstrated a sustained speed of 2.507 Pflops in Linpack Benchmark testing runs and thus became the No. 1 supercomputer in

the 2010 Top 500 list. The system was built by the National University of Defense Technology (NUDT) and was installed in August 2010 at the National Supercomputer Center (NSC), Tianjin, in northern China ([www.nscc.tj.gov.cn](http://www.nscc.tj.gov.cn)). The system is intended as an open platform for research and education. Figure 2.24 shows the Tianhe-1A system installed at NSC.



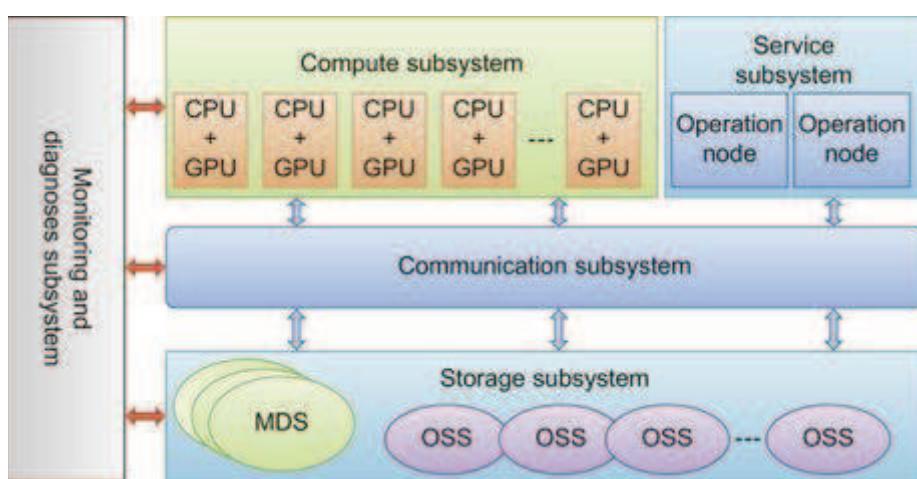
**FIGURE 2.24** The Tianhe-1A system built by the National University of Defense Technology and installed at the National Supercomputer Center, Tianjin, China, in 2010 [11].

### 2.5.1.1 Architecture of Tianhe-1A

Figure 2.25 shows the abstract architecture of the Tianhe-1A system. The system consists of five major components. The compute subsystem houses all the CPUs and GPUs on 7,168 compute nodes. The service subsystem comprises eight operation nodes. The storage subsystem has a large number of shared disks. The monitoring and diagnosis subsystem is used for control and I/O operations. The communication subsystem is composed of switches for connecting to all functional subsystems.

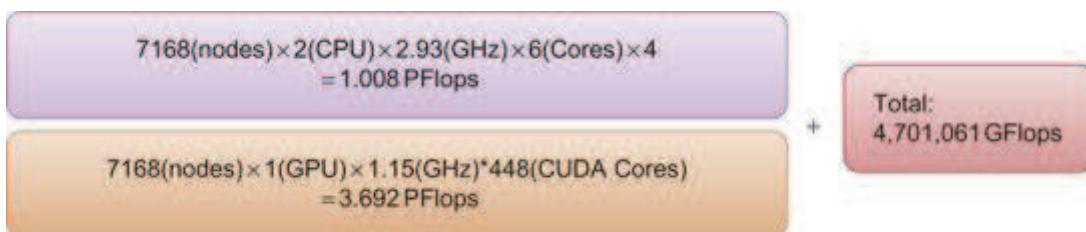
### 2.5.1.2 Hardware Implementation

This system is equipped with 14,336 six-core Xeon E5540/E5450 processors running 2.93 GHz with 7,168 NVIDIA Tesla M2050s. It has 7,168 *compute nodes*, each composed of two Intel Xeon X5670 (Westmere) processors at 2.93 GHz, six cores per socket, and one NVIDIA M2050 GPU connected via PCI-E. A blade has two nodes and is 2U in height (Figure 2.25). The complete system has 14,336 Intel sockets (Westmere) plus 7,168 NVIDIA Fermi boards plus 2,048 Galaxy sockets (the Galaxy processor-based nodes are used as frontend processing for the system). A compute node has two Intel sockets plus a Fermi board plus 32 GB of memory.



**FIGURE 2.25** Abstract architecture of the Tianhe-1A system.

The total system has a theoretical peak of 4.7 Pflops/second as calculated in Figure 2.26. Note that there are 448 CUDA cores in each GPU node. The peak speed is achieved through 14,236 Xeon CPUs (with 380,064 cores) and 7,168 Tesla GPUs (with 448 CUDA cores per node and 3,496,884 CUDA cores in total). There are 3,876,948 processing cores in both the CPU and GPU chips. An operational node has two eight-core Galaxy chips (1 GHz, SPARC architecture) plus 32 GB of memory. The Tianhe-1A system is packaged in 112 compute cabinets, 12 storage cabinets, six communications cabinets, and eight I/O cabinets.



**FIGURE 2.26** Calculation of the theoretical peak speed of Tianhe-1A system.

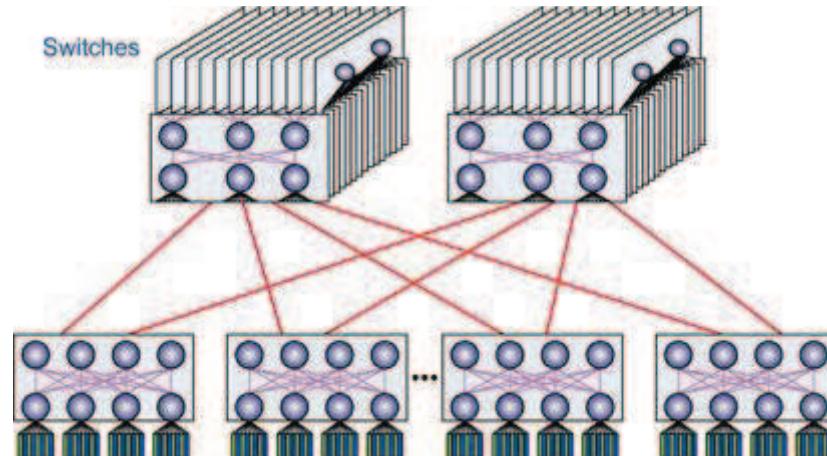
The operation nodes are composed of two eight-core Galaxy FT-1000 chips. These processors were designed by NUDT and run at 1 GHz. The theoretical peak for the eight-core chip is 8 Gflops/second. The complete system has 1,024 of these operational nodes with each having 32 GB of memory. These operational nodes are intended to function as service nodes for job creation and submission. They are not intended as general-purpose computational nodes. Their speed is excluded from the calculation of the peak or sustained speed. The peak speed of the Tianhe-1A is calculated as 3.692 Pflops [11]. It uses 7,168 compute nodes (with 448 CUDA cores/GPU/compute node) in parallel with 14,236 CPUs with six cores in four subsystems.

The system has total disk storage of 2 petabytes implemented with a Lustre clustered file system. There are 262 terabytes of main memory distributed in the cluster system. The Tianhe-1A epitomizes modern heterogeneous CPU/GPU computing, enabling significant achievements in performance, size, and power. The system would require

more than 50,000 CPUs and twice as much floor space to deliver the same performance using CPUs alone. A 2.507-petaflop system built entirely with CPUs would consume at least 12 megawatts, which is three times more power than what the Tianhe-1A consumes.

### 2.5.1.3 ARCH Fat-Tree Interconnect

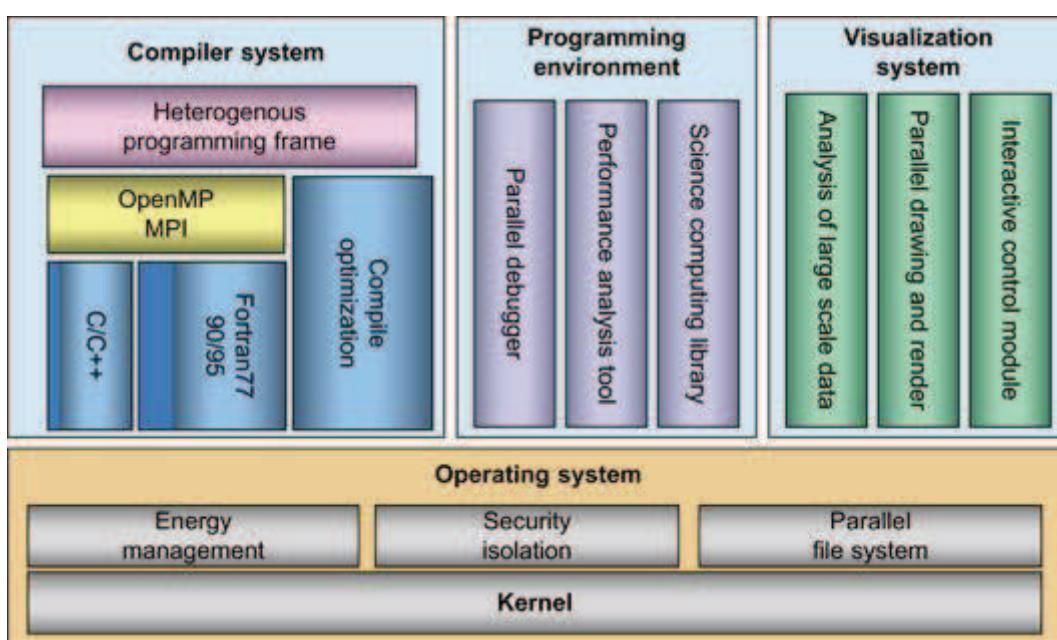
The high performance of the Tianhe-1A is attributed to a customed-designed ARCH interconnect by the NUDT builder. This ARCH is built with the InfiniBand DDR 4X and 98 TB of memory. It assumes a fat-tree architecture as shown in [Figure 2.27](#). The bidirectional bandwidth is 160 Gbps, about twice the bandwidth of the QDR InfiniBand network over the same number of nodes. The ARCH has a latency for a node hop of 1.57 microseconds, and an aggregated bandwidth of 61 Tb/second. At the first stage of the ARCH fat tree, 16 nodes are connected by a 16-port switching board. At the second stage, all parts are connects to eleven 384-port switches. The router and network interface chips are designed by the NUDT team.



**FIGURE 2.27** The ARCH fat-tree interconnect in two stages of high-bandwidth switches [11].

### 2.5.1.4 Software Stack

The software stack on the Tianhe-1A is typical of any high-performance system. It uses Kylin Linux, an operating system developed by NUDT and successfully approved by China's 863 Hi-tech Research and Development Program office in 2006. Kylin is based on Mach and FreeBSD, is compatible with other mainstream operating systems, and supports multiple microprocessors and computers of different structures. Kylin packages include standard open source and public packages, which have been brought onto one system for easy installation. [Figure 2.28](#) depicts the Tianhe-1A software architecture.



**FIGURE 2.28** Software architecture of the Tianhe-1A supercomputer [11].

The system features FORTRAN, C, C++, and Java compilers from Intel (icc 11.1), CUDA, OpenMP, and MPI based on MPICH2 with custom GLEX (Galaxy Express) Channel support. The NUDT builder developed a mathematics library, which is based on Intel's MKL 10.3.1.048 and BLAS for the GPU based on NVIDIA and optimized by NUDT. In addition, a High Productive Parallel Running Environment (HPPRE) was installed. This provides a parallel toolkit based on Eclipse, which is intended to integrate all the tools for editing, debugging, and performance analysis. In addition, the designers provide workflow support for Quality of Service (QoS) negotiations and resource reservations.

### 2.5.1.5 Power Consumption, Space, and Cost

The power consumption of the Tianhe-1A under load is 4.04 MWatt. The system has a footprint of 700 square meters and is cooled by a closed-coupled chilled water-cooling system with forced air. The hybrid architecture consumes less power—about one-third of the 12 MW that is needed to run the system entirely with the multicore CPUs. The budget for the system is 600 million RMB (approximately \$90 million); 200 million RMB comes from the Ministry of Science and Technology (MOST) and 400 million RMB is from the Tianjin local government. It takes about \$20 million annually to run, maintain, and keep the system cool in normal operations.

### 2.5.1.6 Linpack Benchmark Results and Planned Applications

The performance of the Linpack Benchmark on October 30, 2010 was 2.566 Pflops/second on a matrix of 3,600,000 and a  $N_{1/2} = 1,000,000$ . The total time for the run was 3 hours and 22 minutes. The system has an efficiency of 54.58 percent, which is much lower than the 75 percent efficiency achieved by Jaguar and Roadrunner. Listed below are some applications of Tianhe-1A. Most of them are specially tailored to satisfy China's national needs.

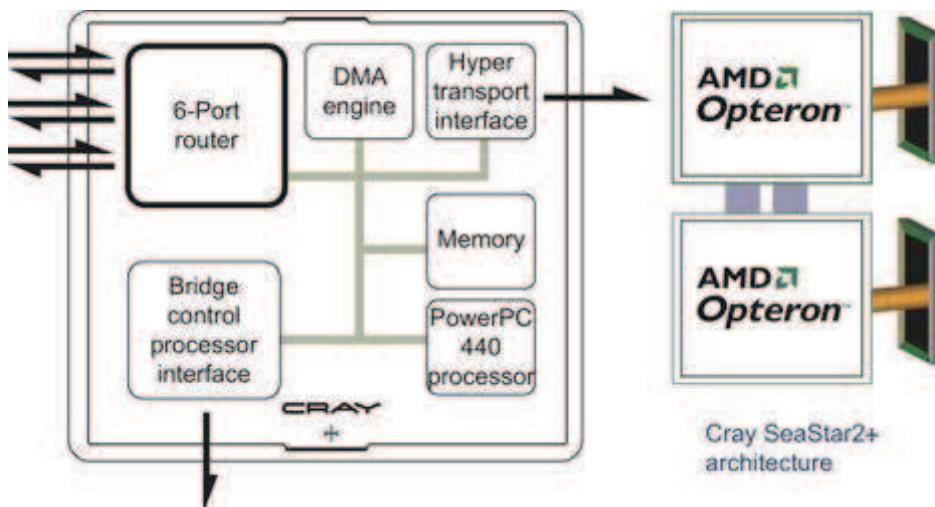
- Parallel AMR (Adaptive Mesh Refinement) method
- Parallel eigenvalue problems
- Parallel fast multipole methods
- Parallel computing models
- Gridmol computational chemistry
- ScGrid middleware, grid portal
- PSEPS parallel symmetric eigenvalue package solvers
- FMM-radar fast multipole methods on radar cross sections
- Transplant many open source software programs
- Sandstorm prediction, climate modeling, EM scattering, or cosmology
- CAD/CAE for automotive industry

## 2.5.2 Cray XT5 Jaguar: The Top Supercomputer in 2009

The Cray XT5 Jaguar was ranked the world's fastest supercomputer in the Top 500 list released at the ACM Supercomputing Conference in June 2010. This system became the second fastest supercomputer in the Top 500 list released in November 2010, when China's Tianhe-1A replaced the Jaguar as the No. 1 machine. This is a scalable MPP system built by Cray, Inc. The Jaguar belongs to Cray's system model XT5-HE. The system is installed at the Oak Ridge National Laboratory, Department of Energy, in the United States. The entire Jaguar system is built with 86 cabinets. The following are some interesting architectural and operational features of the Jaguar system:

- Built with AMD six-core Opteron processors running Linux at a 2.6 GHz clock rate
- Has a total of 224,162 cores on more than 37,360 processors in 88 cabinets in four rows (there are 1,536 or 2,304 processor cores per cabinet)
- Features 8,256 compute nodes and 96 service nodes interconnected by a 3D torus network, built with Cray SeaStar2+ chips
- Attained a sustained speed,  $R_{max}$ , from the Linpack Benchmark test of 1.759 Pflops
- Largest Linpack matrix size tested recorded as  $N_{max} = 5,474,272$  unknowns

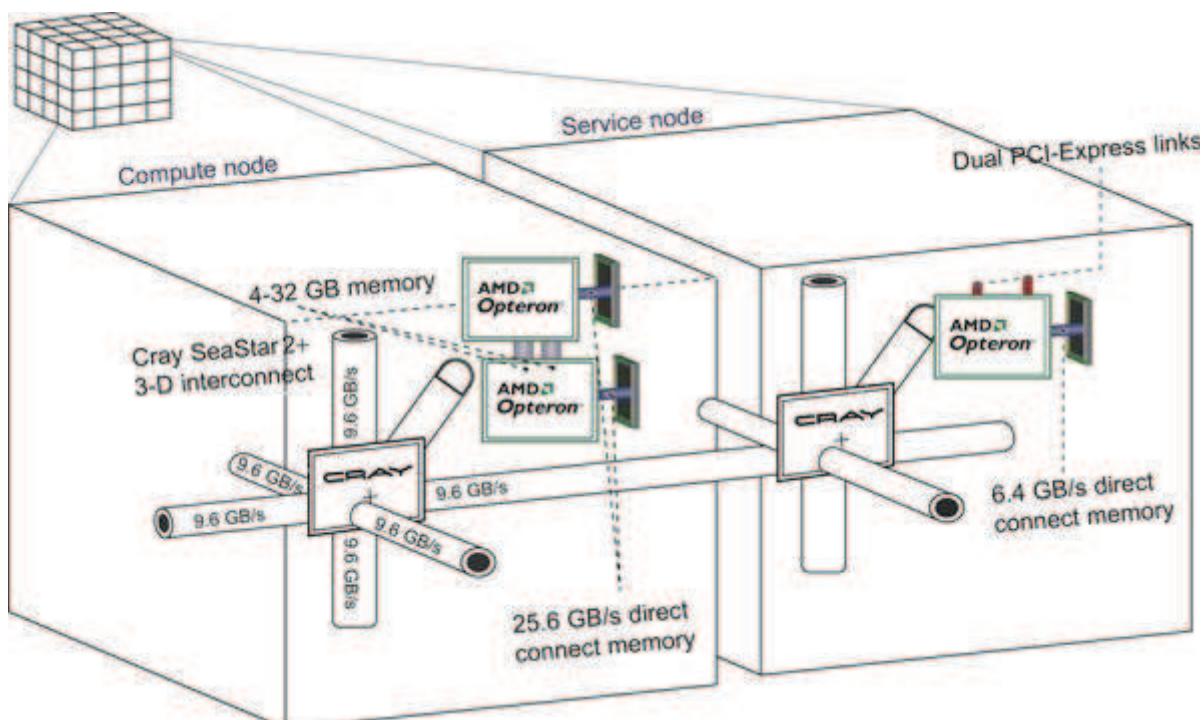
The basic building blocks are the compute blades. The interconnect router in the SeaStar+ chip ([Figure 2.29](#)) provides six high-speed links to six neighbors in the 3D torus, as seen in [Figure 2.30](#). The system is scalable by design from small to large configurations. The entire system has 129 TB of compute memory. In theory, the system was designed with a peak speed of  $R_{peak} = 2.331$  Pflops. In other words, only 75 percent ( $=1.759/2.331$ ) efficiency was achieved in Linpack experiments. The external I/O interface uses 10 Gbps Ethernet and InfiniBand links. MPI 2.1 was applied in message-passing programming. The system consumes 32–43 KW per cabinet. With 160 cabinets, the entire system consumes up to 6.950 MW. The system is cooled with forced cool air, which consumes a lot of electricity.



**FIGURE 2.29** The interconnect SeaStar router chip design in the Cray XT5 Jaguar supercomputer. Courtesy of Cray, Inc. [9] and Oak Ridge National Laboratory, United States, 2009

### 2.5.2.1 3D Torus Interconnect

Figure 2.30 shows the system's interconnect architecture. The Cray XT5 system incorporates a high-bandwidth, low-latency interconnect using the Cray SeaStar2+ router chips. The system is configured with XT5 compute blades with eight sockets supporting dual or quad-core Opterons. The XT5 applies a 3D torus network topology. This SeaStar2+ chip provides six high-speed network links which connect to six neighbors in the 3D torus. The peak bidirectional bandwidth of each link is 9.6 GB/second with sustained bandwidth in excess of 6 GB/second. Each port is configured with an independent router table, ensuring contention-free access for packets.



**FIGURE 2.30** The 3D torus interconnect in the Cray XT5 Jaguar supercomputer. Courtesy of Cray, Inc. [9] and Oak Ridge National Laboratory, United States, 2009

The router is designed with a reliable link-level protocol with error correction and retransmission, ensuring that message-passing traffic reliably reaches its destination without the costly timeout and retry mechanism used in typical clusters. The torus interconnect directly connects all the nodes in the Cray XT5 system, eliminating the cost and complexity of external switches and allowing for easy expandability. This allows systems to economically scale to tens of thousands of nodes—well beyond the capacity of fat-tree switches. The interconnect carries all message-passing and I/O traffic to the global file system.

### **2.5.2.2 Hardware Packaging**

The Cray XT5 family employs an energy-efficient packaging technology, which reduces power use and thus lowers maintenance costs. The system's compute blades are packaged with only the necessary components for building an MPP with processors, memory, and interconnect. In a Cray XT5 cabinet, vertical cooling takes cold air straight from its source—the floor—and efficiently cools the processors on the blades, which are uniquely positioned for optimal airflow. Each processor also has a custom-designed heat sink depending on its position within the cabinet. Each Cray XT5 system cabinet is cooled with a single, high-efficiency ducted turbine fan. It takes 400/480VAC directly from the power grid without transformer and PDU loss.

The Cray XT5 3D torus architecture is designed for superior MPI performance in HPC applications. This is accomplished by incorporating dedicated compute nodes and service nodes. Compute nodes are designed to run MPI tasks efficiently and reliably to completion. Each compute node is composed of one or two AMD Opteron microprocessors (dual or quad core) and direct attached memory, coupled with a dedicated communications resource. Service nodes are designed to provide system and I/O connectivity and also serve as login nodes from which jobs are compiled and launched. The I/O bandwidth of each compute node is designed for 25.6 GB/second performance.

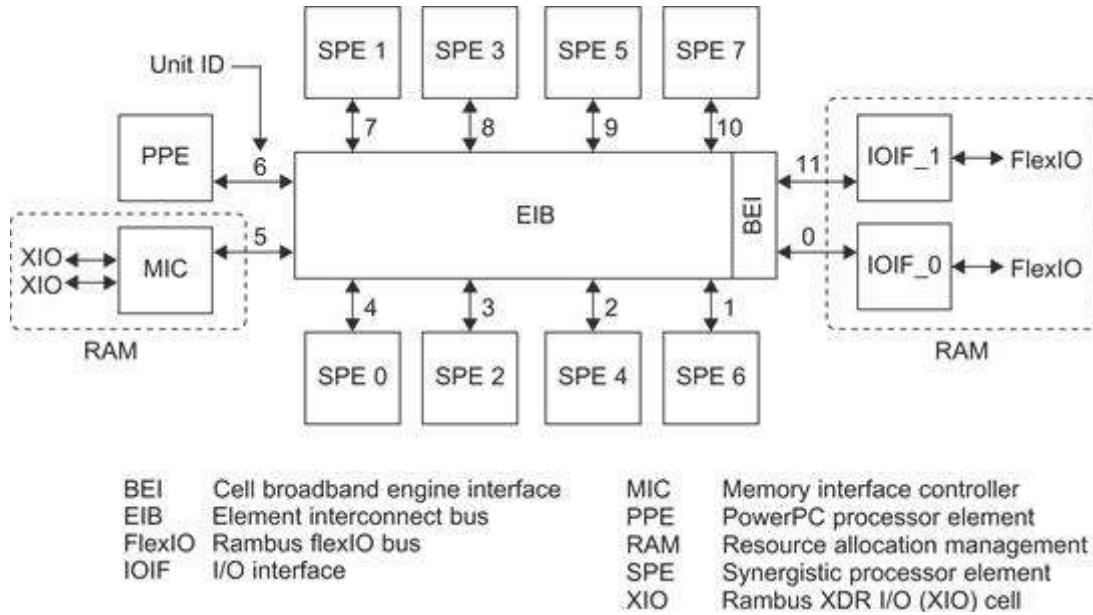
### **2.5.3 IBM Roadrunner: The Top Supercomputer in 2008**

In 2008, the IBM Roadrunner was the first general-purpose computer system in the world to reach petaflops performance. The system has a Linpack performance of 1.456 Pflops and is installed at the Los Alamos National Laboratory (LANL) in New Mexico. Subsequently, Cray's Jaguar topped the Roadrunner in late 2009. The system was used mainly to assess the decay of the U.S. nuclear arsenal. The system has a hybrid design with 12,960 IBM 3.2 GHz PowerXcell 8i CPUs ([Figure 2.26](#)) and 6,480 AMD 1.8 GHz Opteron 2210 dual-core processors. In total, the system has 122,400 cores. Roadrunner is an Opteron cluster accelerated by IBM Cell processors with eight floating-point cores.

#### **2.5.3.1 Processor Chip and Compute Blade Design**

The Cell/B.E. processors provide extraordinary compute power that can be harnessed

from a single multicore chip. As shown in [Figure 2.31](#), the Cell/B.E. architecture supports a very broad range of applications. The first implementation is a single-chip multiprocessor with nine processor elements operating on a shared memory model. The rack is built with TriBlade servers, which are connected by an InfiniBand network. In order to sustain this compute power, the connectivity within each node consists of four PCI Express x8 links, each capable of 2 GB/s transfer rates, with a 2  $\mu$ s latency. The expansion slot also contains the InfiniBand interconnect, which allows communications to the rest of the cluster. The capability of the InfiniBand interconnect is rated at 2 GB/s with a 2  $\mu$ s latency.



**FIGURE 2.31** Schematic of the IBMCell processor architecture. Courtesy of IBM,  
<http://www.redbooks.ibm.com/redpapers/pdfs/redp4477.pdf> [28]

### 2.5.3.2 InfiniBand Interconnect

The Roadrunner cluster was constructed hierarchically. The InfiniBand switches cluster together 18 connected units in 270 racks. In total, the cluster connects 12,960 IBM Power XCell 8i processors and 6,480 Opteron 2210 processors together with a total of 103.6 TB of RAM. This cluster complex delivers approximately 1.3 Pflops. In addition, the system's 18 Com/Service nodes deliver 4.5 Tflops using 18 InfiniBand switches. The second storage units are connected with eight InfiniBand switches. In total, 296 racks are installed in the system. The tiered architecture is constructed in two levels. The system consumes 2.35 MW power, and was the fourth most energy-efficient supercomputer built in 2009.

### 2.5.3.3 Message-Passing Performance

The Roadrunner uses MPI APIs to communicate with the other Opteron processors the application is running on in a typical single-program, multiple-data (SPMD) fashion. The number of compute nodes used to run the application is determined at program launch. The MPI implementation of Roadrunner is based on the open source Open MPI Project,

and therefore is standard MPI. In this regard, Roadrunner applications are similar to other typical MPI applications such as those that run on the IBM Blue Gene solution. Where Roadrunner differs in the sphere of application architecture is how its Cell/B.E. accelerators are employed. At any point in the application flow, the MPI application running on each Opteron can offload computationally complex logic to its subordinate Cell/B.E. processor.

## 2.6 Bibliographic Notes and Homework Problems

Cluster computing has been a hot research area since 1990. Cluster computing was pioneered by DEC and IBM as reported in Pfister [26]. His book provides a good introduction of several key concepts, including SSI and HA. Historically, milestone computer clusters include the VAXcluster running the VMS/OS in 1984, the Tandem Himalaya HA cluster (1994), and the IBM SP2 cluster in 1996. These earlier clusters were covered in [3,7,14,23,26]. In recent years, more than 85 percent of the Top 500 systems are built with cluster configurations [9,11,20,25,28,29].

Annually, IEEE and ACM hold several international conferences related to this topic. They include Cluster Computing (Cluster); Supercomputing Conference (SC); International Symposium on Parallel and Distributed Systems (IPDPS); International Conferences on Distributed Computing Systems (ICDCS); High-Performance Distributed Computing (HPDC); and Clusters, Clouds, and The Grids (CCGrid). There are also several journals related to this topic, including the Journal of Cluster Computing, Journal of Parallel and Distributed Computing (JPDC), and IEEE Transactions on Parallel and Distributed Systems (TPDS).

Cluster applications are assessed in Bader and Pennington [2]. Some figures and examples in this chapter are modified from the earlier book by Hwang and Xu [14]. Buyya has treated cluster computing in two edited volumes [7]. Two books on Linux clusters are [20,23]. HA clusters are treated in [24]. Recent assessment of HPC interconnects can be found in [6,8,12,22]. The Google cluster interconnect was reported by Barroso, et al. [6]. GPUs for supercomputing was discussed in [10]. GPU clusters were studied in [19]. CUDA parallel programming for GPUs is treated in [31]. MOSIX/OS for cluster or grid computing is treated in [4,5,30].

Hwang, Jin, and Ho developed a distributed RAID system for achieving a single I/O space in a cluster of PCs or workstations [13–17]. More details of LSF can be found in Zhou [35]. The Top 500 list was cited from the release in June and November 2010 [25]. The material on the Tianhe-1A can be found in Dongarra [11] and on Wikipedia [29]. The IBM Blue Gene/L architecture was reported by Adiga, et al. [1] and subsequently upgraded to a newer model called the Blue Gene/P solution. The IBM Roadrunner was reported by Kevin, et al. [18] and also in Wikipedia [28]. The Cray XT5 and Jaguar systems are described in [9]. China's Nebulae supercomputer was reported in [27]. Specific cluster applications and checkpointing techniques can be found in [12,16,17,24,32,34]. Cluster applications can be found in [7,15,18,21,26,27,33,34].

## Acknowledgments

This chapter is authored by Kai Hwang of USC and by Jack Dongarra of UTK jointly. Some cluster material are borrowed from the earlier book [14] by Kai Hwang of USC and Zhiwei Xu of the Chinese Academy of Sciences. Valuable suggestions to update the material were made by Rajkumar Buyya of the University of Melbourne.

## References

1. Adiga N, et al. In: *An overview of the blue gene/L supercomputer*. ACM Supercomputing Conference 2002. November 2002; <http://SC-2002.org/paperpdfs/pap.pap207.pdf>.
2. Bader D, Pennington R. Cluster computing applications. *Int J High Perform Comput.* 2001.
3. Baker M, et al. *Cluster computing white paper* January 2001; <http://arxiv.org/abs/cs/0004014>.
4. Barak A, Shiloh A. The MOSIX Management Systems for Linux Clusters. *Multi-Clusters and Clouds White paper* 2010; [www.MOSIX.org//txt\\_pub.html](http://www.MOSIX.org//txt_pub.html).
5. Barak A, La'adan R. The MOSIX multicomputer operating systems for high-performance cluster computing. *Future Gener Comput Syst.* 1998;13:361–372.
6. Barroso L, Dean J, Holzle U. Web search for a planet: The Google cluster architecture. *IEEE Micro.* 2003;23(2):22–28.
7. *High-Performance Cluster Computing*. Vols. 1 and 2. Buyya R, ed. New Jersey: Prentice Hall; 1999.
8. Celebioglu O, Rajagopalan R, Ali R. *Exploring InfiniBand as an HPC cluster interconnect* 2004.
9. Cray, Inc, CrayXT System Specifications. [www.cray.com/Products/XT/Specifications.aspx](http://www.cray.com/Products/XT/Specifications.aspx), January 2010.
10. Dally B. GPU Computing to Exascale and Beyond. In: *Keynote Address*. ACM Supercomputing Conference. November 2010.
11. J. Dongarra, Visit to the National Supercomputer Center in Tianjin, China, Technical Report, University of Tennessee and Oak Ridge National Laboratory, 20 February 2011.
12. Dongarra J. Survey of present and future supercomputer architectures and their interconnects. In: *International Supercomputer Conference*. Heidelberg, Germany. 2004.
13. Hwang K, Jin H, Ho RS. Orthogonal striping and mirroring in distributed RAID for I/O-Centric cluster computing. *IEEE Trans Parallel Distrib Syst.* 2002;13(2):26–44.
14. Hwang K, Xu Z. Support of clustering and availability. In: *Scalable Parallel Computing*. McGraw-Hill 1998; Chapter 9.
15. Hwang K, Wang CM, Wang CL, Xu Z. Resource scaling effects on MPP performance: STAP benchmark implications. *IEEE Trans Parallel Distrib Syst.* 1999;(May):509–527.

16. Hwang K, Jin H, Chow E, Wang CL, Xu Z. Designing SSI clusters with hierarchical checkpointing and single-I/O space. *IEEE Concurrency* 1999;(January):60–69.
17. Jin H, Hwang K. Adaptive sector grouping to reduce false sharing of distributed RAID clusters. *J Clust Comput.* 2001;4(2):133–143.
18. J. Kevin, et al., Entering the petaflop era: the architecture of performance of Roadrunner, [www.c3.lanl.gov/~kei/mypubbib/papers/SC08:Roadrunner.pdf](http://www.c3.lanl.gov/~kei/mypubbib/papers/SC08:Roadrunner.pdf), November 2008.
19. Kindratenko V, et al. *GPU Clusters for High-Performance Computing*. National Center for Supercomputing Applications Urbana, IL: University of Illinois at Urbana-Champaign; 2009.
20. Kopper K. *The Linux Enterprise Cluster: Building a Highly Available Cluster with Commodity Hardware and Free Software* San Francisco, CA: No Starch Press; 2005.
21. Lin SW, Lau RW, Hwang K, Lin X, Cheung PY. Adaptive parallel Image rendering on multiprocessors and workstation clusters. *IEEE Trans Parallel Distrib Syst.* 2001;12(3):241–258.
22. Liu J, Panda DK, et al. Performance comparison of MPI implementations over InfiniBand. *Myrinet and Quadrics* 2003.
23. Lucke R. *Building Clustered Linux Systems* New Jersey: Prentice Hall; 2005.
24. E. Marcus, H. Stern, *Blueprints for High Availability: Designing Resilient Distributed Systems*, Wiley.
25. TOP500.org. Top-500 World's fastest supercomputers, [www.top500.org](http://www.top500.org), November 2010.
26. Pfister GF. *Search of Clusters* second ed. Prentice-Hall 2001.
27. Sun NH. *China's Nebulae Supercomputer* Institute of Computing Technology, Chinese Academy of Sciences July 2010.
28. Wikipedia, IBM Roadrunner. [http://en.wikipedia.org/wiki/IBM\\_Roadrunner](http://en.wikipedia.org/wiki/IBM_Roadrunner), 2010, (accessed 10.01.10).
29. Wikipedia, Tianhe-1. <http://en.wikipedia.org/wiki/Tianhe-1>, 2011, (accessed 5.02.11).
30. Wikipedia, MOSIX. <http://en.wikipedia.org/wiki/MOSIX>, 2011, (accessed 10.02.11).
31. Wikipedia, CUDA. <http://en.wikipedia.org/wiki/CUDA>, 2011, (accessed 19.02.11).
32. Wong K, Franklin M. Checkpointing in distributed computing systems. *J Parallel Distrib Comput* 1996;67–75.
33. Xu Z, Hwang K. *Designing superservers with clusters and commodity components*. Annual Advances in Scalable Computing Singapore: World Scientific; 1999.
34. Xu Z, Hwang K. MPP versus clusters for scalable computing. In: *Proceedings of the 2nd IEEE International Symposium on Parallel Architectures, Algorithms, and Networks*. June 1996;117–123.
35. Zhou S. *LSF: Load Sharing and Batch Queuing Software* Canada: Platform Computing Corp.; 1996.

## Homework Problems

## Problem 2.1

Differentiate and exemplify the following terms related to clusters:

- a. Compact versus slack clusters
- b. Centralized versus decentralized clusters
- c. Homogeneous versus heterogeneous clusters
- d. Enclosed versus exposed clusters
- e. Dedicated versus enterprise clusters

## Problem 2.2

This problem refers to the redundancy technique. Assume that when a node fails, it takes 10 seconds to diagnose the fault and another 30 seconds for the workload to be switched over.

- a. What is the availability of the cluster if planned downtime is ignored?
- b. What is the availability of the cluster if the cluster is taken down one hour per week for maintenance, but one node at a time?

## Problem 2.3

This is a research project to evaluate the cluster architectures of four supercomputers built in recent years. Study the details of the No. 1 supercomputer, the Tianhe-1A, which was announced in the Top 500 list released in November 2010. Your study should include the following:

- a. Conduct an in-depth evaluation of the Tianhe-1A architecture, hardware components, operating system, software support, parallelizing compilers, packaging, cooling, and new applications.
- b. Compare the relative strengths and limitations of the Tianhe-1A with respect to the three case-study systems: the Jaguar, Nebulae, and Roadrunner, studied in [Section 2.5](#). Use tabulations or plot curves, if you find enough benchmark data to conduct the comparison study.

## Problem 2.4

This problem consists of two parts related to cluster computing:

1. Define and distinguish among the following terms on scalability:
  - a. Scalability over machine size
  - b. Scalability over problem size
  - c. Resource scalability
  - d. Generation scalability
2. Explain the architectural and functional differences among three availability cluster configurations: *hot standby*, *active takeover*, and *fault-tolerant clusters*. Give two example commercial cluster systems in each availability cluster configuration. Comment on their relative strengths and weaknesses in commercial applications.

## Problem 2.5

Distinguish between multiprocessors and multicomputers based on their structures, resource sharing, and interprocessor communications.

- a. Explain the differences among UMA, NUMA, COMA, DSM, and NORMA memory models.
- b. What are the additional functional features of a cluster that are not found in a conventional network of autonomous computers?
- c. What are the advantages of a clustered system over a traditional SMP server?

## Problem 2.6

Study the five research virtual cluster projects listed in [Table 2.6](#) and answer the following questions regarding the coverage on COD and Violin experience given in [Sections 2.5.3 and 2.5.4](#):

- a. From the viewpoints of dynamic resource provisioning, evaluate the five virtual clusters and discuss their relative strengths and weaknesses based on the open literature.
- b. Report on the unique contribution from each of the five virtual cluster projects in terms of the hardware setting, software tools, and experimental environments developed and performance results reported.

## Problem 2.7

This problem is related to the use of high-end x86 processors in HPC system construction. Answer the following questions:

- a. Referring to the latest Top 500 list of supercomputing systems, list all systems that have used x86 processors. Identify the processor models and key processor characteristics such as number of cores, clock frequency, and projected performance.
- b. Some have used GPUs to complement the x86 CPUs. Identify those systems that have procured substantial GPUs. Discuss the roles of GPUs to provide peak or sustained flops per dollar.

## Problem 2.8

Assume a sequential computer has 512 MB of main memory and enough disk space. The disk read/write bandwidth for a large data block is 1 MB/second. The following code needs to apply checkpointing:

**do** 1000 iterations

A = foo (C from last iteration) /\* this statement takes 10 minutes \*/

B = goo (A) /\* this statement takes 10 minutes \*/

C = hoo (B) /\* this statement takes 10 minutes \*/

**end do**

A, B, and C are arrays of 120 MB each. All other parts of the code, operating system,

libraries take, at most, 16 MB of memory. Assume the computer fails exactly once, and the time to restore the computer is ignored.

- a. What is the worst-case execution time for the successful completion of the code if checkpointing is performed?
- b. What is the worst-case execution time for the successful completion of the code if plain transparent checkpointing is performed?
- c. Is it beneficial to use forked checkpointing with (b)?
- d. What is the worst-case execution time for the code if user-directed checkpointing is performed? Show the code where user directives are added.
- e. What is the worst-case execution time of the code if forked checkpointing is used with (d)?

## Problem 2.9

Compare the latest Top 500 list with the Top 500 Green List of HPC systems. Discuss a few top winners and losers in terms of energy efficiency in power and cooling costs. Reveal the green-energy winners' stories and report their special design features, packaging, cooling, and management policies that make them the winners. How different are the ranking orders in the two lists? Discuss their causes and implications based on publicly reported data.

## Problem 2.10

This problem is related to processor selection and system interconnects used in building the top three clustered systems with commercial interconnects in the latest Top 500 list.

- a. Compare the processors used in these clusters and identify their strengths and weaknesses in terms of potential peak floating-point performance.
- b. Compare the commercial interconnects of these three clusters. Discuss their potential performance in terms of their topological properties, network latency, bisection bandwidth, and hardware used.

## Problem 2.11

Study [Example 2.6](#) and the original paper [14] reporting the distributed RAID-x architecture and performance results. Answer the following questions with technical justifications or evidence:

- a. Explain how the RAID-x system achieved a single I/O address space across distributed disks attached to cluster nodes.
- b. Explain the functionality of the cooperative disk drivers (CCDs) implemented in the RAID-x system. Comment on its application requirements and scalability based on current PC architecture, SCSI bus, and SCSI disk technology.
- c. Explain why RAID-x has a fault-tolerance capability equal to that of the RAID-5 architecture.

d. Explain the strengths and limitations of RAID-x, compared with other RAID architectures.

## Problem 2.12

Study the relevant material in [Sections 2.2](#) and [2.5](#) and compare the system interconnects of the IBM Blue Gene/L, IBM Roadrunner, and Cray XT5 supercomputers released in the November 2009 Top 500 evaluation. Dig deeper to reveal the details of these systems. These systems may use custom-designed routers in interconnects. Some also use some commercial interconnects and components.

- a. Compare the basic routers or switches used in the three system interconnects in terms of technology, chip design, routing scheme, and claimed message-passing performance.
- b. Compare the topological properties, network latency, bisection bandwidth, and hardware packaging of the three system interconnects.

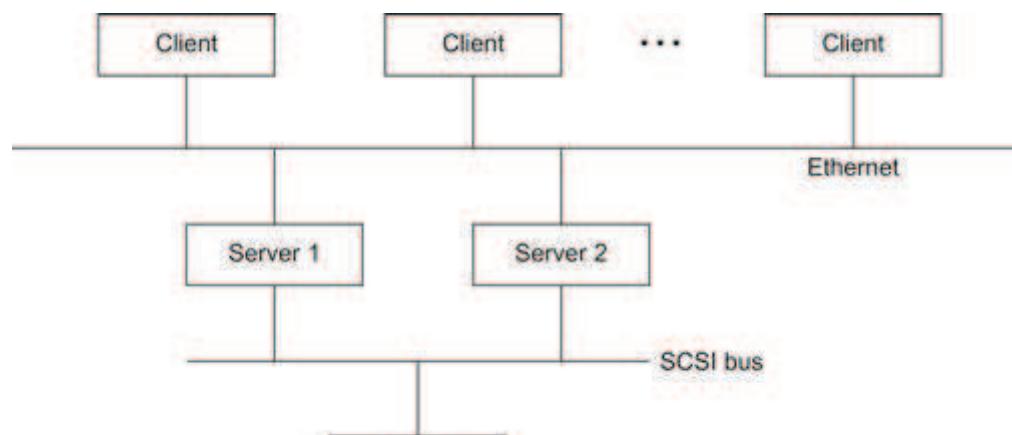
## Problem 2.13

Study the latest and largest commercial HPC clustered system built by SGI, and report on the cluster architecture in the following technical and benchmark aspects:

- a. What is the SGI system model and its specification? Illustrate the cluster architecture with a block diagram and describe the functionality of each building block.
- b. Discuss the claimed peak performance and reported sustained performance from SGI.
- c. What are the unique hardware, software, networking, or design features that contribute to the claimed performance in Part (b)? Describe or illustrate those system features.

## Problem 2.14

Consider in [Figure 2.32](#) a server-client cluster with an active-takeover configuration between two identical servers. The servers share a disk via a SCSI bus. The clients (PCs or workstations) and the Ethernet are fail-free. When a server fails, its workload is switched to the surviving server.



**FIGURE 2.32** An HA cluster with redundant hardware components.

- a. Assume that each server has an MTTF of 200 days and an MTTR of five days. The disk has an MTTF of 800 days and an MTTR of 20 days. In addition, each server is shut down for maintenance for one day every week, during which time that server is considered unavailable. Only one server is shut down for maintenance at a time. The failure rates cover both natural failures and scheduled maintenance. The SCSI bus has a failure rate of 2 percent. The servers and the disk fail independently. The disk and SCSI bus have no scheduled shutdown. The client machine will never fail.
1. The servers are considered available if at least one server is available. What is the combined availability of the two servers?
  2. In normal operation, the cluster must have the SCSI bus, the disk, and at least one server available simultaneously. What are the possible single points of failure in this cluster?
  - b. The cluster is considered unacceptable if both servers fail at the same time. Furthermore, the cluster is declared unavailable when either the SCSI bus or the disk is down. Based on the aforementioned conditions, what is the system availability of the entire cluster?
  - c. Under the aforementioned failure and maintenance conditions, propose an improved architecture to eliminate all single points of failure identified in Part (a).

## Problem 2.15

Study various cluster job scheduling policies in [Table 2.6](#) and answer the following questions. You may need to gather more information from Wikipedia, Google, or other sources if any of the scheduling policies are new to you.

- a. Explain the advantages and disadvantages of nonpreemptive and preemptive scheduling policies and suggest methods to amend the problems.
- b. Repeat Part (a) for static and dynamic scheduling policies.
- c. Repeat Part (a) for dedicated and space-sharing scheduling policies.
- d. Compare the relative performance of time-sharing, independent, and gang scheduling policies.
- e. Compare the relative performance in stay and migrating policies on local jobs against remote jobs.

## Problem 2.16

Study various SSI features and HA support for clusters in [Section 2.3](#) and answer the following questions, providing reasons for your answers. Identify some example cluster systems that are equipped with these features. Comment on their implementation requirements and discuss the operational obstacles to establish each SSI feature in a cluster system.

- a. Single entry point in a cluster environment
- b. Single memory space in a cluster system
- c. Single file hierarchy in a cluster system
- d. Single I/O space in a cluster system
- e. Single network space in a cluster system
- f. Single networking in a cluster system
- g. Single point of control in a cluster system
- h. Single job management in a cluster system
- i. Single user interface in a cluster system
- j. Single process space in a cluster system

## Problem 2.17

Use examples to explain the following terms on cluster job management systems.

- a. Serial jobs versus parallel jobs
- b. Batch jobs versus interactive jobs
- c. Cluster jobs versus foreign (local) jobs
- d. Cluster processes, local processes, and kernel processes
- e. Dedicated mode, space-sharing mode, and timesharing mode
- f. Independent scheduling versus gang scheduling

## Problem 2.18

This problem focuses on the concept of LSF:

- a. Give an example of each of the four types of LSF jobs.
- b. For a 1,000-server cluster, give two reasons why the LSF load-sharing policy is better if (1) the entire cluster has one master LIM or (2) all LIMs are masters.
- c. In the LSF master-election scheme, a node in the “no master” state waits for a time period proportional to the node number before becoming a new master. Why is the wait time proportional to the node number?

## Problem 2.19

This problem is related to the use of MOSIX for cluster computing. Check with the open literature on current features that have been claimed by designers and developers in supporting Linux clusters, GPU clusters, multiclusters, and even virtualized clouds. Discuss the advantages and shortcomings from the user’s perspective.

## Problem 2.20

Compare China’s Tianhe-1A with the Cray Jaguar in terms of their relative strengths and weaknesses in architecture design, resource management, software environment, and reported applications. You may need to conduct some research to find the latest developments regarding these systems. Justify your assessment with reasoning and

evidential information.

## CHAPTER 3

# Virtual Machines and Virtualization of Clusters and Data Centers

## CHAPTER OUTLINE

### Summary

#### 3.1 Implementation Levels of Virtualization

- 3.1.1 Levels of Virtualization Implementation
- 3.1.2 VMM Design Requirements and Providers
- 3.1.3 Virtualization Support at the OS Level
- 3.1.4 Middleware Support for Virtualization

#### 3.2 Virtualization Structures/Tools and Mechanisms

- 3.2.1 Hypervisor and Xen Architecture
- 3.2.2 Binary Translation with Full Virtualization
- 3.2.3 Para-Virtualization with Compiler Support

#### 3.3 Virtualization of CPU, Memory, and I/O Devices

- 3.3.1 Hardware Support for Virtualization
- 3.3.2 CPU Virtualization
- 3.3.3 Memory Virtualization
- 3.3.4 I/O Virtualization
- 3.3.5 Virtualization in Multi-Core Processors

#### 3.4 Virtual Clusters and Resource Management

- 3.4.1 Physical versus Virtual Clusters
- 3.4.2 Live VM Migration Steps and Performance Effects
- 3.4.3 Migration of Memory, Files, and Network Resources
- 3.4.4 Dynamic Deployment of Virtual Clusters

#### 3.5 Virtualization for Data-Center Automation

- 3.5.1 Server Consolidation in Data Centers
- 3.5.2 Virtual Storage Management
- 3.5.3 Cloud OS for Virtualized Data Centers
- 3.5.4 Trust Management in Virtualized Data Centers

#### 3.6 Bibliographic Notes and Homework Problems

[Acknowledgments](#)

[References](#)

[Homework Problems](#)

The reincarnation of virtual machines (VMs) presents a great opportunity for parallel, cluster, grid, cloud, and distributed computing. Virtualization technology benefits the computer and IT industries by enabling users to share expensive hardware resources by multiplexing VMs on the same set of hardware hosts. This chapter covers virtualization levels, VM architectures, virtual networking, virtual cluster construction, and virtualized data-center design and automation in cloud computing. In particular, the designs of dynamically structured clusters, grids, and clouds are presented with VMs and virtual clusters.

virtualization, virtual clusters, data centers, hypervisor, Xen, trust management

## Summary

The reincarnation of *virtual machines* (VMs) presents a great opportunity for parallel, cluster, grid, cloud, and distributed computing. Virtualization technology benefits the computer and IT industries by enabling users to share expensive hardware resources by multiplexing VMs on the same set of hardware hosts. This chapter covers virtualization levels, VM architectures, virtual networking, virtual cluster construction, and virtualized data-center design and automation in cloud computing. In particular, the designs of dynamically structured clusters, grids, and clouds are presented with VMs and virtual clusters.

## 3.1 Implementation Levels of Virtualization

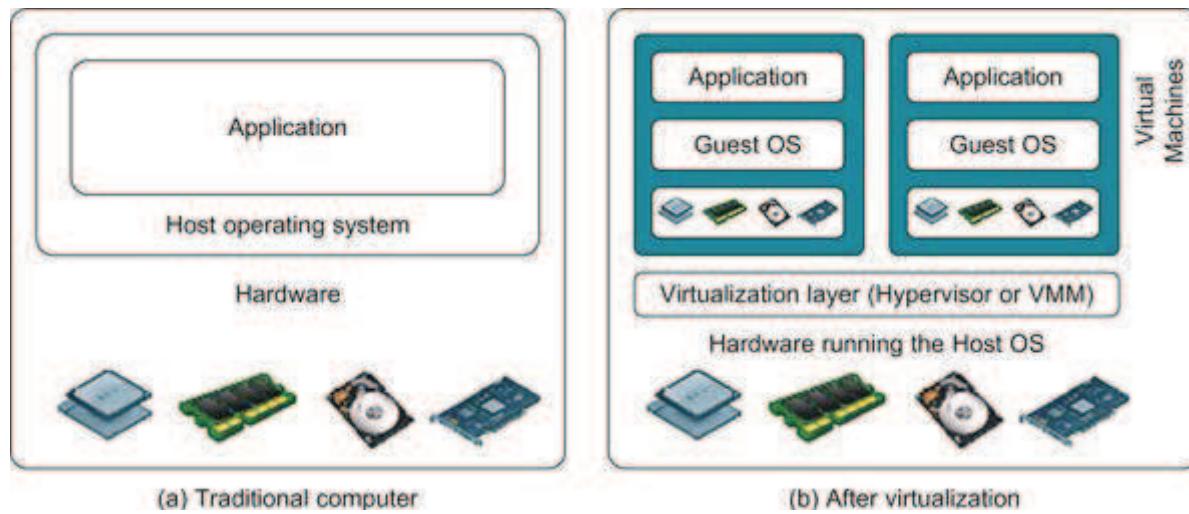
Virtualization is a computer architecture technology by which multiple *virtual machines* (VMs) are multiplexed in the same hardware machine. The idea of VMs can be dated back to the 1960s [53]. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility. Hardware resources (CPU, memory, I/O devices, etc.) or software resources (operating system and software libraries) can be virtualized in various functional layers. This virtualization technology has been revitalized as the demand for distributed and cloud computing increased sharply in recent years [41].

The idea is to separate the hardware from the software to yield better system efficiency. For example, computer users gained access to much enlarged memory space when the concept of *virtual memory* was introduced. Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks, and storage. In this chapter we will discuss VMs and their applications for building distributed systems. According to a 2009 Gartner Report, virtualization was the top strategic technology

poised to change the computer industry. With sufficient storage, any computer platform can be installed in another host computer, even if they use processors with different instruction sets and run with distinct operating systems on the same hardware.

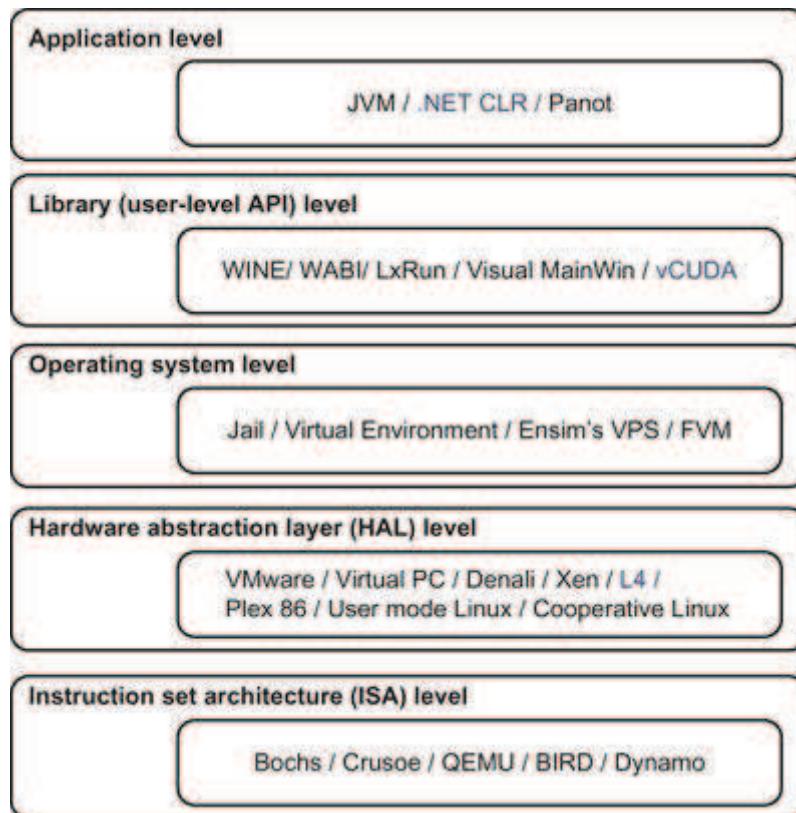
### 3.1.1 Levels of Virtualization Implementation

A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in [Figure 3.1\(a\)](#). After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a *virtualization layer* as shown in [Figure 3.1\(b\)](#). This virtualization layer is known as *hypervisor* or *virtual machine monitor* (VMM) [54]. The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources.



**FIGURE 3.1** The architecture of a computer system before and after virtualization, where VMM stands for virtual machine monitor.

The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. This can be implemented at various operational levels, as we will discuss shortly. The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system. Common virtualization layers include the *instruction set architecture (ISA)* level, hardware level, operating system level, library support level, and application level (see [Figure 3.2](#)).



**FIGURE 3.2** Virtualization ranging from hardware to applications in five abstraction levels.

### 3.1.1.1 Instruction Set Architecture Level

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation. With this approach, it is possible to run a large amount of legacy binary code written for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.

The basic emulation method is through *code interpretation*. An interpreter program interprets the source instructions to target instructions one by one. One source instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow. For better performance, *dynamic binary translation* is desired. This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A *virtual instruction set architecture* (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

### 3.1.1.2 Hardware Abstraction Level

Hardware-level virtualization is performed right on top of the bare hardware. On the one hand, this approach generates a virtual hardware environment for a VM. On the other hand, the process manages the underlying hardware through virtualization. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The

intention is to upgrade the hardware utilization rate by multiple users concurrently. The idea was implemented in the IBM VM/370 in the 1960s. More recently, the Xen hypervisor has been applied to virtualize x86-based machines to run Linux or other guest OS applications. We will discuss hardware virtualization approaches in more detail in [Section 3.3](#).

### **3.1.1.3 Operating System Level**

This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated *containers* on a single physical server and the OS instances to utilize the hardware and software in data centers. The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server. OS-level virtualization is depicted in [Section 3.1.3](#).

### **3.1.1.4 Library Support Level**

Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS. Since most systems provide well-documented APIs, such an interface becomes another candidate for virtualization. Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration. This approach is detailed in [Section 3.1.4](#).

### **3.1.1.5 User-Application Level**

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, *application-level virtualization* is also known as *process-level virtualization*. The most popular approach is to deploy *high level language (HLL)* VMs. In this scenario, the virtualization layer sits as an application program on top of the operating system, and the layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition. Any program written in the HLL and compiled for this VM will be able to run on it. The Microsoft .NET CLR and *Java Virtual Machine (JVM)* are two good examples of this class of VM.

Other forms of application-level virtualization are known as *application isolation*, *application sandboxing*, or *application streaming*. The process involves wrapping the application in a layer that is isolated from the host OS and other applications. The result is an application that is much easier to distribute and remove from user workstations. An example is the LANDesk application virtualization platform which deploys software applications as self-contained, executable files in an isolated environment without

requiring installation, system modifications, or elevated security privileges.

### 3.1.1.6 Relative Merits of Different Approaches

Table 3.1 compares the relative merits of implementing virtualization at various levels. The column headings correspond to four technical merits. “Higher Performance” and “Application Flexibility” are self-explanatory. “Implementation Complexity” implies the cost to implement that particular virtualization level. “Application Isolation” refers to the effort required to isolate resources committed to different VMs. Each row corresponds to a particular level of virtualization.

**Table 3.1**

**Relative Merits of Virtualization at Various Levels (More “X”’s Means Higher Merit, with a Maximum of 5 X’s)**

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

The number of X's in the table cells reflects the advantage points of each implementation level. Five X's implies the best case and one X implies the worst case. Overall, hardware and OS support will yield the highest performance. However, the hardware and application levels are also the most expensive to implement. User isolation is the most difficult to achieve. ISA implementation offers the best application flexibility.

### 3.1.2 VMM Design Requirements and Providers

As mentioned earlier, hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This layer is commonly called the *Virtual Machine Monitor* (VMM) and it manages the hardware resources of a computing system. Each time programs access the hardware the VMM captures the process. In this sense, the VMM acts as a traditional OS. One hardware component, such as the CPU, can be virtualized as several virtual copies. Therefore, several traditional operating systems which are the same or different can sit on the same set of hardware simultaneously.

There are three requirements for a VMM. First, a VMM should provide an environment for programs which is essentially identical to the original machine. Second, programs run in this environment should show, at worst, only minor decreases in speed. Third, a VMM should be in complete control of the system resources. Any program run under a VMM should exhibit a function identical to that which it runs on the original machine directly. Two possible exceptions in terms of differences are permitted with this requirement: differences caused by the availability of system resources and differences caused by

timing dependencies. The former arises when more than one VM is running on the same machine.

The hardware resource requirements, such as memory, of each VM are reduced, but the sum of them is greater than that of the real machine installed. The latter qualification is required because of the intervening level of software and the effect of any other VMs concurrently existing on the same hardware. Obviously, these two differences pertain to performance, while the function a VMM provides stays the same as that of a real machine. However, the identical environment requirement excludes the behavior of the usual time-sharing operating system from being classed as a VMM.

A VMM should demonstrate efficiency in using the VMs. Compared with a physical machine, no one prefers a VMM if its efficiency is too low. Traditional emulators and complete software interpreters (simulators) emulate each instruction by means of functions or macros. Such a method provides the most flexible solutions for VMMs. However, emulators or simulators are too slow to be used as real machines. To guarantee the efficiency of a VMM, a statistically dominant subset of the virtual processor's instructions needs to be executed directly by the real processor, with no software intervention by the VMM. [Table 3.2](#) compares four hypervisors and VMMs that are in use today.

**Table 3.2**

### Comparison of Four VMM and Hypervisor Software Packages

Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

Complete control of these resources by a VMM includes the following aspects: (1) The VMM is responsible for allocating hardware resources for programs; (2) it is not possible for a program to access any resource not explicitly allocated to it; and (3) it is possible under certain circumstances for a VMM to regain control of resources already allocated. Not all processors satisfy these requirements for a VMM. A VMM is tightly related to the architectures of processors. It is difficult to implement a VMM for some types of processors, such as the x86. Specific limitations include the inability to trap on some privileged instructions. If a processor is not designed to support virtualization primarily, it is necessary to modify the hardware to satisfy the three requirements for a VMM. This

is known as hardware-assisted virtualization.

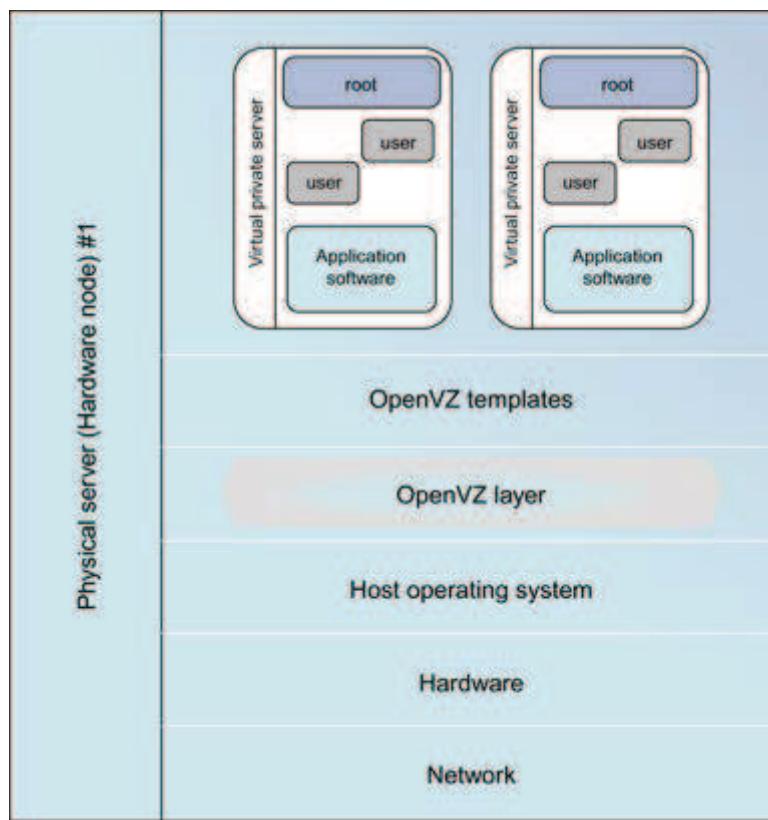
### 3.1.3 Virtualization Support at the OS Level

With the help of VM technology, a new computing mode known as cloud computing is emerging. Cloud computing is transforming the computing landscape by shifting the hardware and staffing costs of managing a computational center to third parties, just like banks. However, cloud computing has at least two challenges. The first is the ability to use a variable number of physical machines and VM instances depending on the needs of a problem. For example, a task may need only a single CPU during some phases of execution but may need hundreds of CPUs at other times. The second challenge concerns the slow operation of instantiating new VMs. Currently, new VMs originate either as fresh boots or as replicates of a template VM, unaware of the current application state. Therefore, to better support cloud computing, a large amount of research and development should be done.

#### 3.1.3.1 Why OS-Level Virtualization?

As mentioned earlier, it is slow to initialize a hardware-level VM because each VM creates its own image from scratch. In a cloud computing environment, perhaps thousands of VMs need to be initialized simultaneously. Besides slow operation, storing the VM images also becomes an issue. As a matter of fact, there is considerable repeated content among VM images. Moreover, full virtualization at the hardware level also has the disadvantages of slow performance and low density, and the need for paravirtualization to modify the guest OS. To reduce the performance overhead of hardware-level virtualization, even hardware modification is needed. OS-level virtualization provides a feasible solution for these hardware-level virtualization issues.

Operating system virtualization inserts a virtualization layer inside an operating system to partition a machine's physical resources. It enables multiple isolated VMs within a single operating system kernel. This kind of VM is often called a *virtual execution environment (VE)*, *Virtual Private System (VPS)*, or simply *container*. From the user's point of view, VEs look like real servers. This means a VE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings. Although VEs can be customized for different people, they share the same operating system kernel. Therefore, OS-level virtualization is also called single-OS image virtualization. [Figure 3.3](#) illustrates operating system virtualization from the point of view of a machine stack.



**FIGURE 3.3** The OpenVZ virtualization layer inside the host OS, which provides some OS images to create VMs quickly. Courtesy of OpenVZ User's Guide [65]

### 3.1.3.2 Advantages of OS Extensions

Compared to hardware-level virtualization, the benefits of OS extensions are twofold: (1) VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability; and (2) for an OS-level VM, it is possible for a VM and its host environment to synchronize state changes when necessary. These benefits can be achieved via two mechanisms of OS-level virtualization: (1) All OS-level VMs on the same physical machine share a single operating system kernel; and (2) the virtualization layer can be designed in a way that allows processes in VMs to access as many resources of the host machine as possible, but never to modify them. In cloud computing, the first and second benefits can be used to overcome the defects of slow initialization of VMs at the hardware level, and being unaware of the current application state, respectively.

### 3.1.3.3 Disadvantages of OS Extensions

The main disadvantage of OS extensions is that all the VMs at operating system level on a single container must have the same kind of guest operating system. That is, although different OS-level VMs may have different operating system distributions, they must pertain to the same operating system family. For example, a Windows distribution such as Windows XP cannot run on a Linux-based container. However, users of cloud computing have various preferences. Some prefer Windows and others prefer Linux or other operating systems. Therefore, there is a challenge for OS-level virtualization in such cases.

**Figure 3.3** illustrates the concept of OS-level virtualization. The virtualization layer is inserted inside the OS to partition the hardware resources for multiple VMs to run their applications in multiple virtual environments. To implement OS-level virtualization, isolated execution environments (VMs) should be created based on a single OS kernel. Furthermore, the access requests from a VM need to be redirected to the VM's local resource partition on the physical machine. For example, the *chroot* command in a UNIX system can create several virtual root directories within a host OS. These virtual root directories are the root directories of all VMs created.

There are two ways to implement virtual root directories: duplicating common resources to each VM partition; or sharing most resources with the host environment and only creating private resource copies on the VM on demand. The first way incurs significant resource costs and overhead on a physical machine. This issue neutralizes the benefits of OS-level virtualization, compared with hardware-assisted virtualization. Therefore, OS-level virtualization is often a second choice.

### 3.1.3.4 Virtualization on Linux or Windows Platforms

By far, most reported OS-level virtualization systems are Linux-based. Virtualization support on the Windows-based platform is still in the research stage. The Linux kernel offers an abstraction layer to allow software processes to work with and operate on resources without knowing the hardware details. New hardware may need a new Linux kernel to support. Therefore, different Linux platforms use patched kernels to provide special support for extended functionality.

However, most Linux platforms are not tied to a special kernel. In such a case, a host can run several VMs simultaneously on the same hardware. **Table 3.3** summarizes several examples of OS-level virtualization tools that have been developed in recent years. Two OS tools (Linux vServer and OpenVZ) support Linux platforms to run other platform-based applications through virtualization. These two OS-level tools are illustrated in **Example 3.1**. The third tool, FVM, is an attempt specifically developed for virtualization on the Windows NT platform.

**Table 3.3**

**Virtualization Support for Linux and Windows NT Platforms**

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
<b>Linux vServer</b> for Linux platforms ( <a href="http://linux-vserver.org/">http://linux-vserver.org/</a> )	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
<b>OpenVZ</b> for Linux platforms [65]; <a href="http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf">http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf</a>	Supports virtualization by creating <i>virtual private servers</i> (VPSes); the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
<b>FVM</b> (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78]	Uses system call interfaces to create VMs at the NY kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

## **Example 3.1**

### **Virtualization Support for the Linux Platform**

OpenVZ is an OS-level tool designed to support Linux platforms to create virtual environments for running VMs under different guest OSes. OpenVZ is an open source container-based virtualization solution built on Linux. To support virtualization and isolation of various subsystems, limited resource management, and checkpointing, OpenVZ modifies the Linux kernel. The overall picture of the OpenVZ system is illustrated in [Figure 3.3](#). Several VPSes can run simultaneously on a physical machine. These VPSes look like normal Linux servers. Each VPS has its own files, users and groups, process tree, virtual network, virtual devices, and IPC through semaphores and messages.

The resource management subsystem of OpenVZ consists of three components: two-level disk allocation, a two-level CPU scheduler, and a resource controller. The amount of disk space a VM can use is set by the OpenVZ server administrator. This is the first level of disk allocation. Each VM acts as a standard Linux system. Hence, the VM administrator is responsible for allocating disk space for each user and group. This is the second-level disk quota. The first-level CPU scheduler of OpenVZ decides which VM to give the time slice to, taking into account the virtual CPU priority and limit settings.

The second-level CPU scheduler is the same as that of Linux. OpenVZ has a set of about 20 parameters which are carefully chosen to cover all aspects of VM operation. Therefore, the resources that a VM can use are well controlled. OpenVZ also supports checkpointing and live migration. The complete state of a VM can quickly be saved to a disk file. This file can then be transferred to another physical machine and the VM can be restored there. It only takes a few seconds to complete the whole process. However, there is still a delay in processing because the established network connections are also migrated.

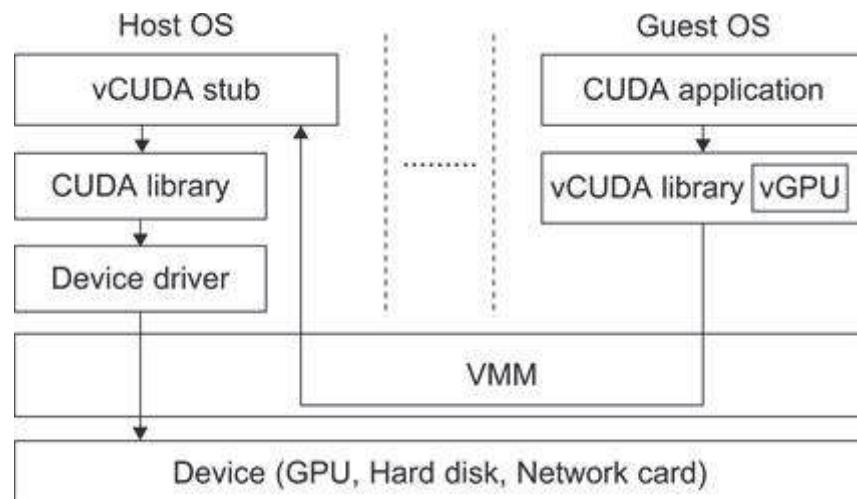
### **3.1.4 Middleware Support for Virtualization**

Library-level virtualization is also known as user-level *Application Binary Interface (ABI)* or API emulation. This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system. API call interception and remapping are the key functions performed. This section provides an overview of several library-level virtualization systems: namely the *Windows Application Binary Interface (WABI)*, lxrun, WINE, Visual MainWin, and vCUDA, which are summarized in [Table 3.4](#).

**Table 3.4****Middleware and Library Support for Virtualization**

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application Platforms
<b>WABI</b> ( <a href="http://docs.sun.com/app/docs/doc/802-6306">http://docs.sun.com/app/docs/doc/802-6306</a> )	Middleware that converts Windows system calls running on x86 PCs to Solaris system calls running on SPARC workstations
<b>Lxrun</b> (Linux Run) ( <a href="http://www.ugcs.caltech.edu/~steven/lxrun/">http://www.ugcs.caltech.edu/~steven/lxrun/</a> )	A system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems such as the SCO OpenServer
<b>WINE</b> ( <a href="http://www.winehq.org/">http://www.winehq.org/</a> )	A library support system for virtualizing x86 processors to run Windows applications under Linux, FreeBSD, and Solaris
<b>Visual MainWin</b> ( <a href="http://www.mainsoft.com/">http://www.mainsoft.com/</a> )	A compiler support system to develop Windows applications using Visual Studio to run on Solaris, Linux, and AIX hosts
<b>vCUDA</b> (Example 3.2) (IEEE IPDPS 2009 [57])	Virtualization support for using general-purpose GPUs to run data-intensive applications under a special guest OS

The WABI offers middleware to convert Windows system calls to Solaris system calls. Lxrun is really a system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems. Similarly, Wine offers library support for virtualizing x86 processors to run Windows applications on UNIX hosts. Visual MainWin offers a compiler support system to develop Windows applications using Visual Studio to run on some UNIX hosts. The vCUDA is explained in Example 3.2 with a graphical illustration in Figure 3.4.

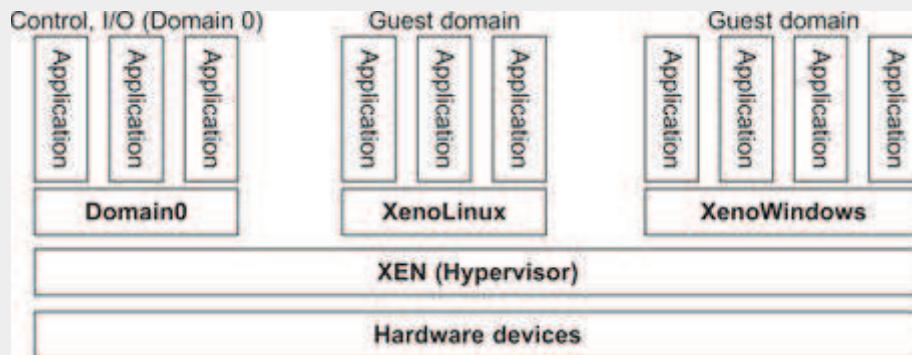


**FIGURE 3.4** Basic concept of the vCUDA architecture. Courtesy of Lin Shi, et al. © IEEE [57]

### Example 3.2

## The vCUDA for Virtualization of General-Purpose GPUs

CUDA is a programming model and library for general-purpose GPUs. It leverages the high performance of GPUs to run compute-intensive applications on host operating systems. However, it is difficult to run CUDA applications on hardware-level VMs directly. vCUDA virtualizes the CUDA library and can be installed on guest OSes. When CUDA applications run on a guest OS and issue a call to the CUDA API, vCUDA intercepts the call and redirects it to the CUDA API running on the host OS. [Figure 3.4](#) shows the basic concept of the vCUDA architecture [57].



**FIGURE 3.5** The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications. Courtesy of P. Barham, et al. [7]

The vCUDA employs a client-server model to implement CUDA virtualization. It consists of three user space components: the vCUDA library, a virtual GPU in the guest OS (which acts as a client), and the vCUDA stub in the host OS (which acts as a server). The vCUDA library resides in the guest OS as a substitute for the standard CUDA library. It is responsible for intercepting and redirecting API calls from the client to the stub. Besides these tasks, vCUDA also creates vGPUs and manages them.

The functionality of a vGPU is threefold: It abstracts the GPU structure and gives applications a uniform view of the underlying hardware; when a CUDA application in the guest OS allocates a device's memory the vGPU can return a local virtual address to the application and notify the remote stub to allocate the real device memory, and the vGPU is responsible for storing the CUDA API flow. The vCUDA stub receives and interprets remote requests and creates a corresponding execution context for the API calls from the guest OS, then returns the results to the guest OS. The vCUDA stub also manages actual physical resource allocation.

## 3.2 Virtualization Structures/Tools and Mechanisms

In general, there are three typical classes of VM architecture. [Figure 3.1](#) showed the architectures of a machine before and after virtualization. Before virtualization, the

operating system manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the operating system. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware. Therefore, different operating systems such as Linux and Windows can run on the same physical machine, simultaneously. Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the *hypervisor* architecture, *para-virtualization*, and *host-based virtualization*. The *hypervisor* is also known as the VMM (*Virtual Machine Monitor*). They both perform the same virtualization operations.

### 3.2.1 Hypervisor and Xen Architecture

The hypervisor supports hardware-level virtualization (see [Figure 3.1\(b\)](#)) on bare metal devices like CPU, memory, disk and network interfaces. The hypervisor software sits directly between the physical hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor. The hypervisor provides *hypercalls* for the guest OSes and applications. Depending on the functionality, a hypervisor can assume a *micro-kernel architecture* like the Microsoft Hyper-V. Or it can assume a *monolithic hypervisor architecture* like the VMware ESX for server virtualization.

A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor. A monolithic hypervisor implements all the aforementioned functions, including those of the device drivers. Therefore, the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor. Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the deployed VM to use.

#### 3.2.1.1 The Xen Architecture

Xen is an open source hypervisor program developed by Cambridge University. Xen is a micro-kernel hypervisor, which separates the policy from the mechanism. The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0, as shown in [Figure 3.5](#). Xen does not include any device drivers natively [7]. It just provides a mechanism by which a guest OS can have direct access to the physical devices. As a result, the size of the Xen hypervisor is kept rather small. Xen provides a virtual environment located between the hardware and the OS. A number of vendors are in the process of developing commercial Xen hypervisors, among them are Citrix XenServer [62] and Oracle VM [42].

The core components of a Xen system are the hypervisor, kernel, and applications. The organization of the three components is important. Like other virtualization systems, many guest OSes can run on top of the hypervisor. However, not all guest OSes are created equal, and one in particular controls the others. The guest OS, which has control ability, is called Domain 0, and the others are called Domain U. Domain 0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being

available. Domain 0 is designed to access hardware directly and manage devices. Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).

For example, Xen is based on Linux and its security level is C2. Its management VM is named Domain 0, which has the privilege to manage other VMs implemented on the same host. If Domain 0 is compromised, the hacker can control the entire system. So, in the VM system, security policies are needed to improve the security of Domain 0. Domain 0, behaving as a VMM, allows users to create, copy, save, read, modify, share, migrate, and roll back VMs as easily as manipulating a file, which flexibly provides tremendous benefits for users. Unfortunately, it also brings a series of security problems during the software life cycle and data lifetime.

Traditionally, a machine's lifetime can be envisioned as a straight line where the current state of the machine is a point that progresses monotonically as the software executes. During this time, configuration changes are made, software is installed, and patches are applied. In such an environment, the VM state is akin to a tree: At any point, execution can go into  $N$  different branches where multiple instances of a VM can exist at any point in this tree at any given time. VMs are allowed to roll back to previous states in their execution (e.g., to fix configuration errors) or rerun from the same point many times (e.g., as a means of distributing dynamic content or circulating a "live" system image).

### 3.2.2 Binary Translation with Full Virtualization

Depending on implementation technologies, hardware virtualization can be classified into two categories: *full virtualization* and *host-based virtualization*. Full virtualization does not need to modify the host OS. It relies on *binary translation* to trap and to virtualize the execution of certain sensitive, nonvirtualizable instructions. The guest OSes and their applications consist of noncritical and critical instructions. In a host-based system, both a host OS and a guest OS are used. A virtualization software layer is built between the host OS and guest OS. These two classes of VM architecture are introduced next.

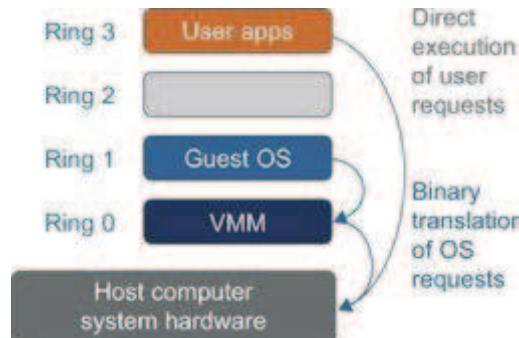
#### 3.2.2.1 Full Virtualization

With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization. Why are only critical instructions trapped into the VMM? This is because binary translation can incur a large performance overhead. Noncritical instructions do not control hardware or threaten the security of the system, but critical instructions do. Therefore, running noncritical instructions on hardware not only can promote efficiency, but also can ensure system security.

#### 3.2.2.2 Binary Translation of Guest OS Requests Using a VMM

This approach was implemented by VMware and many other software companies. As shown in [Figure 3.6](#), VMware puts the VMM at Ring 0 and the guest OS at Ring 1. The

VMM scans the instruction stream and identifies the privileged, control- and behavior-sensitive instructions. When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions. The method used in this emulation is called *binary translation*. Therefore, full virtualization combines binary translation and direct execution. The guest OS is completely decoupled from the underlying hardware. Consequently, the guest OS is unaware that it is being virtualized.



**FIGURE 3.6** Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simple instructions on the same host. Courtesy of VM Ware [71]

The performance of full virtualization may not be ideal, because it involves binary translation which is rather time-consuming. In particular, the full virtualization of I/O-intensive applications is a really a big challenge. Binary translation employs a code cache to store translated hot instructions to improve performance, but it increases the cost of memory usage. At the time of this writing, the performance of full virtualization on the x86 architecture is typically 80 percent to 97 percent that of the host machine.

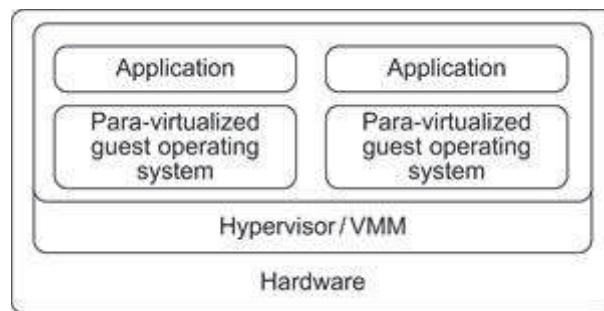
### 3.2.2.3 Host-Based Virtualization

An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware. The guest OSes are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly. This host-based architecture has some distinct advantages, as enumerated next. First, the user can install this VM architecture without modifying the host OS. The virtualizing software can rely on the host OS to provide device drivers and other low-level services. This will simplify the VM design and ease its deployment.

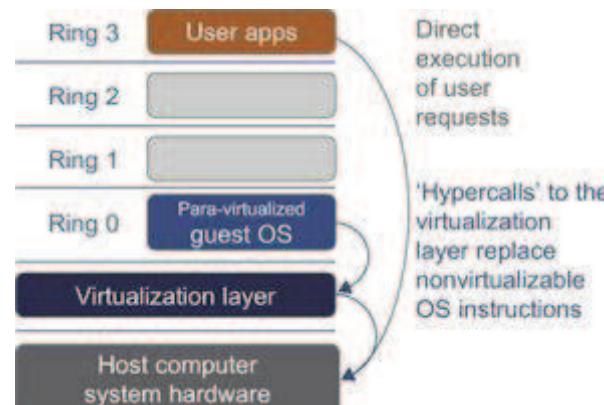
Second, the host-based approach appeals to many host machine configurations. Compared to the hypervisor/VMM architecture, the performance of the host-based architecture may also be low. When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly. When the ISA of a guest OS is different from the ISA of the underlying hardware, binary translation must be adopted. Although the host-based architecture has flexibility, the performance is too low to be useful in practice.

*Para-virtualization* needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system. No one wants to use a VM if it is much slower than using a physical machine. The virtualization layer can be inserted at different positions in a machine software stack. However, para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel.

Figure 3.7 illustrates the concept of a para-virtualized VM architecture. The guest operating systems are para-virtualized. They are assisted by an intelligent compiler to replace the nonvirtualizable OS instructions by hypercalls as illustrated in Figure 3.8. The traditional x86 processor offers four instruction execution rings: Rings 0, 1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed. The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3. The best example of para-virtualization is the KVM to be described below.



**FIGURE 3.7** Para-virtualized VM architecture, which involves modifying the guest OS kernel to replace nonvirtualizable instructions with hypercalls for the hypervisor or the VMM to carry out the virtualization process (See Figure 3.8 for more details.)



**FIGURE 3.8** The use of a para-virtualized guest OS assisted by an intelligent compiler to replace nonvirtualizable OS instructions by hypercalls. Courtesy of VMWare [71]

### **3.2.3.1 Para-Virtualization Architecture**

When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS. According to the x86 ring definition, the virtualization layer should also be installed at Ring 0. Different instructions at Ring 0 may cause some problems. In [Figure 3.8](#), we show that para-virtualization replaces nonvirtualizable instructions with *hypercalls* that communicate directly with the hypervisor or VMM. However, when the guest OS kernel is modified for virtualization, it can no longer run on the hardware directly.

Although para-virtualization reduces the overhead, it has incurred other problems. First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well. Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications. Finally, the performance advantage of para-virtualization varies greatly due to workload variations. Compared with full virtualization, para-virtualization is relatively easy and more practical. The main problem in full virtualization is its low performance in binary translation. To speed up binary translation is difficult. Therefore, many virtualization products employ the para-virtualization architecture. The popular Xen, KVM, and VMware ESX are good examples.

### **3.2.3.2 KVM (Kernel-Based VM)**

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. Memory management and scheduling activities are carried out by the existing Linux kernel. The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine. KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants.

### **3.2.3.3 Para-Virtualization with Compiler Support**

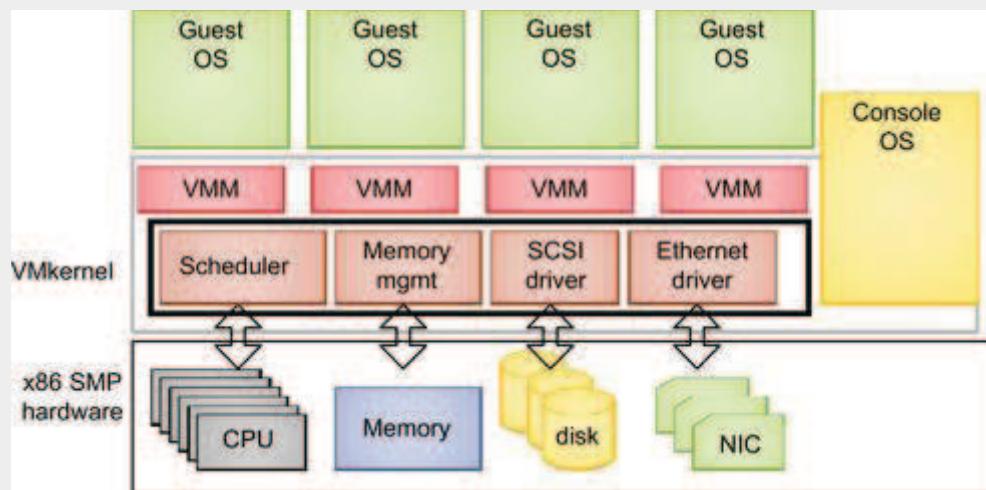
Unlike the full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time. The guest OS kernel is modified to replace the privileged and sensitive instructions with hypercalls to the hypervisor or VMM. Xen assumes such a para-virtualization architecture.

The guest OS running in a guest domain may run at Ring 1 instead of at Ring 0. This implies that the guest OS may not be able to execute some privileged and sensitive instructions. The privileged instructions are implemented by hypercalls to the hypervisor. After replacing the instructions with hypercalls, the modified guest OS emulates the behavior of the original guest OS. On an UNIX system, a system call involves an interrupt or service routine. The hypercalls apply a dedicated service routine in Xen.

### Example 3.3

## VMware ESX Server for Para-Virtualization

VMware pioneered the software market for virtualization. The company has developed virtualization tools for desktop systems and servers as well as virtual infrastructure for large data centers. ESX is a VMM or a hypervisor for bare-metal x86 symmetric multiprocessing (SMP) servers. It accesses hardware resources such as I/O directly and has complete resource management control. An ESX-enabled server consists of four components: a virtualization layer, a resource manager, hardware interface components, and a service console, as shown in [Figure 3.9](#). To improve performance, the ESX server employs a para-virtualization architecture in which the VM kernel interacts directly with the hardware without involving the host OS.



**FIGURE 3.9** The VMware ESX server architecture using para-virtualization. Courtesy of VMware [71]

The VMM layer virtualizes the physical hardware resources such as CPU, memory, network and disk controllers, and human interface devices. Every VM has its own set of virtual hardware resources. The resource manager allocates CPU, memory disk, and network bandwidth and maps them to the virtual hardware resource set of each VM created. Hardware interface components are the device drivers and the VMware ESX Server File System. The service console is responsible for booting the system, initiating the execution of the VMM and resource manager, and relinquishing control to those layers. It also facilitates the process for system administrators.

## 3.3 Virtualization of CPU, Memory, and I/O Devices

To support virtualization, processors such as the x86 employ a special running mode and

instructions, known as *hardware-assisted virtualization*. In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM. To save processor states, mode switching is completed by hardware. For the x86 architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

### 3.3.1 Hardware Support for Virtualization

Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions. In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack. [Example 3.4](#) discusses Intel's hardware support approach.

At the time of this writing, many hardware virtualization products were available. The VMware Workstation is a VM software suite for x86 and x86-64 computers. This software suite allows users to set up multiple x86 and x86-64 virtual computers and to use one or more of these VMs simultaneously with the host operating system. The VMware Workstation assumes the host-based virtualization. Xen is a hypervisor for use in IA-32, x86-64, Itanium, and PowerPC 970 hosts. Actually, Xen modifies Linux as the lowest and most privileged layer, or a hypervisor.

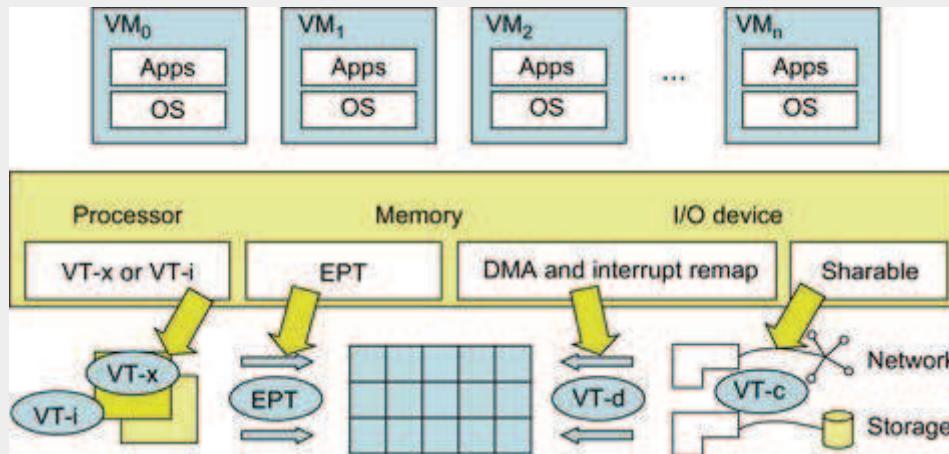
One or more guest OS can run on top of the hypervisor. KVM (*Kernel-based Virtual Machine*) is a Linux kernel virtualization infrastructure. KVM can support hardware-assisted virtualization and paravirtualization by using the Intel VT-x or AMD-v and VirtIO framework, respectively. The VirtIO framework includes a paravirtual Ethernet card, a disk I/O controller, a balloon device for adjusting guest memory usage, and a VGA graphics interface using VMware drivers.

#### Example 3.4

#### Hardware Support for Virtualization in the Intel x86 Processor

Since software-based virtualization techniques are complicated and incur performance overhead, Intel provides a hardware-assist technique to make virtualization easy and improve performance. [Figure 3.10](#) provides an overview of Intel's full virtualization techniques. For processor virtualization, Intel offers the VT-x or VT-i technique. VT-x adds a privileged mode (VMX Root Mode) and some instructions to processors. This enhancement traps all

sensitive instructions in the VMM automatically. For memory virtualization, Intel offers the EPT, which translates the virtual address to the machine's physical addresses to improve performance. For I/O virtualization, Intel implements VT-d and VT-c to support this.



**FIGURE 3.10** Intel hardware support for virtualization of processor, memory, and I/O devices.

Modified from [68], Courtesy of Lizhong Chen, USC

### 3.3.2 CPU Virtualization

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories: *privileged instructions*, *control-sensitive instructions*, and *behavior-sensitive instructions*. Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode. Control-sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control- and behavior-sensitive instructions of a VM are executed, they are trapped in the VMM. In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system. However, not all CPU architectures are virtualizable. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions. On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as *SGDT* and *SMSW*, are not privileged instructions. When

these instructions execute in virtualization, they cannot be trapped in the VMM.

On a native UNIX-like system, a system call triggers the  $80h$  interrupt and passes control to the OS kernel. The interrupt handler in the kernel is then invoked to process the system call. On a paravirtualization system such as Xen, a system call in the guest OS first triggers the  $80h$  interrupt normally. Almost at the same time, the  $82h$  interrupt in the hypervisor is triggered. Incidentally, control is passed on to the hypervisor as well. When the hypervisor completes its task for the guest OS system call, it passes control back to the guest OS kernel. Certainly, the guest OS kernel may also invoke the hypercall while it's running. Although paravirtualization of a CPU lets unmodified applications run in the VM, it causes a small performance penalty.

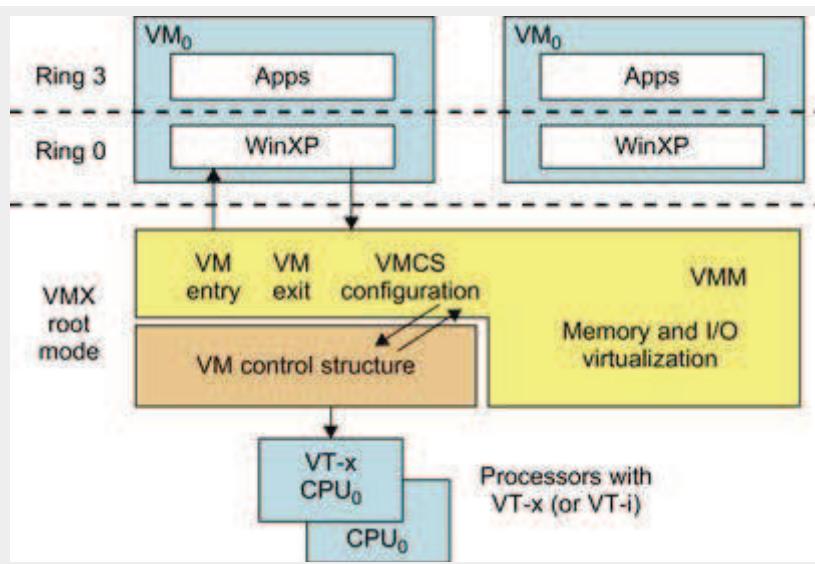
### 3.3.2.1 Hardware-Assisted CPU Virtualization

This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.

#### Example 3.5

#### Intel Hardware-Assisted CPU Virtualization

Although x86 processors are not virtualizable primarily, great effort is taken to virtualize them. They are used widely in comparing RISC processors that the bulk of x86-based legacy systems cannot discard easily. Virtualization of x86 processors is detailed in the following sections. Intel's VT-x technology is an example of hardware-assisted virtualization, as shown in [Figure 3.11](#). Intel calls the privilege level of x86 processors the VMX Root Mode. In order to control the start and stop of a VM and allocate a memory page to maintain the CPU state for VMs, a set of additional instructions is added. At the time of this writing, Xen, VMware, and the Microsoft Virtual PC all implement their hypervisors by using the VT-x technology.



**FIGURE 3.11** Intel hardware-assisted CPU virtualization. Modified from [68], Courtesy of Lizhong Chen, USC

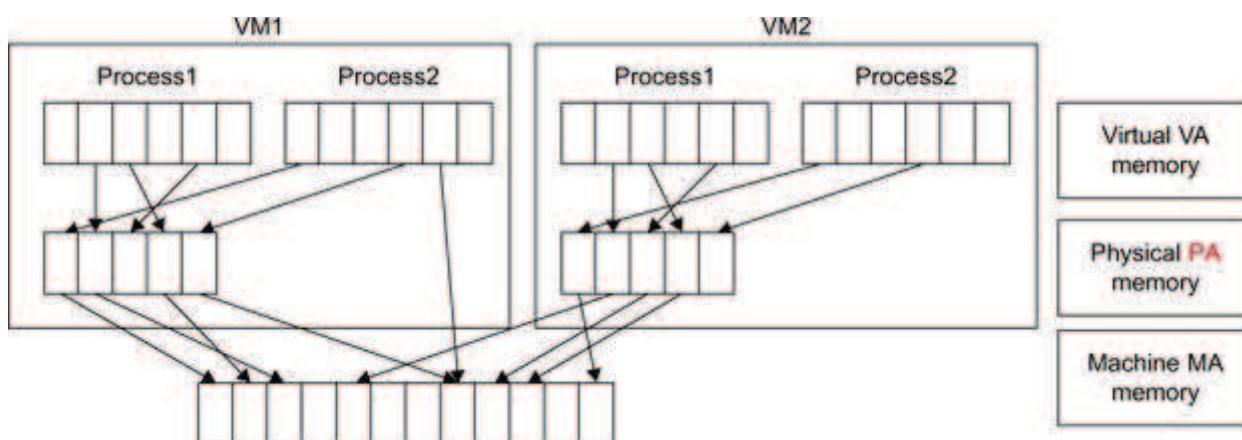
Generally, hardware-assisted virtualization should have high efficiency. However, since the transition from the hypervisor to the guest OS incurs high overhead switches between processor modes, it sometimes cannot outperform binary translation. Hence, virtualization systems such as VMware now use a hybrid approach, in which a few tasks are offloaded to the hardware but the rest is still done in software. In addition, para-virtualization and hardware-assisted virtualization can be combined to improve the performance further.

### 3.3.3 Memory Virtualization

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of *virtual memory* to *machine memory* using page tables, which is a one-stage mapping from virtual memory to machine memory. All modern x86 CPUs include a *memory management unit* (MMU) and a *translation lookaside buffer* (TLB) to optimize virtual memory performance. However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the *physical memory* of the VMs.

That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory. Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 3.12 shows the two-level memory

mapping procedure.



**FIGURE 3.12** Two-level memory mapping procedure. Courtesy of R. Rblig, et al. [68]

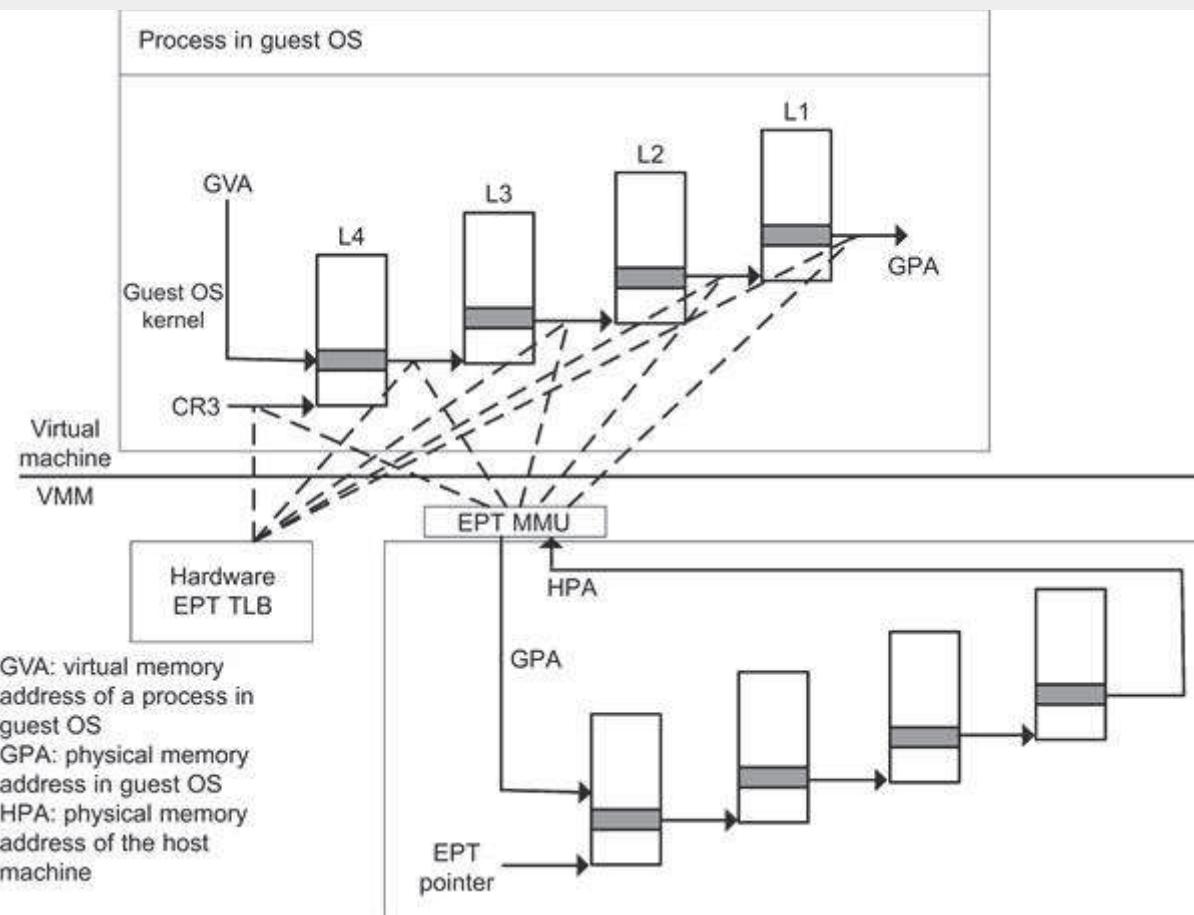
Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table. Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS. Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor. Since modern operating systems maintain a set of page tables for every process, the shadow page tables will get flooded. Consequently, the performance overhead and cost of memory will be very high.

VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup. The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007. It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging.

### Example 3.6

#### Extended Page Table by Intel for Memory Virtualization

Since the efficiency of the software shadow page table technique was too low, Intel developed a hardware-based EPT technique to improve it, as illustrated in Figure 3.13. In addition, Intel offers a Virtual Processor ID (VPID) to improve use of the TLB. Therefore, the performance of memory virtualization is greatly improved. In Figure 3.13, the page tables of the guest OS and EPT are all four-level.



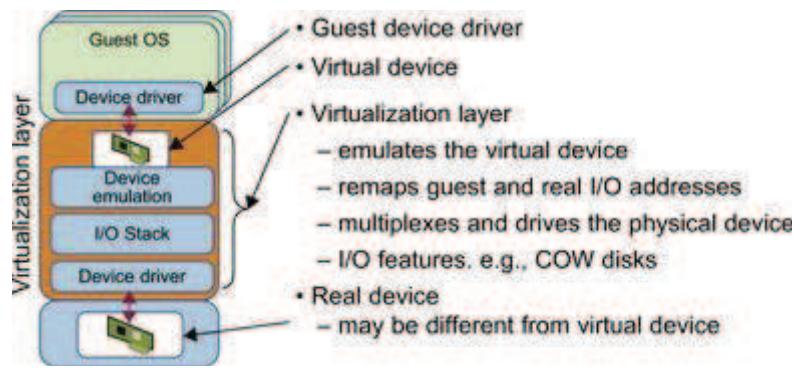
**FIGURE 3.13** Memory virtualization using EPT by Intel (the EPT is also known as the shadow page table [68]).

When a virtual address needs to be translated, the CPU will first look for the L4 page table pointed to by Guest CR3. Since the address in Guest CR3 is a physical address in the guest OS, the CPU needs to convert the Guest CR3 GPA to the host physical address (HPA) using EPT. In this procedure, the CPU will check the EPT TLB to see if the translation is there. If there is no required translation in the EPT TLB, the CPU will look for it in the EPT. If the CPU cannot find the translation in the EPT, an EPT violation exception will be raised.

When the GPA of the L4 page table is obtained, the CPU will calculate the GPA of the L3 page table by using the GVA and the content of the L4 page table. If the entry corresponding to the GVA in the L4 page table is a page fault, the CPU will generate a page fault interrupt and will let the guest OS kernel handle the interrupt. When the PGA of the L3 page table is obtained, the CPU will look for the EPT to get the HPA of the L3 page table, as described earlier. To get the HPA corresponding to a GVA, the CPU needs to look for the EPT five times, and each time, the memory needs to be accessed four times. Therefore, there are 20 memory accesses in the worst case, which is still very slow. To overcome this shortcoming, Intel increased the size of the EPT TLB to decrease the number of memory accesses.

### 3.3.4 I/O Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware. At the time of this writing, there are three ways to implement I/O virtualization: full device emulation, para-virtualization, and direct I/O. Full device emulation is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices. All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices. The full device emulation approach is shown in [Figure 3.14](#).



**FIGURE 3.14** Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use. Courtesy of V. Chadha, et al. [10] and Y. Dong, et al. [15]

A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates [10,15]. The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

Direct I/O virtualization lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs. However, current direct I/O virtualization implementations focus on networking for mainframes. There are a lot of challenges for commodity hardware devices. For example, when a physical device is reclaimed (required by workload migration) for later reassignment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system. Since software-based I/O virtualization requires a very high overhead of device emulation, hardware-assisted I/O virtualization is critical. Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts. The architecture of VT-d provides the flexibility to support multiple usage models that may

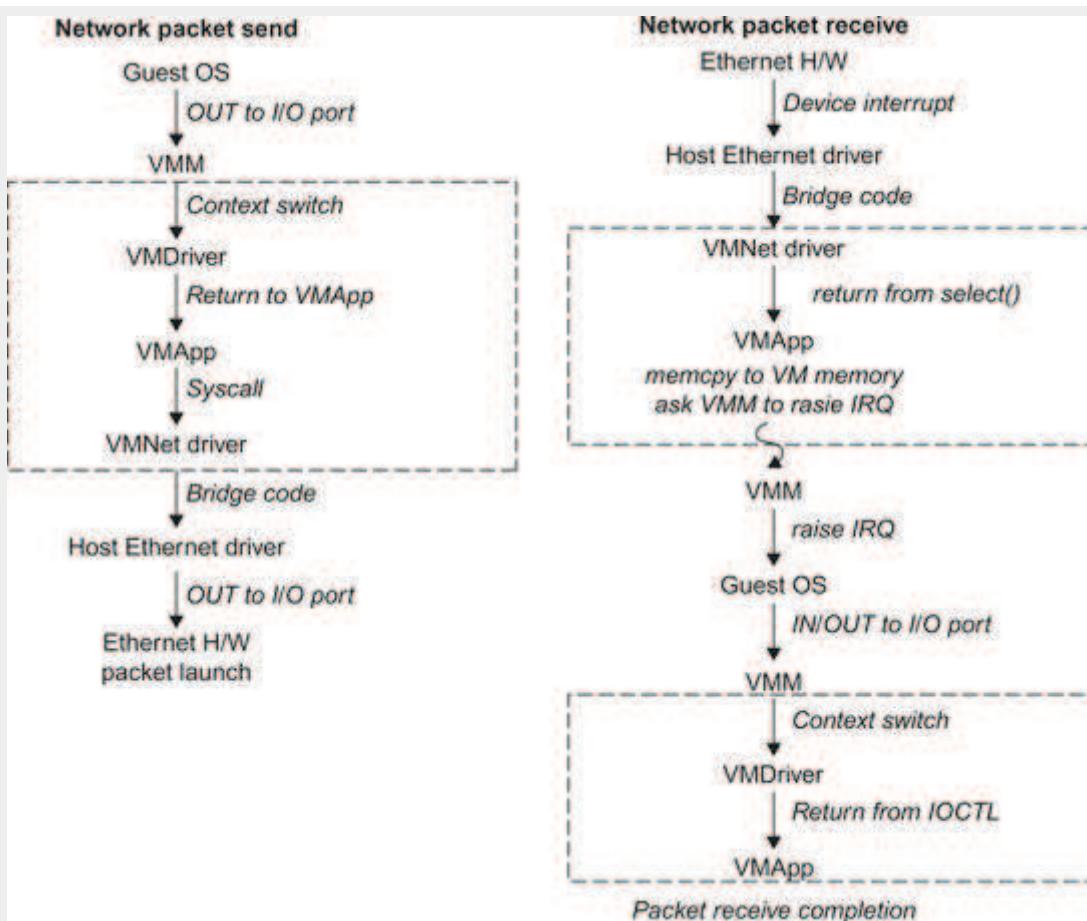
run unmodified, special-purpose, or “virtualization-aware” guest OSes.

Another way to help I/O virtualization is via self-virtualized I/O (SV-I/O) [47]. The key idea of SV-I/O is to harness the rich resources of a multicore processor. All tasks associated with virtualizing an I/O device are encapsulated in SV-I/O. It provides virtual devices and an associated access API to VMs and a management API to the VMM. SV-I/O defines one virtual interface (VIF) for every kind of virtualized I/O device, such as virtual network interfaces, virtual block devices (disk), virtual camera devices, and others. The guest OS interacts with the VIFs via VIF device drivers. Each VIF consists of two message queues. One is for outgoing messages to the devices and the other is for incoming messages from the devices. In addition, each VIF has a unique ID for identifying it in SV-I/O.

### Example 3.7

#### VMware Workstation for I/O Virtualization

The VMware Workstation runs as an application. It leverages the I/O device support in guest OSes, host OSes, and VMM to implement I/O virtualization. The application portion (VMApp) uses a driver loaded into the host operating system (VMDriver) to establish the privileged VMM, which runs directly on the hardware. A given physical processor is executed in either the host world or the VMM world, with the VMDriver facilitating the transfer of control between the two worlds. The VMware Workstation employs full device emulation to implement I/O virtualization. [Figure 3.15](#) shows the functional blocks used in sending and receiving packets via the emulated virtual NIC.



**FIGURE 3.15** Functional blocks involved in sending and receiving network packets. Courtesy of VMWare [71]

The virtual NIC models an AMD Lance Am79C970A controller. The device driver for a Lance controller in the guest OS initiates packet transmissions by reading and writing a sequence of virtual I/O ports; each read or write switches back to the VMApp to emulate the Lance port accesses. When the last OUT instruction of the sequence is encountered, the Lance emulator calls a normal `write()` to the VMNet driver. The VMNet driver then passes the packet onto the network via a host NIC and then the VMApp switches back to the VMM. The switch raises a virtual interrupt to notify the guest device driver that the packet was sent. Packet receives occur in reverse.

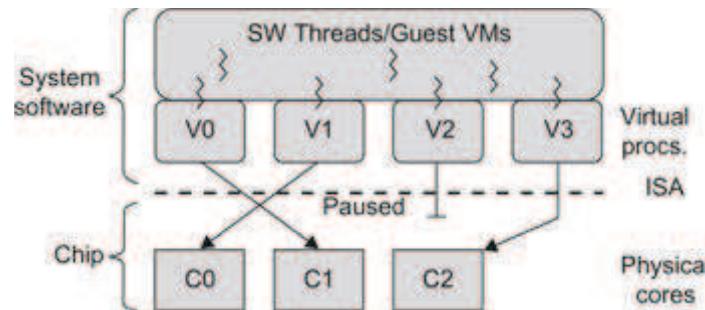
### 3.3.5 Virtualization in Multi-Core Processors

Virtualizing a multi-core processor is relatively more complicated than virtualizing a uni-core processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, multi-core virtualization has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier. The second challenge has spawned research involving scheduling algorithms and resource management policies. Yet these efforts cannot balance well among performance, complexity, and other issues. What is worse, as technology scales, a new challenge called *dynamic heterogeneity* is emerging to mix the fat CPU core and thin GPU cores on the same chip, which further complicates the multi-core or many-core resource management. The dynamic heterogeneity of hardware infrastructure mainly comes from less reliable transistors and increased complexity in using the transistors [33,66].

### 3.3.5.1 Physical versus Virtual Processor Cores

Wells, et al. [74] proposed a multicore virtualization method to allow hardware designers to get an abstraction of the low-level details of the processor cores. This technique alleviates the burden and inefficiency of managing hardware resources by software. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor). Figure 3.16 illustrates the technique of a software-visible VCPU moving from one core to another and temporarily suspending execution of a VCPU when there are no appropriate cores on which it can run.



**FIGURE 3.16** Multicore virtualization method that exposes four VCPUs to the software, when only three cores are actually present. Courtesy of Wells, et al. [74]

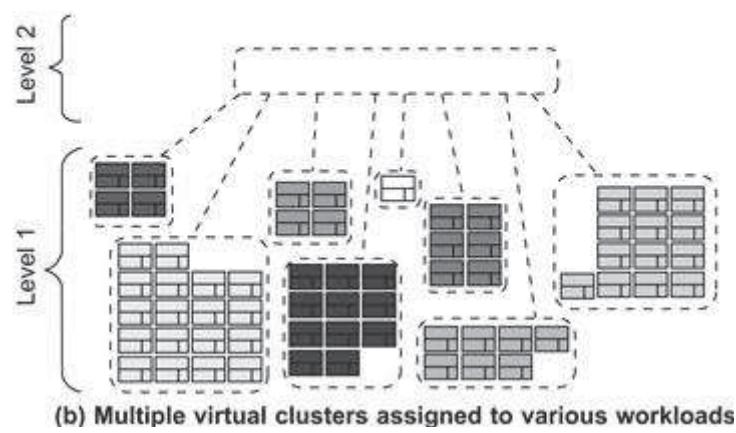
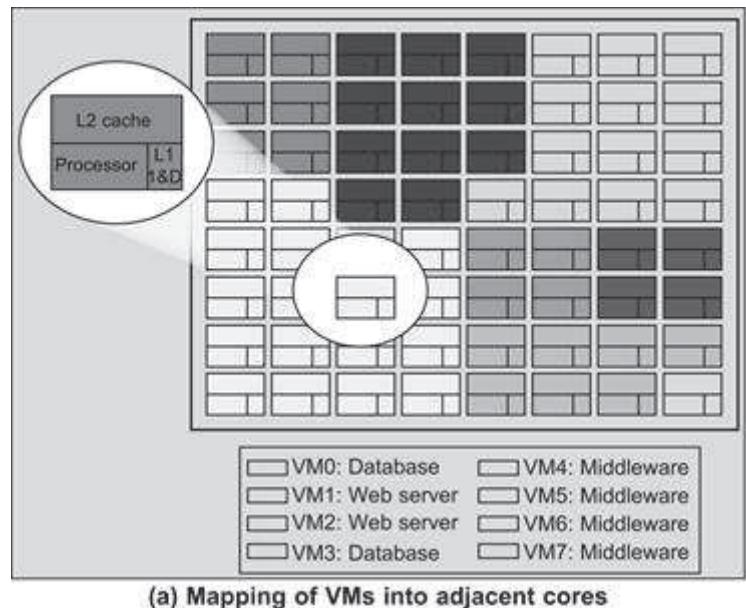
### 3.3.5.2 Virtual Hierarchy

The emerging many-core *chip multiprocessors* (CMPs) provides a new computing landscape. Instead of supporting time-sharing jobs on one or a few cores, we can use the abundant cores in a space-sharing, where single-threaded or multithreaded jobs are simultaneously assigned to separate groups of cores for long time intervals. This idea was originally suggested by Marty and Hill [39]. To optimize for space-shared workloads, they propose using *virtual hierarchies* to overlay a coherence and caching hierarchy onto a physical processor. Unlike a fixed physical hierarchy, a virtual hierarchy can adapt to fit how the work is space shared for improved performance and performance isolation.

Today's many-core CMPs use a physical hierarchy of two or more cache levels that statically determine the cache allocation and mapping. A *virtual hierarchy* is a cache hierarchy that can adapt to fit the workload or mix of workloads [39]. The hierarchy's first

level locates data blocks close to the cores needing them for faster access, establishes a shared-cache domain, and establishes a point of coherence for faster communication. When a miss leaves a tile, it first attempts to locate the block (or sharers) within the first level. The first level can also provide isolation between independent workloads. A miss at the L1 cache can invoke the L2 access.

The idea is illustrated in [Figure 3.17\(a\)](#). Space sharing is applied to assign three workloads to three clusters of virtual cores: namely VM0 and VM3 for database workload, VM1 and VM2 for web server workload, and VM4–VM7 for middleware workload. The basic assumption is that each workload runs in its own VM. However, space sharing applies equally within a single operating system. Statically distributing the directory among tiles can do much better, provided operating systems or hypervisors carefully map virtual pages to physical frames. Marty and Hill suggested a two-level virtual coherence and caching hierarchy that harmonizes with the assignment of tiles to the virtual clusters of VMs.



**FIGURE 3.17** CMP server consolidation by space-sharing of VMs into many cores forming multiple virtual clusters to execute various workloads. Courtesy of Marty and Hill [39]

[Figure 3.17\(b\)](#) illustrates a logical view of such a virtual cluster hierarchy in two levels. Each VM operates in an isolated fashion at the first level. This will minimize both miss

access time and performance interference with other workloads or VMs. Moreover, the shared resources of cache capacity, inter-connect links, and miss handling are mostly isolated between VMs. The second level maintains a globally shared memory. This facilitates dynamically repartitioning resources without costly cache flushes. Furthermore, maintaining globally shared memory minimizes changes to existing system software and allows virtualization features such as content-based page sharing. A virtual hierarchy adapts to space-shared workloads like multiprogramming and server consolidation. [Figure 3.17](#) shows a case study focused on consolidated server workloads in a tiled architecture. This many-core mapping scheme can also optimize for space-shared multiprogrammed workloads in a single-OS environment.

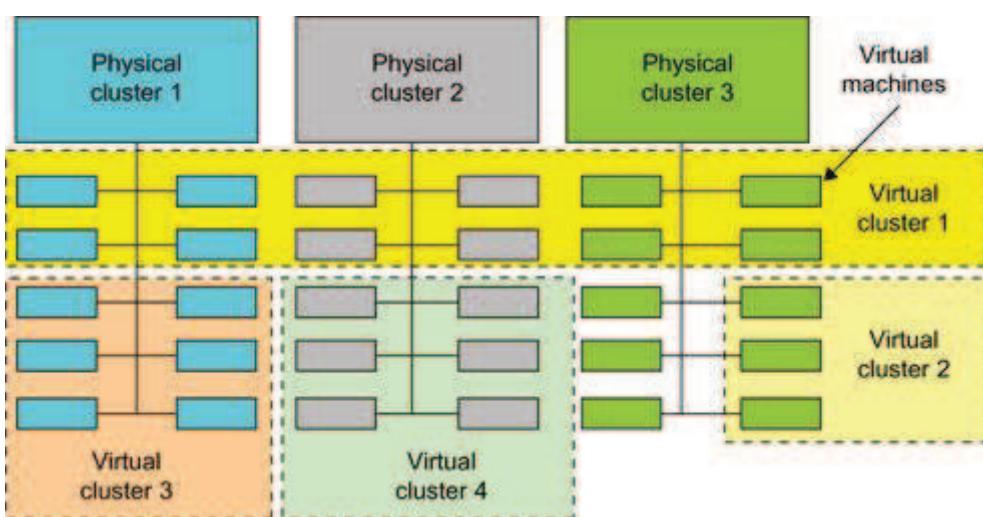
## 3.4 Virtual Clusters and Resource Management

A *physical cluster* is a collection of servers (physical machines) interconnected by a physical network such as a LAN. In [Chapter 2](#), we studied various clustering techniques on physical machines. Here, we introduce virtual clusters and study its properties as well as explore their potential applications. In this section, we will study three critical design issues of virtual clusters: *live migration* of VMs, *memory and file migrations*, and *dynamic deployment* of virtual clusters.

When a traditional VM is initialized, the administrator needs to manually write configuration information or specify the configuration sources. When more VMs join a network, an inefficient configuration always causes problems with overloading or underutilization. Amazon's *Elastic Compute Cloud (EC2)* is a good example of a web service that provides elastic computing power in a cloud. EC2 permits customers to create VMs and to manage user accounts over the time of their use. Most virtualization platforms, including XenServer and VMware ESX Server, support a bridging mode which allows all domains to appear on the network as individual hosts. By using this mode, VMs can communicate with one another freely through the virtual network interface card and configure the network automatically.

### 3.4.1 Physical versus Virtual Clusters

Virtual clusters are built with VMs installed at distributed servers from one or more physical clusters. The VMs in a virtual cluster are interconnected logically by a virtual network across several physical networks. [Figure 3.18](#) illustrates the concepts of virtual clusters and physical clusters. Each virtual cluster is formed with physical machines or a VM hosted by multiple physical clusters. The virtual cluster boundaries are shown as distinct boundaries.



**FIGURE 3.18** A cloud platform with four virtual clusters over three physical clusters shaded differently. Courtesy of Fan Zhang, Tsinghua University

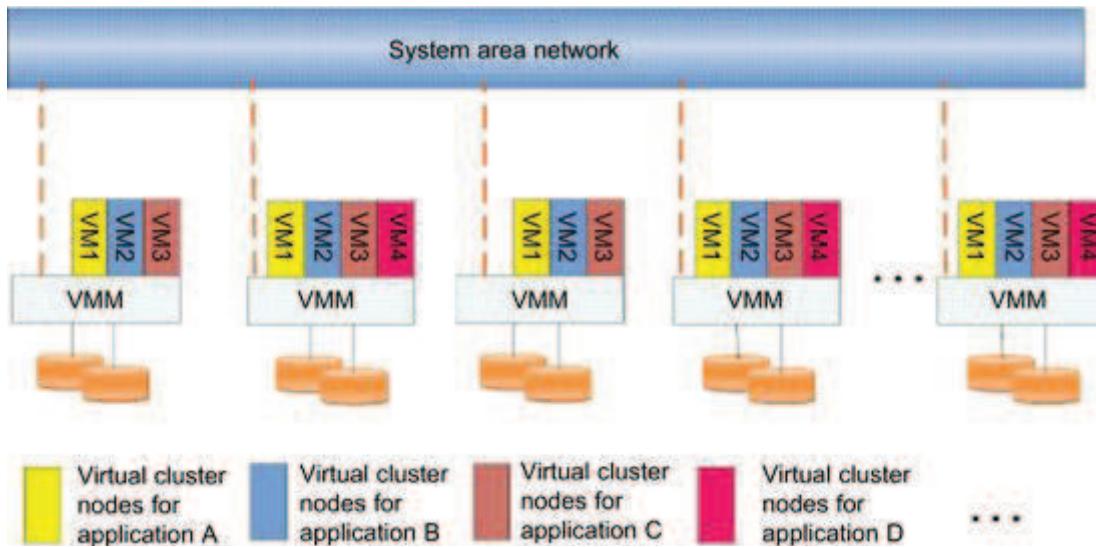
The provisioning of VMs to a virtual cluster is done dynamically to have the following interesting properties:

- The virtual cluster nodes can be either physical or virtual machines. Multiple VMs running with different OSes can be deployed on the same physical node.
- A VM runs with a guest OS, which is often different from the host OS, that manages the resources in the physical machine, where the VM is implemented.
- The purpose of using VMs is to consolidate multiple functionalities on the same server. This will greatly enhance server utilization and application flexibility.
- VMs can be colonized (replicated) in multiple servers for the purpose of promoting distributed parallelism, fault tolerance, and disaster recovery.
- The size (number of nodes) of a virtual cluster can grow or shrink dynamically, similar to the way an overlay network varies in size in a peer-to-peer (P2P) network.
- The failure of any physical nodes may disable some VMs installed on the failing nodes. But the failure of VMs will not pull down the host system.

Since system virtualization has been widely used, it is necessary to effectively manage VMs running on a mass of physical computing nodes (also called virtual clusters) and consequently build a high-performance virtualized computing environment. This involves virtual cluster deployment, monitoring and management over large-scale clusters, as well as resource scheduling, load balancing, server consolidation, fault tolerance, and other techniques. The different node colors in Figure 3.18 refer to different virtual clusters. In a virtual cluster system, it is quite important to store the large number of VM images efficiently.

Figure 3.19 shows the concept of a virtual cluster based on application partitioning or customization. The different colors in the figure represent the nodes in different virtual clusters. As a large number of VM images might be present, the most important thing is to determine how to store those images in the system efficiently. There are common installations for most users or applications, such as operating systems or user-level programming libraries. These software packages can be preinstalled as templates (called template VMs). With these templates, users can build their own software stacks. New OS

instances can be copied from the template VM. User-specific components such as programming libraries and applications can be installed to those instances.



**FIGURE 3.19** The concept of a virtual cluster based on application partitioning. Courtesy of Kang Chen, Tsinghua University 2008

Three physical clusters are shown on the left side of Figure 3.18. Four virtual clusters are created on the right, over the physical clusters. The physical machines are also called *host systems*. In contrast, the VMs are *guest systems*. The host and guest systems may run with different operating systems. Each VM can be installed on a remote server or replicated on multiple servers belonging to the same or different physical clusters. The boundary of a virtual cluster can change as VM nodes are added, removed, or migrated dynamically over time.

### 3.4.1.1 Fast Deployment and Effective Scheduling

The system should have the capability of fast deployment. Here, deployment means two things: to construct and distribute software stacks (OS, libraries, applications) to a physical node inside clusters as fast as possible, and to quickly switch runtime environments from one user's virtual cluster to another user's virtual cluster. If one user finishes using his system, the corresponding virtual cluster should shut down or suspend quickly to save the resources to run other VMs for other users.

The concept of "green computing" has attracted much attention recently. However, previous approaches have focused on saving the energy cost of components in a single workstation without a global vision. Consequently, they do not necessarily reduce the power consumption of the whole cluster. Other cluster-wide energy-efficient techniques can only be applied to homogeneous workstations and specific applications. The live migration of VMs allows workloads of one node to transfer to another node. However, it does not guarantee that VMs can randomly migrate among themselves. In fact, the potential overhead caused by live migrations of VMs cannot be ignored.

The overhead may have serious negative effects on cluster utilization, throughput, and

QoS issues. Therefore, the challenge is to determine how to design migration strategies to implement green computing without influencing the performance of clusters. Another advantage of virtualization is load balancing of applications in a virtual cluster. Load balancing can be achieved using the load index and frequency of user logins. The automatic scale-up and scale-down mechanism of a virtual cluster can be implemented based on this model. Consequently, we can increase the resource utilization of nodes and shorten the response time of systems. Mapping VMs onto the most appropriate physical node should promote performance. Dynamically adjusting loads among nodes by live migration of VMs is desired, when the loads on cluster nodes become quite unbalanced.

### **3.4.1.2 High-Performance Virtual Storage**

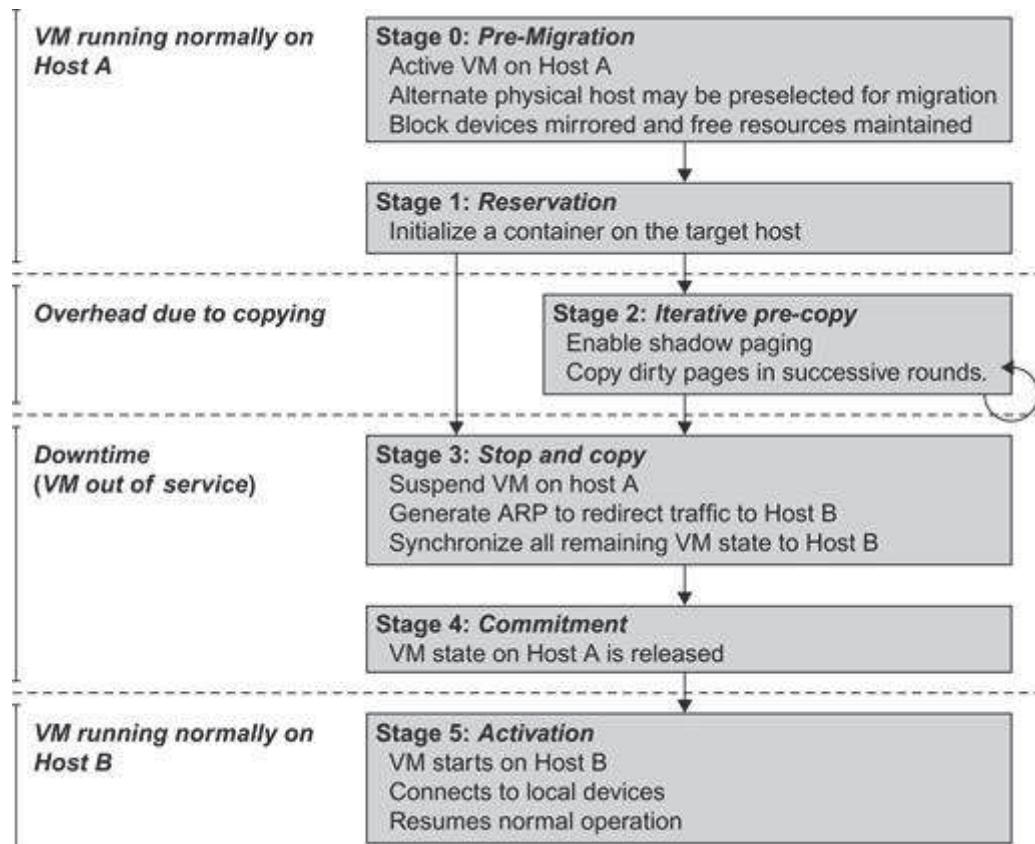
The template VM can be distributed to several physical hosts in the cluster to customize the VMs. In addition, existing software packages reduce the time for customization as well as switching virtual environments. It is important to efficiently manage the disk spaces occupied by template software packages. Some storage architecture design can be applied to reduce duplicated blocks in a distributed file system of virtual clusters. Hash values are used to compare the contents of data blocks. Users have their own profiles which store the identification of the data blocks for corresponding VMs in a user-specific virtual cluster. New blocks are created when users modify the corresponding data. Newly created blocks are identified in the users' profiles.

Basically, there are four steps to deploy a group of VMs onto a target cluster: *preparing the disk image, configuring the VMs, choosing the destination nodes, and executing the VM deployment command* on every host. Many systems use templates to simplify the disk image preparation process. A template is a disk image that includes a preinstalled operating system with or without certain application software. Users choose a proper template according to their requirements and make a duplicate of it as their own disk image. Templates could implement the COW (*Copy on Write*) format. A new COW backup file is very small and easy to create and transfer. Therefore, it definitely reduces disk space consumption. In addition, VM deployment time is much shorter than that of copying the whole raw image file.

Every VM is configured with a name, disk image, network setting, and allocated CPU and memory. One needs to record each VM configuration into a file. However, this method is inefficient when managing a large group of VMs. VMs with the same configurations could use preedited profiles to simplify the process. In this scenario, the system configures the VMs according to the chosen profile. Most configuration items use the same settings, while some of them, such as UUID, VM name, and IP address, are assigned with automatically calculated values. Normally, users do not care which host is running their VM. A strategy to choose the proper destination host for any VM is needed. The deployment principle is to fulfill the VM requirement and to balance workloads among the whole host network.

### **3.4.2 Live VM Migration Steps and Performance Effects**

In a cluster built with mixed nodes of host and guest systems, the normal method of operation is to run everything on the physical machine. When a VM fails, its role could be replaced by another VM on a different node, as long as they both run with the same guest OS. In other words, a physical node can fail over to a VM on another host. This is different from physical-to-physical failover in a traditional physical cluster. The advantage is enhanced failover flexibility. The potential drawback is that a VM must stop playing its role if its residing host node fails. However, this problem can be mitigated with VM life migration. [Figure 3.20](#) shows the process of life migration of a VM from host A to host B. The migration copies the VM state file from the storage area to the host machine.



**FIGURE 3.20** Live migration process of a VM from one host to another. Courtesy of C. Clark, et al. [\[14\]](#)

There are four ways to manage a virtual cluster. First, you can use a *guest-based manager*, by which the cluster manager resides on a guest system. In this case, multiple VMs form a virtual cluster. For example, openMosix is an open source Linux cluster running different guest systems on top of the Xen hypervisor. Another example is Sun's cluster Oasis, an experimental Solaris cluster of VMs supported by a VMware VMM. Second, you can build a cluster manager on the host systems. The *host-based manager* supervises the guest systems and can restart the guest system on another physical machine. A good example is the VMware HA system that can restart a guest system after failure.

These two cluster management systems are either guest-only or host-only, but they do not mix. A third way to manage a virtual cluster is to use an *independent cluster manager*.

on both the host and guest systems. This will make infrastructure management more complex, however. Finally, you can use an *integrated cluster* on the guest and host systems. This means the manager must be designed to distinguish between virtualized resources and physical resources. Various cluster management schemes can be greatly enhanced when VM life migration is enabled with minimal overhead.

VMs can be live-migrated from one physical machine to another; in case of failure, one VM can be replaced by another VM. Virtual clusters can be applied in computational grids, cloud platforms, and high-performance computing (HPC) systems. The major attraction of this scenario is that virtual clustering provides dynamic resources that can be quickly put together upon user demand or after a node failure. In particular, virtual clustering plays a key role in cloud computing. When a VM runs a live service, it is necessary to make a trade-off to ensure that the migration occurs in a manner that minimizes all three metrics. The motivation is to design a live VM migration scheme with negligible downtime, the lowest network bandwidth consumption possible, and a reasonable total migration time.

Furthermore, we should ensure that the migration will not disrupt other active services residing in the same host through resource contention (e.g., CPU, network bandwidth). A VM can be in one of the following four states. An *inactive state* is defined by the virtualization platform, under which the VM is not enabled. An *active state* refers to a VM that has been instantiated at the virtualization platform to perform a real task. A *paused state* corresponds to a VM that has been instantiated but disabled to process a task or paused in a waiting state. A VM enters the *suspended state* if its machine file and virtual resources are stored back to the disk. As shown in [Figure 3.20](#), live migration of a VM consists of the following six steps:

**Steps 0 and 1: Start migration.** This step makes preparations for the migration, including determining the migrating VM and the destination host. Although users could manually make a VM migrate to an appointed host, in most circumstances, the migration is automatically started by strategies such as load balancing and server consolidation.

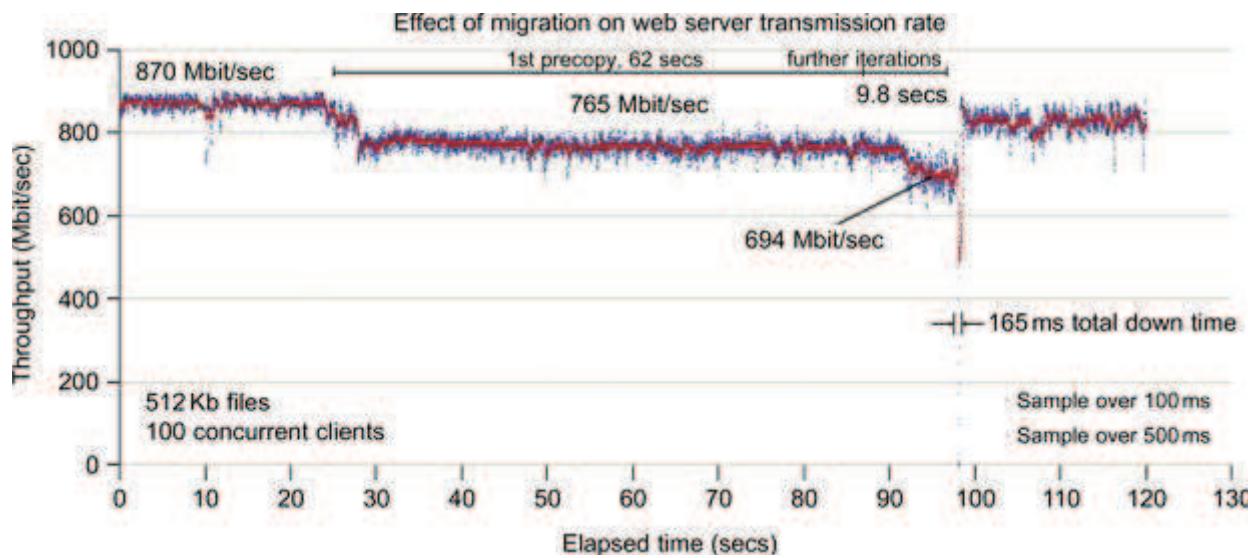
**Steps 2: Transfer memory.** Since the whole execution state of the VM is stored in memory, sending the VM's memory to the destination node ensures continuity of the service provided by the VM. All of the memory data is transferred in the first round, and then the migration controller recopies the memory data which is changed in the last round. These steps keep iterating until the dirty portion of the memory is small enough to handle the final copy. Although precopying memory is performed iteratively, the execution of programs is not obviously interrupted.

**Step 3: Suspend the VM and copy the last portion of the data.** The migrating VM's execution is suspended when the last round's memory data is transferred. Other nonmemory data such as CPU and network states should be sent as well. During this step, the VM is stopped and its applications will no longer run. This "service unavailable" time is called the "downtime" of migration, which should be as short as possible so that it can be negligible to users.

**Steps 4 and 5: Commit and activate the new host.** After all the needed data is copied, on

the destination host, the VM reloads the states and recovers the execution of programs in it, and the service provided by this VM continues. Then the network connection is redirected to the new VM and the dependency to the source host is cleared. The whole migration process finishes by removing the original VM from the source host.

Figure 3.21 shows the effect on the data transmission rate (Mbit/second) of live migration of a VM from one host to another. Before copying the VM with 512 KB files for 100 clients, the data throughput was 870 MB/second. The first precopy takes 63 seconds, during which the rate is reduced to 765 MB/second. Then the data rate reduces to 694 MB/second in 9.8 seconds for more iterations of the copying process. The system experiences only 165 ms of downtime, before the VM is restored at the destination host. This experimental result shows a very small migration overhead in live transfer of a VM between host nodes. This is critical to achieve dynamic cluster reconfiguration and disaster recovery as needed in cloud computing. We will study these techniques in more detail in Chapter 4.



**FIGURE 3.21** Effect on data transmission rate of a VM migrated from one failing web server to another. Courtesy of C. Clark, et al. [14]

With the emergence of widespread cluster computing more than a decade ago, many cluster configuration and management systems have been developed to achieve a range of goals. These goals naturally influence individual approaches to cluster management. VM technology has become a popular method for simplifying management and sharing of physical computing resources. Platforms such as VMware and Xen allow multiple VMs with different operating systems and configurations to coexist on the same physical host in mutual isolation. Clustering inexpensive computers is an effective way to obtain reliable, scalable computing power for network services and compute-intensive applications.

### 3.4.3 Migration of Memory, Files, and Network Resources

Since clusters have a high initial cost of ownership, including space, power conditioning, and cooling equipment, leasing or sharing access to a common cluster is an attractive solution when demands vary over time. Shared clusters offer economies of scale and more effective utilization of resources by multiplexing. Early configuration and management systems focus on expressive and scalable mechanisms for defining clusters for specific types of service, and physically partition cluster nodes among those types. When one system migrates to another physical node, we should consider the following issues.

### **3.4.3.1 Memory Migration**

This is one of the most important aspects of VM migration. Moving the memory instance of a VM from one physical host to another can be approached in any number of ways. But traditionally, the concepts behind the techniques tend to share common implementation paradigms. The techniques employed for this purpose depend upon the characteristics of application/workloads supported by the guest OS.

Memory migration can be in a range of hundreds of megabytes to a few gigabytes in a typical system today, and it needs to be done in an efficient manner. The *Internet Suspend-Resume (ISR)* technique exploits temporal locality as memory states are likely to have considerable overlap in the suspended and the resumed instances of a VM. Temporal locality refers to the fact that the memory states differ only by the amount of work done since a VM was last suspended before being initiated for migration.

To exploit temporal locality, each file in the file system is represented as a tree of small subfiles. A copy of this tree exists in both the suspended and resumed VM instances. The advantage of using a tree-based representation of files is that the caching ensures the transmission of only those files which have been changed. The ISR technique deals with situations where the migration of live machines is not a necessity. Predictably, the downtime (the period during which the service is unavailable due to there being no currently executing instance of a VM) is high, compared to some of the other techniques discussed later.

### **3.4.3.2 File System Migration**

To support VM migration, a system must provide each VM with a consistent, location-independent view of the file system that is available on all hosts. A simple way to achieve this is to provide each VM with its own virtual disk which the file system is mapped to and transport the contents of this virtual disk along with the other states of the VM. However, due to the current trend of high-capacity disks, migration of the contents of an entire disk over a network is not a viable solution. Another way is to have a global file system across all machines where a VM could be located. This way removes the need to copy files from one machine to another because all files are network-accessible.

A distributed file system is used in ISR serving as a transport mechanism for propagating a suspended VM state. The actual file systems themselves are not mapped onto the distributed file system. Instead, the VMM only accesses its local file system. The

relevant VM files are explicitly copied into the local file system for a resume operation and taken out of the local file system for a suspend operation. This approach relieves developers from the complexities of implementing several different file system calls for different distributed file systems. It also essentially disassociates the VMM from any particular distributed file system semantics. However, this decoupling means that the VMM has to store the contents of each VM's virtual disks in its local files, which have to be moved around with the other state information of that VM.

In smart copying, the VMM exploits spatial locality. Typically, people often move between the same small number of locations, such as their home and office. In these conditions, it is possible to transmit only the difference between the two file systems at suspending and resuming locations. This technique significantly reduces the amount of actual physical data that has to be moved. In situations where there is no locality to exploit, a different approach is to synthesize much of the state at the resuming site. On many systems, user files only form a small fraction of the actual data on disk. Operating system and application software account for the majority of storage space. The proactive state transfer solution works in those cases where the resuming site can be predicted with reasonable confidence.

### **3.4.3.3 Network Migration**

A migrating VM should maintain all open network connections without relying on forwarding mechanisms on the original host or on support from mobility or redirection mechanisms. To enable remote systems to locate and communicate with a VM, each VM must be assigned a virtual IP address known to other entities. This address can be distinct from the IP address of the host machine where the VM is currently located. Each VM can also have its own distinct virtual MAC address. The VMM maintains a mapping of the virtual IP and MAC addresses to their corresponding VMs. In general, a migrating VM includes all the protocol states and carries its IP address with it.

If the source and destination machines of a VM migration are typically connected to a single switched LAN, an unsolicited ARP reply from the migrating host is provided advertising that the IP has moved to a new location. This solves the open network connection problem by reconfiguring all the peers to send future packets to a new location. Although a few packets that have already been transmitted might be lost, there are no other problems with this mechanism. Alternatively, on a switched network, the migrating OS can keep its original Ethernet MAC address and rely on the network switch to detect its move to a new port.

Live migration means moving a VM from one physical node to another while keeping its OS environment and applications unbroken. This capability is being increasingly utilized in today's enterprise environments to provide efficient online system maintenance, reconfiguration, load balancing, and proactive fault tolerance. It provides desirable features to satisfy requirements for computing resources in modern computing systems, including server consolidation, performance isolation, and ease of management. As a result, many implementations are available which support the feature using disparate functionalities. Traditional migration suspends VMs before the transportation

and then resumes them at the end of the process. By importing the precopy mechanism, a VM could be live-migrated without stopping the VM and keep the applications running during the migration.

Live migration is a key feature of system virtualization technologies. Here, we focus on VM migration within a cluster environment where a network-accessible storage system, such as *storage area network* (SAN) or *network attached storage* (NAS), is employed. Only memory and CPU status needs to be transferred from the source node to the target node. Live migration techniques mainly use the precopy approach, which first transfers all memory pages, and then only copies modified pages during the last round iteratively. The VM service downtime is expected to be minimal by using iterative copy operations. When applications' writable working set becomes small, the VM is suspended and only the CPU state and dirty pages in the last round are sent out to the destination.

In the precopy phase, although a VM service is still available, much performance degradation will occur because the migration daemon continually consumes network bandwidth to transfer dirty pages in each round. An adaptive rate limiting approach is employed to mitigate this issue, but total migration time is prolonged by nearly 10 times. Moreover, the maximum number of iterations must be set because not all applications' dirty pages are ensured to converge to a small writable working set over multiple rounds.

In fact, these issues with the precopy approach are caused by the large amount of transferred data during the whole migration process. A checkpointing/recovery and trace/replay approach (CR/TR-Motion) is proposed to provide fast VM migration. This approach transfers the execution trace file in iterations rather than dirty pages, which is logged by a trace daemon. Apparently, the total size of all log files is much less than that of dirty pages. So, total migration time and downtime of migration are drastically reduced. However, CR/TR-Motion is valid only when the log replay rate is larger than the log growth rate. The inequality between source and target nodes limits the application scope of live migration in clusters.

Another strategy of postcopy is introduced for live migration of VMs. Here, all memory pages are transferred only once during the whole migration process and the baseline total migration time is reduced. But the downtime is much higher than that of precopy due to the latency of fetching pages from the source node before the VM can be resumed on the target. With the advent of multicore or many-core machines, abundant CPU resources are available. Even if several VMs reside on a same multicore machine, CPU resources are still rich because physical CPUs are frequently amenable to multiplexing. We can exploit these copious CPU resources to compress page frames and the amount of transferred data can be significantly reduced. Memory compression algorithms typically have little memory overhead. Decompression is simple and very fast and requires no memory for decompression.

#### **3.4.3.4 Live Migration of VM Using Xen**

In [Section 3.2.1](#), we studied Xen as a VMM or hypervisor, which allows multiple commodity OSes to share x86 hardware in a safe and orderly fashion. The following example explains how to perform live migration of a VM between two Xen-enabled host

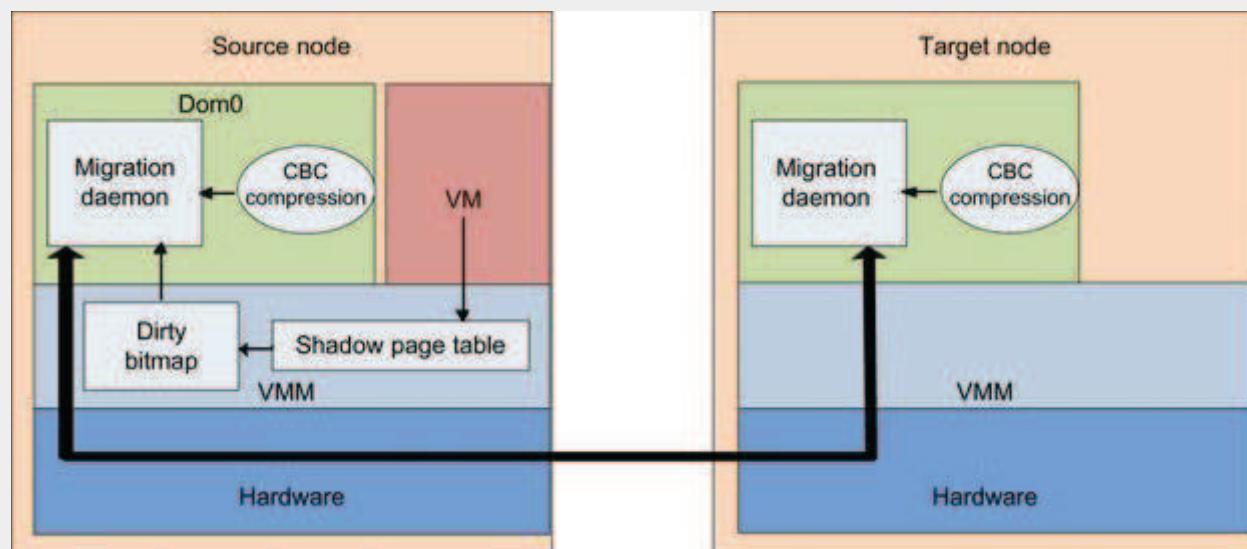
machines. Domain 0 (or Dom0) performs tasks to create, terminate, or migrate to another host. Xen uses a send/recv model to transfer states across VMs.

### Example 3.8

## Live Migration of VMs between Two Xen-Enabled Hosts

Xen supports live migration. It is a useful feature and natural extension to virtualization platforms that allows for the transfer of a VM from one physical machine to another with little or no downtime of the services hosted by the VM. Live migration transfers the working state and memory of a VM across a network when it is running. Xen also supports VM migration by using a mechanism called *Remote Direct Memory Access (RDMA)*.

RDMA speeds up VM migration by avoiding TCP/IP stack processing overhead. RDMA implements a different transfer protocol whose origin and destination VM buffers must be registered before any transfer operations occur, reducing it to a “one-sided” interface. Data communication over RDMA does not need to involve the CPU, caches, or context switches. This allows migration to be carried out with minimal impact on guest operating systems and hosted applications. Figure 3.22 shows the a compression scheme for VM migration.



**FIGURE 3.22** Live migration of VM from the Dom0 domain to a Xen-enabled target host.

This design requires that we make trade-offs between two factors. If an algorithm embodies expectations about the kinds of regularities in the memory footprint, it must be very fast and effective. A single compression algorithm for all memory data is difficult to achieve the win-win status that

we expect. Therefore, it is necessary to provide compression algorithms to pages with different kinds of regularities. The structure of this live migration system is presented in Dom0.

Migration daemons running in the management VMs are responsible for performing migration. Shadow page tables in the VMM layer trace modifications to the memory page in migrated VMs during the precopy phase. Corresponding flags are set in a dirty bitmap. At the start of each precopy round, the bitmap is sent to the migration daemon. Then, the bitmap is cleared and the shadow page tables are destroyed and re-created in the next round. The system resides in Xen's management VM. Memory pages denoted by bitmap are extracted and compressed before they are sent to the destination. The compressed data is then decompressed on the target.

### 3.4.4 Dynamic Deployment of Virtual Clusters

Table 3.5 summarizes four virtual cluster research projects. We briefly introduce them here just to identify their design objectives and reported results. The Cellular Disco at Stanford is a virtual cluster built in a shared-memory multiprocessor system. The INRIA virtual cluster was built to test parallel algorithm performance. The COD and VIOLIN clusters are studied in forthcoming examples.

**Table 3.5**

**Experimental Results on Four Research Virtual Clusters**

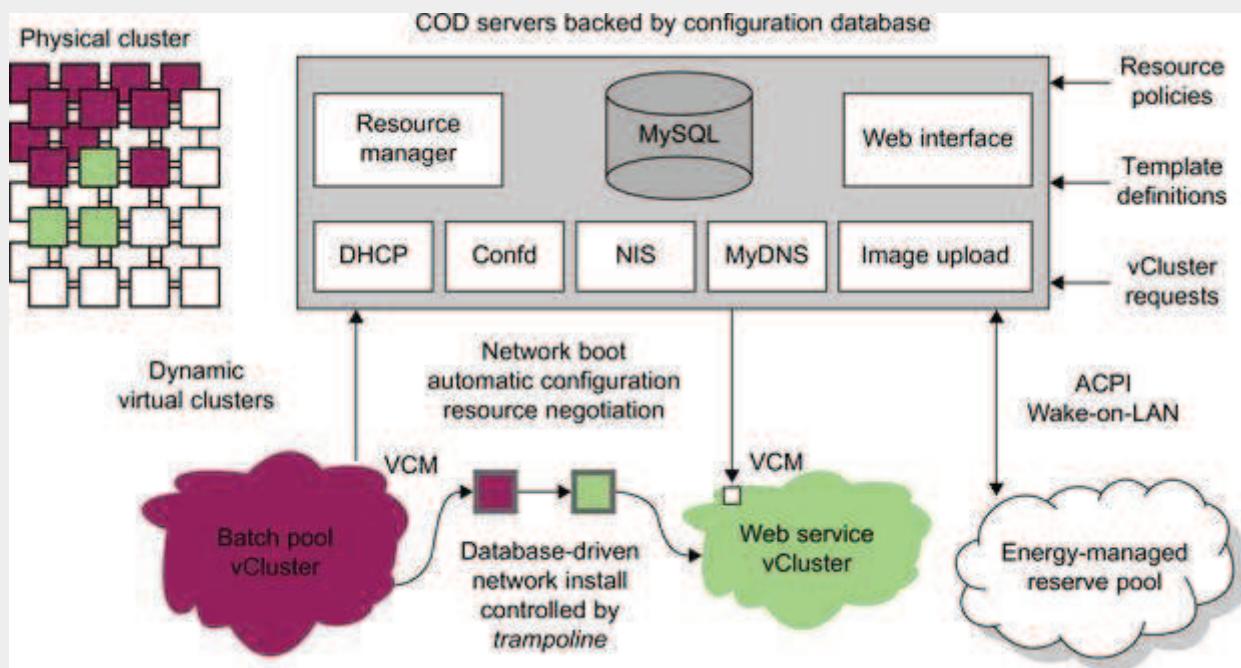
Project Name	Design Objectives	Reported Results and References
Cluster-on-Demand at Duke Univ.	Dynamic resource allocation with a virtual cluster management system	Sharing of VMs by multiple virtual clusters using Sun GridEngine [12]
Cellular Disco at Stanford Univ.	To deploy a virtual cluster on a shared-memory multiprocessor	VMs deployed on multiple processors under a VMM called Cellular Disco [8]
VIOLIN at Purdue Univ.	Multiple VM clustering to prove the advantage of dynamic adaptation	Reduce execution time of applications running VIOLIN with adaptation [25,55]
GRAAL Project at INRIA in France	Performance of parallel algorithms in Xen-enabled virtual clusters	75% of max. performance achieved with 30% resource slacks over VM clusters

#### Example 3.9

#### The Cluster-on-Demand (COD) Project at Duke University

Developed by researchers at Duke University, the COD (*Cluster-on-Demand*)

project is a virtual cluster management system for dynamic allocation of servers from a computing pool to multiple virtual clusters [12]. The idea is illustrated by the prototype implementation of the COD shown in Figure 3.23. The COD partitions a physical cluster into multiple virtual clusters (*vClusters*). vCluster owners specify the operating systems and software for their clusters through an XML-RPC interface. The *vClusters* run a batch schedule from Sun's GridEngine on a web server cluster. The COD system can respond to load changes in restructuring the virtual clusters dynamically.

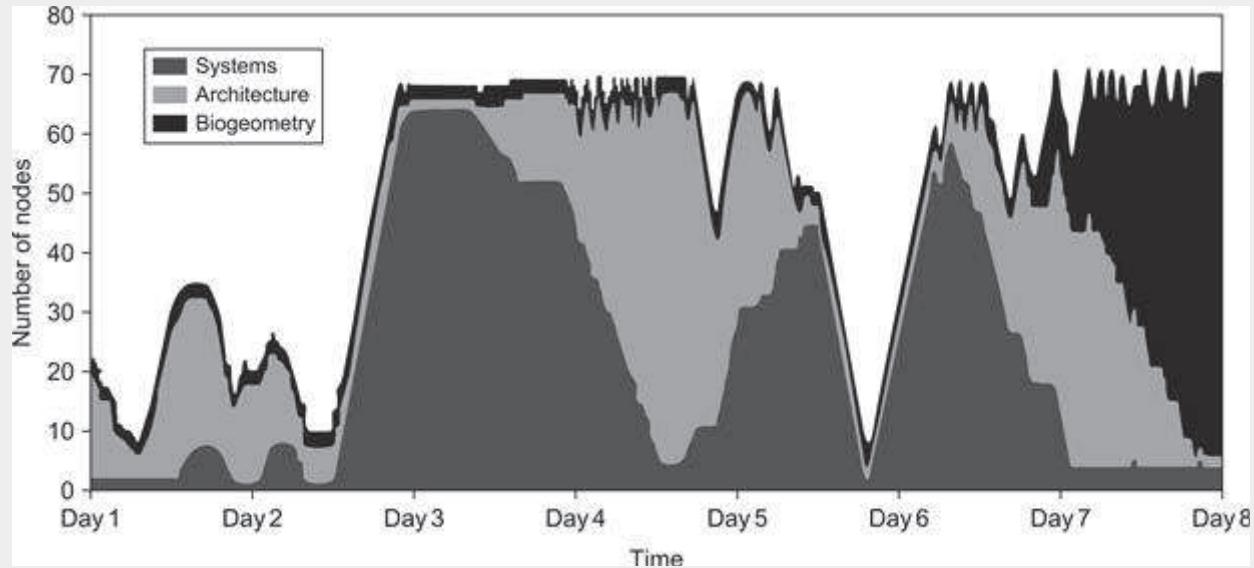


**FIGURE 3.23** COD partitioning a physical cluster into multiple virtual clusters. Courtesy of Jeff Chase, et al., HPDC-2003 © IEEE [12]

The Duke researchers used the Sun GridEngine scheduler to demonstrate that dynamic virtual clusters are an enabling abstraction for advanced resource management in computing utilities such as grids. The system supports dynamic, policy-based cluster sharing between local users and hosted grid services. Attractive features include resource reservation, adaptive provisioning, scavenging of idle resources, and dynamic instantiation of grid services. The COD servers are backed by a configuration database. This system provides resource policies and template definition in response to user requests.

Figure 3.24 shows the variation in the number of nodes in each of three virtual clusters during eight days of a live deployment. Three application workloads requested by three user groups are labeled "Systems," "Architecture," and "BioGeometry" in the trace plot. The experiments were performed with multiple SGE batch pools on a test bed of 80 rack-mounted IBM xSeries-335 servers within the Duke cluster. This trace plot clearly shows the sharp variation in cluster size (number of nodes) over the eight days.

Dynamic provisioning and deprovisioning of virtual clusters are needed in real-life cluster applications.

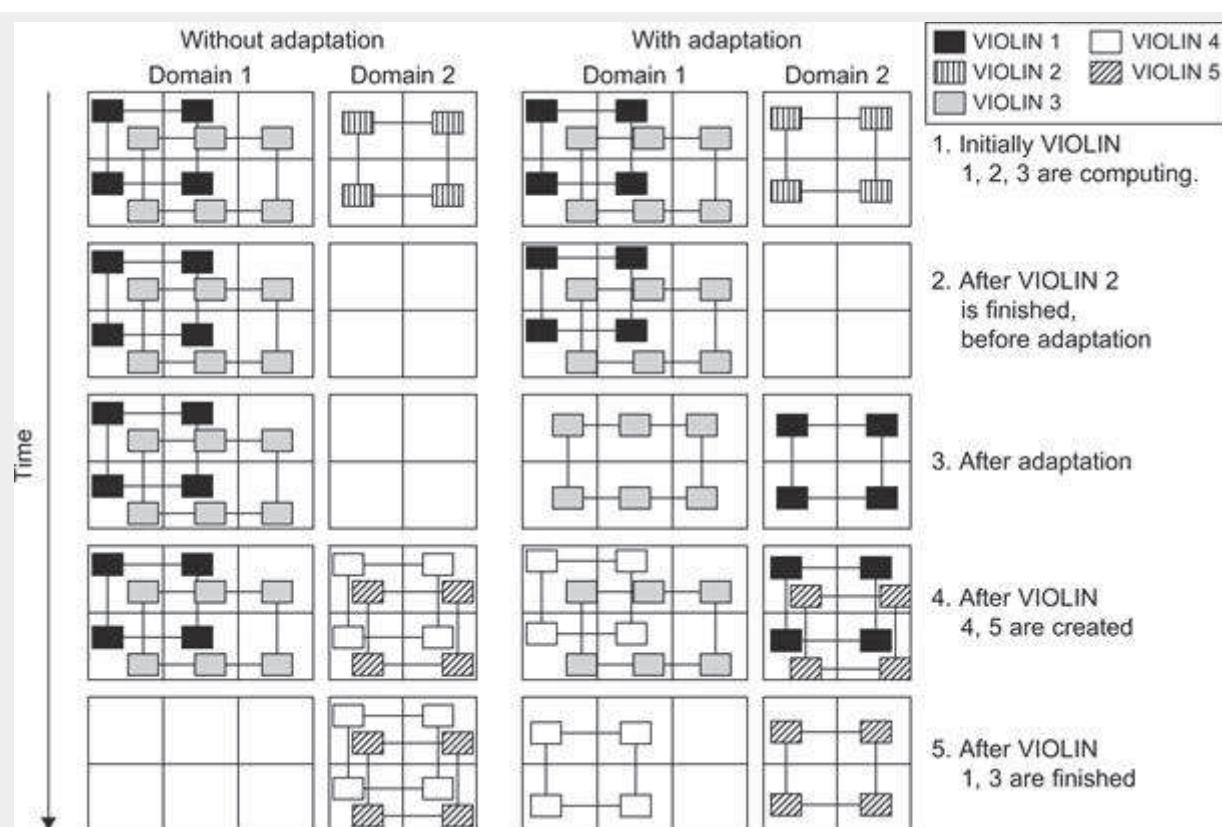


**FIGURE 3.24** Cluster size variations in COD over eight days at Duke University. Courtesy of Jeff Chase, et al., HPDC-2003 © IEEE [12]

### Example 3.10

#### The VIOLIN Project at Purdue University

The Purdue VIOLIN Project applies live VM migration to reconfigure a virtual cluster environment. Its purpose is to achieve better resource utilization in executing multiple cluster jobs on multiple cluster domains. The project leverages the maturity of VM migration and environment adaptation technology. The approach is to enable mutually isolated virtual environments for executing parallel applications on top of a shared physical infrastructure consisting of multiple domains. Figure 3.25 illustrates the idea with five concurrent virtual environments, labeled as VIOLIN 1–5, sharing two physical clusters.



**FIGURE 3.25** VIOLIN adaptation scenario of five virtual environments sharing two hosted clusters. Note that there are more idle squares (blank nodes) before and after the adaptation. Courtesy of P. Ruth, et al. [55]

The squares of various shadings represent the VMs deployed in the physical server nodes. The major contribution by the Purdue group is to achieve autonomic adaptation of the virtual computation environments as active, integrated entities. A virtual execution environment is able to relocate itself across the infrastructure, and can scale its share of infrastructural resources. The adaptation is transparent to both users of virtual environments and administrations of infrastructures. The adaptation overhead is maintained at 20 sec out of 1,200 sec in solving a large NEMO3D problem of 1 million particles.

The message being conveyed here is that the virtual environment adaptation can enhance resource utilization significantly at the expense of less than 1 percent of an increase in total execution time. The migration of VIOLIN environments does pay off. Of course, the gain in shared resource utilization will benefit many users, and the performance gain varies with different adaptation scenarios. We leave readers to trace the execution of another scenario in [Problem 3.17](#) at the end of this chapter to tell the differences. Virtual networking is a fundamental component of the VIOLIN system.

Data centers have grown rapidly in recent years, and all major IT companies are pouring their resources into building new data centers. In addition, Google, Yahoo!, Amazon, Microsoft, HP, Apple, and IBM are all in the game. All these companies have invested billions of dollars in data-center construction and automation. Data-center automation means that huge volumes of hardware, software, and database resources in these data centers can be allocated dynamically to millions of Internet users simultaneously, with guaranteed QoS and cost-effectiveness.

This automation process is triggered by the growth of virtualization products and cloud computing services. From 2006 to 2011, according to an IDC 2007 report on the growth of virtualization and its market distribution in major IT sectors. In 2006, virtualization has a market share of \$1,044 million in business and enterprise opportunities. The majority was dominated by production consolidation and software development. Virtualization is moving towards enhancing mobility, reducing planned downtime (for maintenance), and increasing the number of virtual clients.

The latest virtualization development highlights *high availability* (HA), backup services, workload balancing, and further increases in client bases. IDC projected that automation, service orientation, policy-based, and variable costs in the virtualization market. The total business opportunities may increase to \$3.2 billion by 2011. The major market share moves to the areas of HA, utility computing, production consolidation, and client bases. In what follows, we will discuss server consolidation, virtual storage, OS support, and trust management in automated data-center designs.

### 3.5.1 Server Consolidation in Data Centers

In data centers, a large number of heterogeneous workloads can run on servers at various times. These heterogeneous workloads can be roughly divided into two categories: chatty workloads and noninteractive workloads. Chatty workloads may burst at some point and return to a silent state at some other point. A web video service is an example of this, whereby a lot of people use it at night and few people use it during the day. Noninteractive workloads do not require people's efforts to make progress after they are submitted. High-performance computing is a typical example of this. At various stages, the requirements for resources of these workloads are dramatically different. However, to guarantee that a workload will always be able to cope with all demand levels, the workload is statically allocated enough resources so that peak demand is satisfied. [Figure 3.29](#) illustrates server virtualization in a data center. In this case, the granularity of resource optimization is focused on the CPU, memory, and network interfaces.

Therefore, it is common that most servers in data centers are underutilized. A large amount of hardware, space, power, and management cost of these servers is wasted. Server consolidation is an approach to improve the low utility ratio of hardware resources by reducing the number of physical servers. Among several server consolidation techniques such as centralized and physical consolidation, virtualization-based server consolidation is the most powerful. Data centers need to optimize their resource management. Yet these techniques are performed with the granularity of a full server

machine, which makes resource management far from well optimized. Server virtualization enables smaller resource allocation than a physical machine.

In general, the use of VMs increases resource management complexity. This causes a challenge in terms of how to improve resource utilization as well as guarantee QoS in data centers. In detail, server virtualization has the following side effects:

- Consolidation enhances hardware utilization. Many underutilized servers are consolidated into fewer servers to enhance resource utilization. Consolidation also facilitates backup services and disaster recovery.
- This approach enables more agile provisioning and deployment of resources. In a virtual environment, the images of the guest OSes and their applications are readily cloned and reused.
- The total cost of ownership is reduced. In this sense, server virtualization causes deferred purchases of new servers, a smaller data-center footprint, lower maintenance costs, and lower power, cooling, and cabling requirements.
- This approach improves availability and business continuity. The crash of a guest OS has no effect on the host OS or any other guest OS. It becomes easier to transfer a VM from one server to another, because virtual servers are unaware of the underlying hardware.

To automate data-center operations, one must consider resource scheduling, architectural support, power management, automatic or autonomic resource management, performance of analytical models, and so on. In virtualized data centers, an efficient, on-demand, fine-grained scheduler is one of the key factors to improve resource utilization. Scheduling and reallocations can be done in a wide range of levels in a set of data centers. The levels match at least at the VM level, server level, and data-center level. Ideally, scheduling and resource reallocations should be done at all levels. However, due to the complexity of this, current techniques only focus on a single level or, at most, two levels.

Dynamic CPU allocation is based on VM utilization and application-level QoS metrics. One method considers both CPU and memory flowing as well as automatically adjusting resource overhead based on varying workloads in hosted services. Another scheme uses a two-level resource management system to handle the complexity involved. A local controller at the VM level and a global controller at the server level are designed. They implement autonomic resource allocation via the interaction of the local and global controllers. Multicore and virtualization are two cutting techniques that can enhance each other.

However, the use of CMP is far from well optimized. The memory system of CMP is a typical example. One can design a virtual hierarchy on a CMP in data centers. One can consider protocols that minimize the memory access time, inter-VM interferences, facilitating VM reassignment, and supporting inter-VM sharing. One can also consider a VM-aware power budgeting scheme using multiple managers integrated to achieve better power management. The power budgeting policies cannot ignore the heterogeneity problems. Consequently, one must address the trade-off of power saving and data-center performance.

## 3.5.2 Virtual Storage Management

The term “storage virtualization” was widely used before the renaissance of system virtualization. Yet the term has a different meaning in a system virtualization environment. Previously, storage virtualization was largely used to describe the aggregation and repartitioning of disks at very coarse time scales for use by physical machines. In system virtualization, virtual storage includes the storage managed by VMMs and guest OSes. Generally, the data stored in this environment can be classified into two categories: VM images and application data. The VM images are special to the virtual environment, while application data includes all other data which is the same as the data in traditional OS environments.

The most important aspects of system virtualization are encapsulation and isolation. Traditional operating systems and applications running on them can be encapsulated in VMs. Only one operating system runs in a virtualization while many applications run in the operating system. System virtualization allows multiple VMs to run on a physical machine and the VMs are completely isolated. To achieve encapsulation and isolation, both the system software and the hardware platform, such as CPUs and chipsets, are rapidly updated. However, storage is lagging. The storage systems become the main bottleneck of VM deployment.

In virtualization environments, a virtualization layer is inserted between the hardware and traditional operating systems or a traditional operating system is modified to support virtualization. This procedure complicates storage operations. On the one hand, storage management of the guest OS performs as though it is operating in a real hard disk while the guest OSes cannot access the hard disk directly. On the other hand, many guest OSes contest the hard disk when many VMs are running on a single physical machine. Therefore, storage management of the underlying VMM is much more complex than that of guest OSes (traditional OSes).

In addition, the storage primitives used by VMs are not nimble. Hence, operations such as remapping volumes across hosts and checkpointing disks are frequently clumsy and esoteric, and sometimes simply unavailable. In data centers, there are often thousands of VMs, which cause the VM images to become flooded. Many researchers tried to solve these problems in virtual storage management. The main purposes of their research are to make management easy while enhancing performance and reducing the amount of storage occupied by the VM images. Parallax is a distributed storage system customized for virtualization environments. Content Addressable Storage (CAS) is a solution to reduce the total size of VM images, and therefore supports a large set of VM-based systems in data centers.

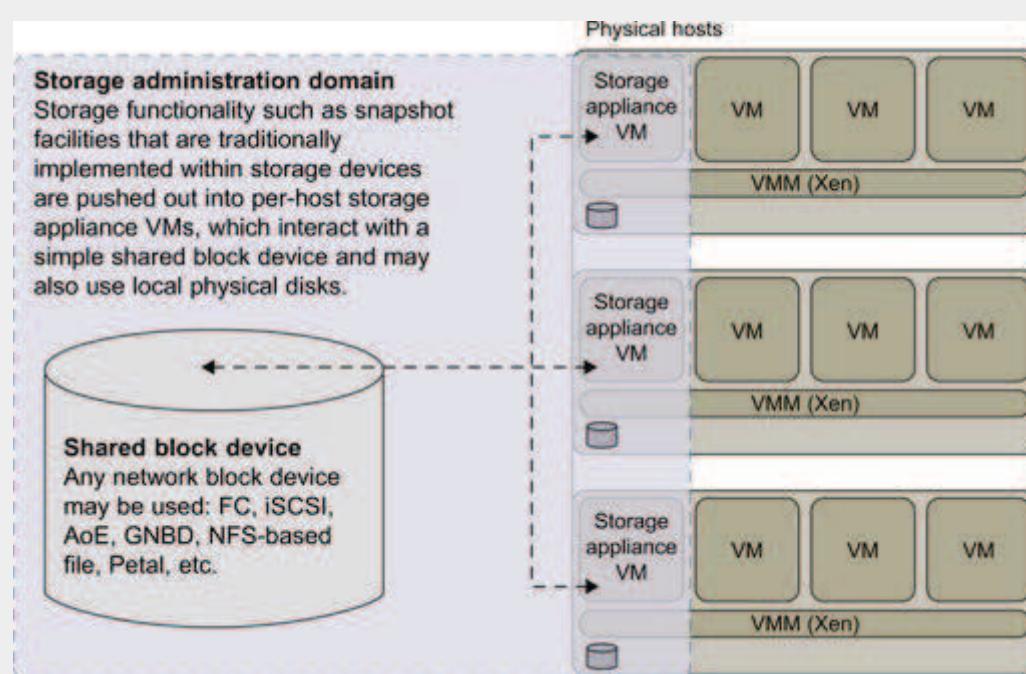
Since traditional storage management techniques do not consider the features of storage in virtualization environments, Parallax designs a novel architecture in which storage features that have traditionally been implemented directly on high-end storage arrays and switchers are relocated into a federation of storage VMs. These storage VMs share the same physical hosts as the VMs that they serve. [Figure 3.30](#) provides an overview of the Parallax system architecture. It supports all popular system virtualization

techniques, such as paravirtualization and full virtualization. For each physical machine, Parallax customizes a special storage appliance VM. The storage appliance VM acts as a block virtualization layer between individual VMs and the physical storage device. It provides a virtual disk for each VM on the same physical machine.

### Example 3.11

#### Parallax Providing Virtual Disks to Client VMs from a Large Common Shared Physical Disk

The architecture of Parallax is scalable and especially suitable for use in cluster-based environments. Figure 3.26 shows a high-level view of the structure of a Parallax-based cluster. A cluster-wide administrative domain manages all storage appliance VMs, which makes storage management easy. The storage appliance VM also allows functionality that is currently implemented within data-center hardware to be pushed out and implemented on individual hosts. This mechanism enables advanced storage features such as snapshot facilities to be implemented in software and delivered above commodity network storage targets.



**FIGURE 3.26** Parallax is a set of per-host storage appliances that share access to a common block device and presents virtual disks to client VMs. Courtesy of D. Meyer, et al. [43]

Parallax itself runs as a user-level application in the storage appliance VM. It provides *virtual disk images (VDIs)* to VMs. A VDI is a single-writer virtual disk which may be accessed in a location-transparent manner from any of the physical hosts in the Parallax cluster. The VDIs are the core abstraction

provided by Parallax. Parallax uses Xen's block tap driver to handle block requests and it is implemented as a tapdisk library. This library acts as a single block virtualization service for all client VMs on the same physical host. In the Parallax system, it is the storage appliance VM that connects the physical hardware device for block and network access. As shown in Figure 3.30, physical device drivers are included in the storage appliance VM. This implementation enables a storage administrator to live-upgrade the block device drivers in an active cluster.

### 3.5.3 Cloud OS for Virtualized Data Centers

Data centers must be virtualized to serve as cloud providers. Table 3.6 summarizes four *virtual infrastructure (VI)* managers and OSes. These VI managers and OSes are specially tailored for virtualizing data centers which often own a large number of servers in clusters. Nimbus, Eucalyptus, and OpenNebula are all open source software available to the general public. Only vSphere 4 is a proprietary OS for cloud resource virtualization and management over data centers.

**Table 3.6**

**VI Managers and Operating Systems for Virtualizing Data Centers [9]**

Manager/ OS, Platforms, License	Resources Being Virtualized, Web Link	Client API, Language	Hypervisors Used	Public Cloud Interface	Special Features
<b>Nimbus</b> Linux, Apache v2	VM creation, virtual cluster, <a href="http://www.nimbusproject.org/">www.nimbusproject.org/</a>	EC2 WS, WSRF, CLI	Xen, KVM	EC2	Virtual networks
<b>Eucalyptus</b> Linux, BSD	Virtual networking (Example 3.12 and [41]), <a href="http://www.eucalyptus.com/">www.eucalyptus.com/</a>	EC2 WS, CLI	Xen, KVM	EC2	Virtual networks
<b>OpenNebula</b> Linux, Apache v2	Management of VM, host, virtual network, and scheduling tools, <a href="http://www.opennebula.org/">www.opennebula.org/</a>	XML-RPC, CLI, Java	Xen, KVM	EC2, Elastic Host	Virtual networks, dynamic provisioning
<b>vSphere 4</b> Linux, Windows, proprietary	Virtualizing OS for data centers (Example 3.13), <a href="http://www.vmware.com/products/vsphere/">www.vmware.com/ products/vsphere/</a> [66]	CLI, GUI, Portal, WS	VMware ESX, ESXi	VMware vCloud partners	Data protection, vStorage, VMFS, DRM, HA

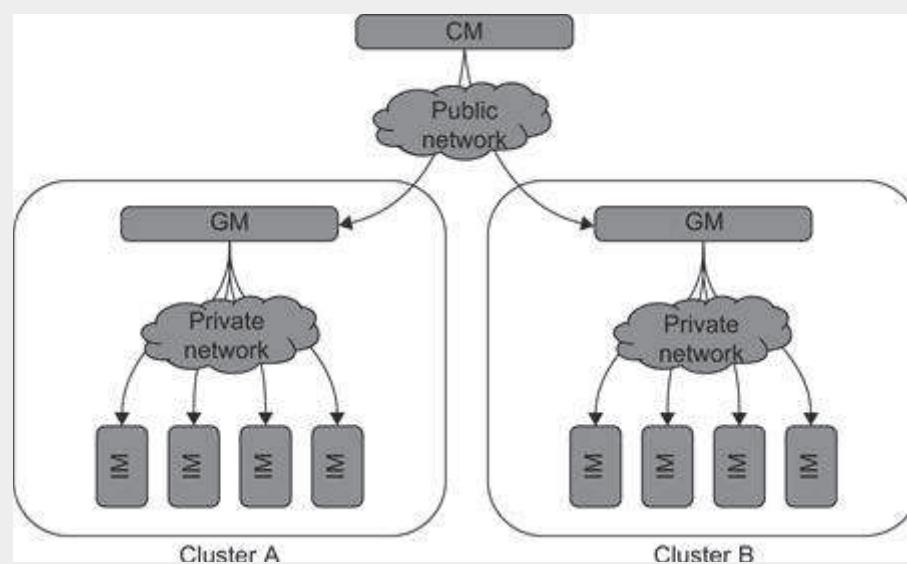
These VI managers are used to create VMs and aggregate them into virtual clusters as elastic resources. Nimbus and Eucalyptus support essentially virtual networks. OpenNebula has additional features to provision dynamic resources and make advance

reservations. All three public VI managers apply Xen and KVM for virtualization. vSphere 4 uses the hypervisors ESX and ESXi from VMware. Only vSphere 4 supports virtual storage in addition to virtual networking and data protection. We will study Eucalyptus and vSphere 4 in the next two examples.

### Example 3.12

#### Eucalyptus for Virtual Networking of Private Cloud

Eucalyptus is an open source software system (Figure 3.27) intended mainly for supporting Infrastructure as a Service (IaaS) clouds. The system primarily supports virtual networking and the management of VMs; virtual storage is not supported. Its purpose is to build private clouds that can interact with end users through Ethernet or the Internet. The system also supports interaction with other private clouds or public clouds over the Internet. The system is short on security and other desired features for general-purpose grid or cloud applications.



**FIGURE 3.27** Eucalyptus for building private clouds by establishing virtual networks over the VMs linking through Ethernet and the Internet. Courtesy of D. Nurmi, et al. [45]

The designers of Eucalyptus [45] implemented each high-level system component as a stand-alone web service. Each web service exposes a well-defined language-agnostic API in the form of a WSDL document containing both operations that the service can perform and input/output data structures. Furthermore, the designers leverage existing web-service features such as WS-Security policies for secure communication between components. The three resource managers in Figure 3.27 are specified below:

- **Instance Manager** controls the execution, inspection, and terminating of

VM instances on the host where it runs.

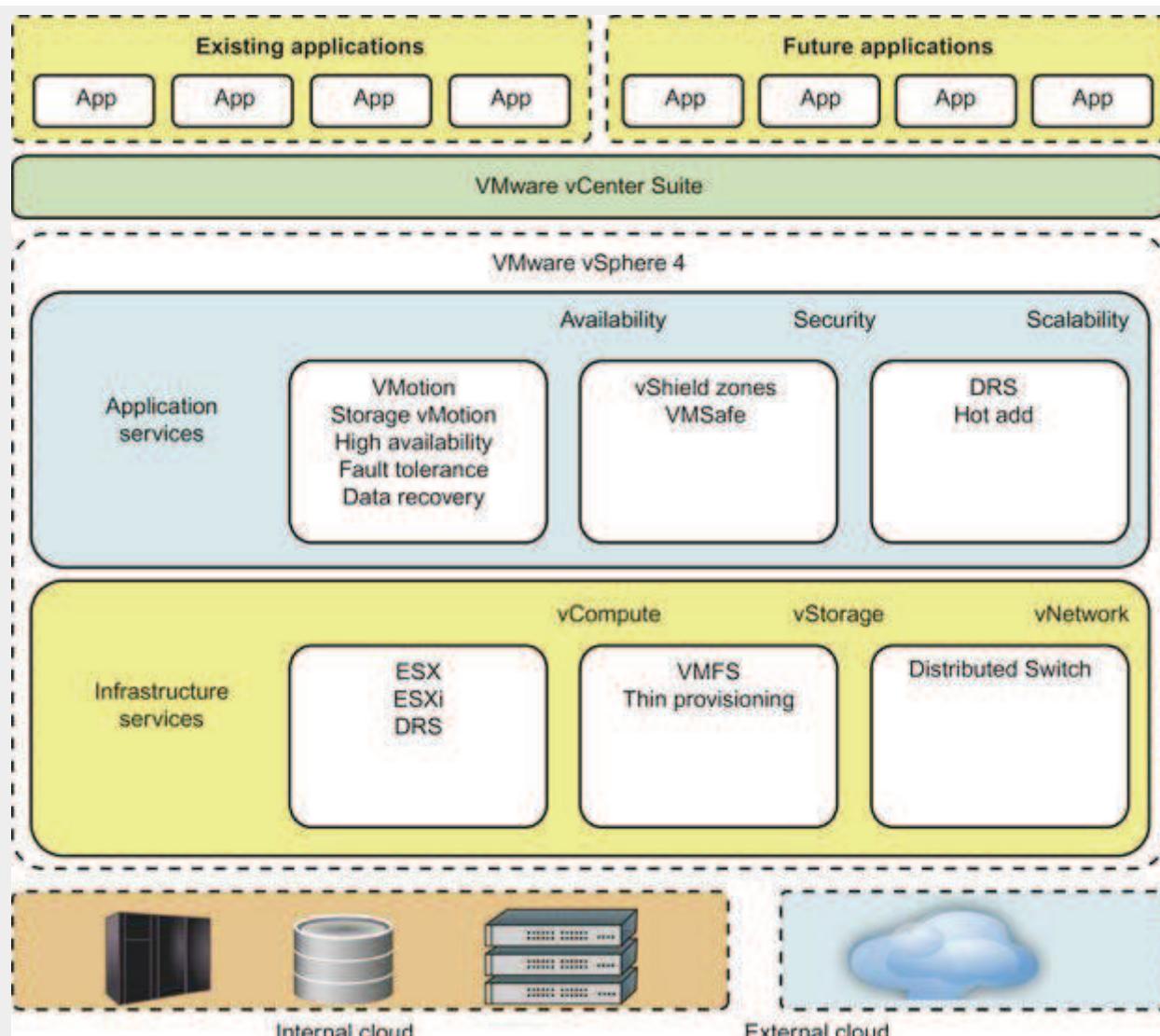
- **Group Manager** gathers information about and schedules VM execution on specific instance managers, as well as manages virtual instance network.
- **Cloud Manager** is the entry-point into the cloud for users and administrators. It queries node managers for information about resources, makes scheduling decisions, and implements them by making requests to group managers.

In terms of functionality, Eucalyptus works like AWS APIs. Therefore, it can interact with EC2. It does provide a storage API to emulate the Amazon S3 API for storing user data and VM images. It is installed on Linux-based platforms, is compatible with EC2 with SOAP and Query, and is S3-compatible with SOAP and REST. CLI and web portal services can be applied with Eucalyptus.

### **Example 3.13**

#### **VMware vSphere 4 as a Commercial Cloud OS**[\[66\]](#)

The vSphere 4 offers a hardware and software ecosystem developed by VMware and released in April 2009. vSphere extends earlier virtualization software products by VMware, namely the VMware Workstation, ESX for server virtualization, and Virtual Infrastructure for server clusters. [Figure 3.28](#) shows vSphere's overall architecture. The system interacts with user applications via an interface layer, called *vCenter*. vSphere is primarily intended to offer virtualization support and resource management of data-center resources in building private clouds. VMware claims the system is the first cloud OS that supports availability, security, and scalability in providing cloud computing services.



**FIGURE 3.28** vSphere/4, a cloud operating system that manages compute, storage, and network resources over virtualized data centers. Courtesy of VMware, April 2010 [72]

The vSphere 4 is built with two functional software suites: *infrastructure services* and *application services*. It also has three component packages intended mainly for virtualization purposes: *vCompute* is supported by ESX, ESXi, and DRS virtualization libraries from VMware; *vStorage* is supported by VMS and thin provisioning libraries; and *vNetwork* offers distributed switching and networking functions. These packages interact with the hardware servers, disks, and networks in the data center. These infrastructure functions also communicate with other external clouds.

The application services are also divided into three groups: *availability*, *security*, and *scalability*. Availability support includes VMotion, Storage VMotion, HA, Fault Tolerance, and Data Recovery from VMware. The security package supports vShield Zones and VMsafe. The scalability package was built with DRS and Hot Add. Interested readers should refer to the vSphere 4 web site for more details regarding these component software functions. To fully understand the use of vSphere 4, users must also learn how to use the vCenter interfaces in order to link with existing applications or to develop new applications.

### 3.5.4 Trust Management in Virtualized Data Centers

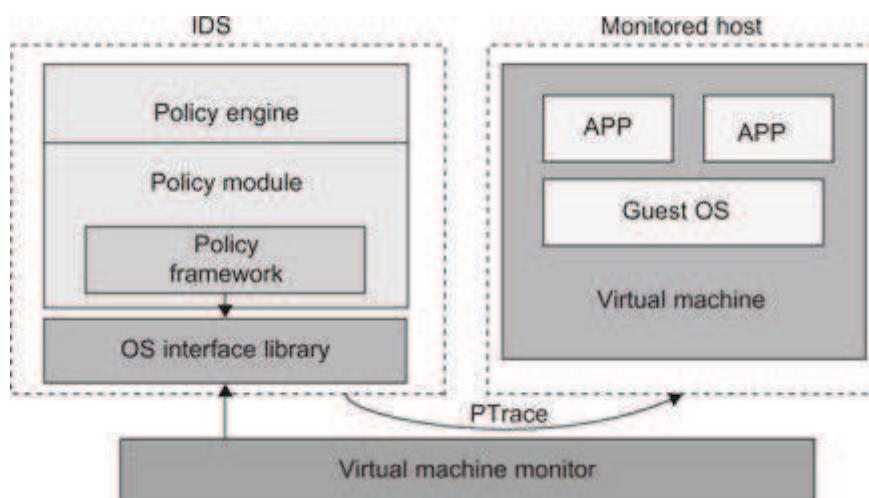
A VMM changes the computer architecture. It provides a layer of software between the operating systems and system hardware to create one or more VMs on a single physical platform. A VM entirely encapsulates the state of the guest operating system running inside it. Encapsulated machine state can be copied and shared over the network and removed like a normal file, which proposes a challenge to VM security. In general, a VMM can provide secure isolation and a VM accesses hardware resources through the control of the VMM, so the VMM is the base of the security of a virtual system. Normally, one VM is taken as a management VM to have some privileges such as creating, suspending, resuming, or deleting a VM.

Once a hacker successfully enters the VMM or management VM, the whole system is in danger. A subtler problem arises in protocols that rely on the “freshness” of their random number source for generating session keys. Considering a VM, rolling back to a point after a random number has been chosen, but before it has been used, resumes execution; the random number, which must be “fresh” for security purposes, is reused. With a stream cipher, two different plaintexts could be encrypted under the same key stream, which could, in turn, expose both plaintexts if the plaintexts have sufficient redundancy. Noncryptographic protocols that rely on freshness are also at risk. For example, the reuse of TCP initial sequence numbers can raise TCP hijacking attacks.

#### 3.5.4.1 VM-Based Intrusion Detection

Intrusions are unauthorized access to a certain computer from local or network users and intrusion detection is used to recognize the unauthorized access. An intrusion detection system (IDS) is built on operating systems, and is based on the characteristics of intrusion actions. A typical IDS can be classified as a *host-based IDS (HIDS)* or a *network-based IDS (NIDS)*, depending on the data source. A HIDS can be implemented on the monitored system. When the monitored system is attacked by hackers, the HIDS also faces the risk of being attacked. A NIDS is based on the flow of network traffic which can't detect fake actions.

Virtualization-based intrusion detection can isolate guest VMs on the same hardware platform. Even some VMs can be invaded successfully; they never influence other VMs, which is similar to the way in which a NIDS operates. Furthermore, a VMM monitors and audits access requests for hardware and system software. This can avoid fake actions and possess the merit of a HIDS. There are two different methods for implementing a VM-based IDS: Either the IDS is an independent process in each VM or a high-privileged VM on the VMM; or the IDS is integrated into the VMM and has the same privilege to access the hardware as well as the VMM. Garfinkel and Rosenblum [17] have proposed an IDS to run on a VMM as a high-privileged VM. [Figure 3.29](#) illustrates the concept.



**FIGURE 3.29** The architecture of livewire for intrusion detection using a dedicated VM. Courtesy of Garfinkel and Rosenblum, 2002 [17]

The VM-based IDS contains a policy engine and a policy module. The policy framework can monitor events in different guest VMs by operating system interface library and PTrace indicates trace to secure policy of monitored host. It's difficult to predict and prevent all intrusions without delay. Therefore, an analysis of the intrusion action is extremely important after an intrusion occurs. At the time of this writing, most computer systems use logs to analyze attack actions, but it is hard to ensure the credibility and integrity of a log. The IDS log service is based on the operating system kernel. Thus, when an operating system is invaded by attackers, the log service should be unaffected.

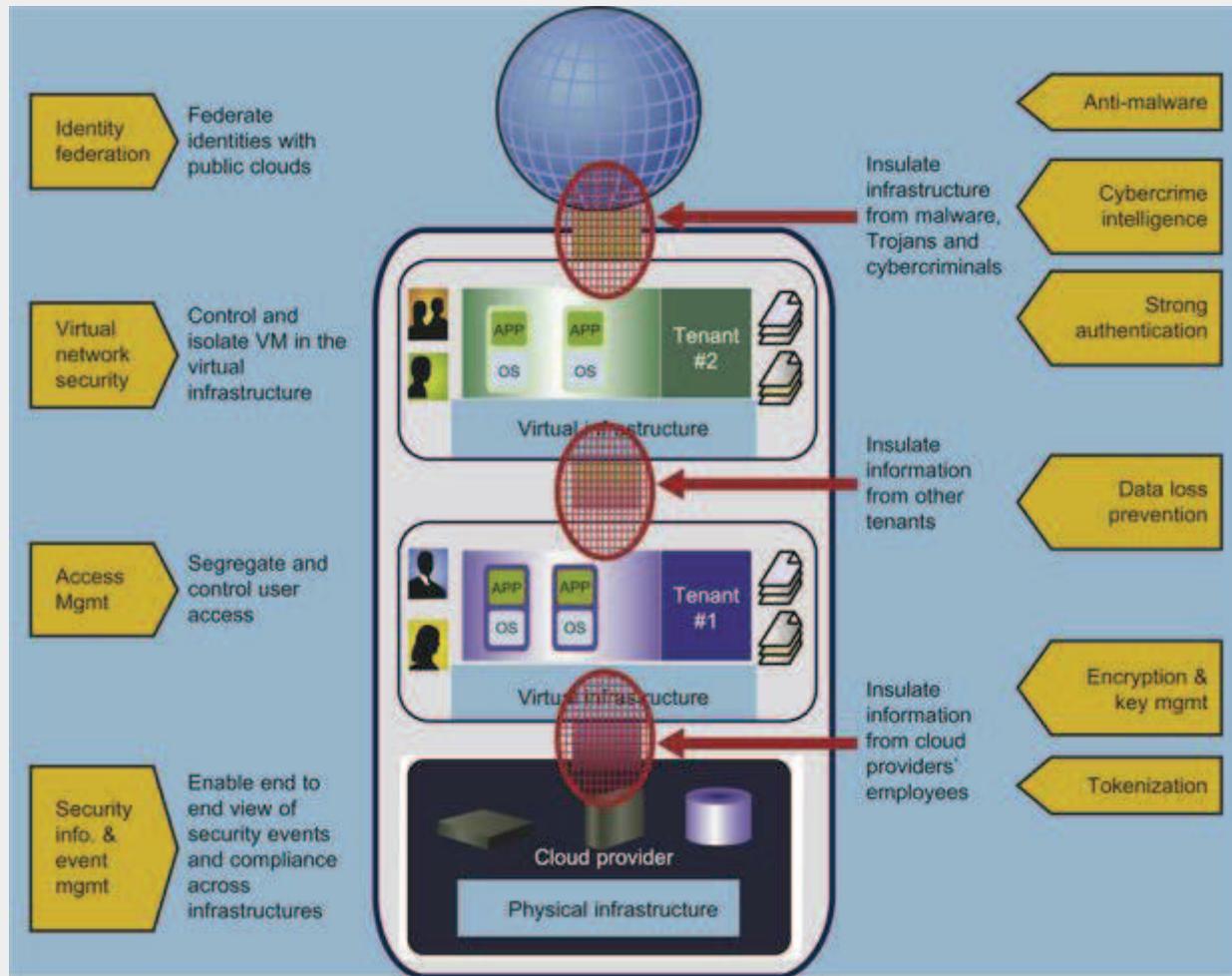
Besides IDS, honeypots and honeynets are also prevalent in intrusion detection. They attract and provide a fake system view to attackers in order to protect the real system. In addition, the attack action can be analyzed, and a secure IDS can be built. A honeypot is a purposely defective system that simulates an operating system to cheat and monitor the actions of an attacker. A honeypot can be divided into physical and virtual forms. A guest operating system and the applications running on it constitute a VM. The host operating system and VMM must be guaranteed to prevent attacks from the VM in a virtual honeypot.

### Example 3.14

#### EMC Establishment of Trusted Zones for Protection of Virtual Clusters Provided to Multiple Tenants

EMC and VMware have joined forces in building security middleware for trust management in distributed systems and private clouds. The concept of *trusted zones* was established as part of the virtual infrastructure. Figure 3.30 illustrates the concept of creating trusted zones for virtual clusters (multiple applications and OSes for each tenant) provisioned in separate virtual environments. The physical infrastructure is shown at the bottom, and marked as a cloud provider. The virtual clusters or infrastructures are shown

in the upper boxes for two tenants. The public cloud is associated with the global user communities at the top.



**FIGURE 3.30** Techniques for establishing trusted zones for virtual cluster insulation and VM isolation. Courtesy of L. Nick, EMC [40]

The arrowed boxes on the left and the brief description between the arrows and the zoning boxes are security functions and actions taken at the four levels from the users to the providers. The small circles between the four boxes refer to interactions between users and providers and among the users themselves. The arrowed boxes on the right are those functions and actions applied between the tenant environments, the provider, and the global communities.

Almost all available countermeasures, such as anti-virus, worm containment, intrusion detection, encryption and decryption mechanisms, are applied here to insulate the trusted zones and isolate the VMs for private tenants. The main innovation here is to establish the trust zones among the virtual clusters. The end result is to enable an end-to-end view of security events and compliance across the virtual clusters dedicated to different tenants. We will discuss security and trust issues in [Chapter 7](#) when we study clouds in more detail.

## 3.6 Bibliographic Notes and Homework Problems

Good reviews on virtualization technology can be found in Rosenblum, et al. [53,54] and in Smith and Nair [58,59]. White papers at [71,72] report on virtualization products at VMware, including the vSphere 4 cloud operating system. The Xen hypervisor was introduced in [7,13,42] and KVM in [31]. Qian, et al. [50] have given some theoretical treatment of the subject. ISA-level virtualization and binary translation techniques are treated in [3] and in Smith and Nair [58]. Some comparison of virtualizing software environments can be found in Buyya, et al. [9]. Hardware-level virtualization is treated in [1,7,8,13,37,43,54,67]. Intel's support of hardware-level virtualization is treated in [62]. Some entries in Table 3.5 are taken from Buyya, et al. [9].

For GPU computing on VMs, the readers are referred to [57]. The x86 host virtualization is treated in [2]. Pentium virtualization is treated in [53]. treated in [11,24,26,52]. I/O virtualization is

Sun Microsystems reports on OS-level virtualization in [64]. OpenVZ is introduced in its user's guide [65]. Virtualization in Windows NT machines is described in [77,78]. For GPU computing on VMs, readers are referred to [53]. The x86 host virtualization is treated in [2]. Pentium virtualization is treated in [53]. The book [24] have a good coverage of hardware support for virtualization. Virtual clusters are treated in [8,12,20,25,49,55,56]. In particular, Duke's COD is reported in [12] and Purdue's Violin in [25,55]. Memory virtualization is treated in [1,10,13,51,58,68].

Hardware-level virtualization is treated in [1, 7,8,13,41,47,58,73]. Intel's support of hardware-level virtualization is treated in [68]. Wells, et al. [74] have studied multi-core virtualization. The integration of multi-core and virtualization on future CPU/GPU chips posts a very hot research area, called asymmetric CMP as reported in [33,39,63,66,74]. Architectural supports for virtualized chip multiprocessors have been also studied in [17,28,30,39,66]. The maturity of this co-design CMP approach will greatly impact the future development of HPC and HTC systems.

Server consolidation in virtualized data centers is treated in [29,39,75]. Power consumption in virtualized data centers is treated in [44,46]. Literatures [46,60,61,62] discussed the virtual resource management in data centers. Kochut gives an analytical model for virtualized data centers [32]. Security protection and trust management for virtualized data centers are treated in [18,45]. For virtual storage, readers are referred to the literature [27,36,43,48,76,79]. Specific references for the examples are identified in the figure captions or in the text description of the example. The Eucalyptus was reported in [45], vSphere in [72], Parallax in [43]. The vCUDA was reported in [57] for CUDA programming on the VMs.

## Acknowledgments

This chapter is coauthored by Zhibin Yu of Huazhong University of Science and Technology (HUST), China and by Kai Hwang of USC. Dr. Hai Jin and Dr. Xiaofei Liao of HUST have extended significant technical support to this work. The authors would like to

thank Dr. Zhong-Yuan Qin of Southeast University, China, Fan Zhang of Tsinghua University, and Lizhong Chen and Zhou Zhao of USC for their assistance in drawing several sketched figures and updating the references in this chapter.

## References

1. Advanced Micro Devices. AMD Secure Virtual Machine Architecture Reference Manual, 2008.
2. Adams K, Agesen O. A comparison of software and hardware techniques for x86 virtualization. In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. San Jose, CA. October 2006;21–25.
3. Adve V, Lattner C, et al. LLVA: A low-level virtual instruction set architecture. In: *Proceedings of the 36th International Symposium on Micro-architecture (MICRO-36 '03)*. 2003.
4. Alonso J, Silva L, Andrzejak A, Silva P, Torres J. High-available grid services through the use of virtualized clustering. In: *Proceedings of the 8th Grid Computing Conference*. 2007.
5. Anedda P, Gaggero M, et al. A general service-oriented approach for managing virtual machine allocation. In: *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC 2009)*. ACM Press March 2009;9–12.
6. Andre Lagar-Cavilla H, Whitney JA, Scannell A, et al. SnowFlock: rapid virtual machine cloning for cloud computing. In: *Proceedings of EuroSystems*. 2009.
7. Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. In: *Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP19)*. ACM Press 2003;164–177.
8. Bugnion E, Devine S, Rosenblum M. Disco: running commodity OS on scalable multiprocessors. In: *Proceedings of SOSP*. 1997.
9. Buyya R, Broberg J, Goscinski A, eds. *Cloud Computing: Principles and Paradigms*. New York: Wiley Press; 2011.
10. Chadha V, Illikkal R, Iyer R. I/O Processing in a virtualized platform: a simulation-driven approach. In: *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE)*. 2007.
11. Chandra R, Zeldovich N, Sapuntzakis C, Lam MS. The collective: a cache-based system management architecture. In: *Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI '05)*. USENIX, Boston. May 2005;259–272.
12. Chase J, Grit L, Irwin D, Moore J, Sprenkle S. Dynamic virtual cluster in a grid site manager. In: *IEEE Int'l Symp on High Performance Distributed Computing, (HPDC-12)*. 2003.
13. Chisnall D. *The Definitive Guide to the Xen Hypervisor* Prentice Hall, International 2007.
14. Clark C, Fraser K, Hand S, et al. Live migration of virtual machines. In: *Proceedings*

*of the Second Symposium on Networked Systems Design and Implementation (NSDI '05).* 2005;273–286.

15. Dong Y, Dai J, et al. Towards high-quality I/O virtualization. In: *Proceedings of SYSTOR 2009*. The Israeli Experimental Systems Conference. 2009.
16. Elnozahy E, Kistler M, Rajamony R. Energy-efficient server clusters. In: *Proceedings of the 2nd Workshop on Power-Aware Computing Systems*. February 2002.
17. Frich J, et al. On the potential of NoC virtualization for multicore chips. In: *IEEE Int'l Conf on Complex, Intelligent and Software-Intensive Systems*. 2008;801–807.
18. T. Garfinkel, M. Rosenblum, A virtual machine introspection-based architecture for intrusion detection, 2002.
19. Grit L, Irwin D, Yumerefendi A, Chase J. Virtual machine hosting for networked clusters: building the foundations for autonomic orchestration. In: *First International Workshop on Virtualization Technology in Distributed Computing (VTDC)*. November 2006.
20. Gupta D, Lee S, Vrable M, et al. Difference engine: Harnessing memory redundancy in virtual machines. In: *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*. 2008;309–322.
21. Hines M, Gopalan K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In: *Proceedings of the ACM/USENIX International Conference on Virtual Execution Environments (VEE '09)*. 2009;51–60.
22. Hirofuchi T, Nakada H, et al. A live storage migration mechanism over WAN and its performance evaluation. In: *Proceedings of the 4th International Workshop on Virtualization Technologies in Distributed Computing*, 15 June. Barcelona, Spain: ACM Press; 2009.
23. Hwang K, Li D. Trusted cloud computing with secure resources and data coloring. *IEEE Internet Comput.* 2010;(September/October):30–39.
24. Intel Open Source Technology Center. *System Virtualization—Principles and Implementation* Beijing, China: Tsinghua University Press; 2009.
25. Jiang X, Xu D. VIOLIN: Virtual internetworking on overlay infrastructure. In: *Proceedings of the International Symposium on Parallel and Distributed Processing and Applications*. 2004;937–946.
26. Jin H, Deng L, Wu S, Shi X, Pan X. Live virtual machine migration with adaptive memory compression. In: *Proceedings of the IEEE International Conference on Cluster Computing*. 2009.
27. Jin K, Miller E. The effectiveness of deduplication on virtual machine disk images. In: *Proceedings of SYSTOR*, 2009. The Israeli Experimental Systems Conference. 2009.
28. Jones S, Arpaci-Disseau A, Arpaci-Disseau R. Geiger: Monitoring the buffer cache in a virtual machine environment. In: *ACM ASPLOS*. San Jose, CA. October 2006;13–14.
29. Kamoun F. Virtualizing the datacenter without compromising server performance. *ACM Ubiquity* 2009 2009.

30. D. Kim, H. Kim, J. Huh, Virtual snooping: Filtering snoops in virtualized multicore, in: 43rd Annual IEEE/ACM Int'l Symposium on Microarchitecture (MICRO-43).
31. Kivity A, et al. KVM: The linux virtual machine monitor. In: *Proceedings of the Linux Symposium*. Ottawa, Canada. 2007;225.
32. Kochut A. On impact of dynamic virtual machine reallocation on data center efficiency. In: Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS). 2008.
33. Kumar R, et al. Heterogeneous chip multtiprocessors. *IEEE Comput Mag*. 2005;38(November):32–38.
34. Kyrre B. Managing large networks of virtual machines. In: *Proceedings of the 20th Large Installation System Administration Conference*. 2006;205–214.
35. Lange J, Dinda P. Transparent network services via a virtual traffic layer for virtual machines. In: *Proceedings of High Performance Distributed Computing*. Monterey, CA: ACM Press; June 2007;25–29.
36. Liguori A, Hensbergen E. Experiences with content addressable storage and virtual disks. In: *Proceedings of the Workshop on I/O Virtualization (WIOV '08)*. 2008.
37. Liu H, Jin H, Liao X, Hu L, Yu C. Live migration of virtual machine based on full system trace and replay. In: *Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC '09)*. 2009;101–110.
38. Mainwaring A, Culler D. Design challenges of virtual networks: Fast, general-purpose communication. In: *Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*. 1999.
39. Marty M, Hill M. Virtual hierarchies to support server consolidation. In: *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*. 2007.
40. M. McNett, D. Gupta, A. Vahdat, G.M. Voelker, Usher: An extensible framework for managing clusters of virtual machines, in: 21st Large Installation System Administration Conference (LISA) 2007.
41. Menasce D. Virtualization: Concepts, applications., performance modeling. In: *Proceedings of the 31st International Computer Measurement Group Conference*. 2005;407–414.
42. Menon A, Renato J, Turner Y. Diagnosing performance overheads in the Xen virtual machine environment. In: *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*. 2005.
43. Meyer D, et al. Parallax: Virtual disks for virtual machines. In: *Proceedings of EuroSys*. 2008.
44. Nick J. Journey to the private cloud: Security and compliance. In: *Technical presentation by EMC Visiting Team, May 25*. Beijing: Tsinghua University; 2010.
45. Nurmi D, et al. The eucalyptus open-source cloud computing system. In: *Proceedings of the 9th IEEE ACM International Symposium on Cluster Computing and The Grid (CCGrid)*. Shanghai, China. September 2009;124–131.

46. P. Padala, et al., Adaptive control of virtualized resources in utility computing environments, in: Proceedings of EuroSys 2007.
47. Peterson L, Bavier A, Fiuczynski ME, Muir S. Experiences Building PlanetLab. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI2006)*. 6–8 November 2006.
48. Pfaff B, Garfinkel T, Rosenblum M. Virtualization aware file systems: Getting beyond the limitations of virtual disks. In: *Proceedings of USENIX Networked Systems Design and Implementation (NSDI 2006)*. May 2006;353–366.
49. Pinheiro E, Bianchini R, Carrera E, Heath T. Dynamic cluster reconfiguration for power and performance. In: Benini L, ed. *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers 2003.
50. Qian H, Miller E, et al. Agility in virtualized utility computing. In: *Proceedings of the Third International Workshop on Virtualization Technology in Distributed Computing (VTDC 2007)*. 12 November 2007.
51. H. Raj, I. Ganev, K. Schwan, Self-Virtualized I/O: High Performance, Scalable I/O Virtualization in Multi-core Systems, Technical Report GIT-CERCS-06-02, CERCS, Georgia Tech, 2006, [www.cercks.gatech.edu/tech-reports/tr2006/git-cercks-06-02.pdf](http://www.cercs.gatech.edu/tech-reports/tr2006/git-cercks-06-02.pdf).
52. Robin J, Irvine C. Analysis of the Intel pentium's ability to support a secure virtual machine monitor. In: 2000; *Proceedings of the 9th USENIX Security Symposium*. Vol. 9.
53. Rosenblum M. The reincarnation of virtual machines. *ACM QUEUE* 2004.
54. Rosenblum M, Garfinkel T. Virtual machine monitors: current technology and future trends. *IEEE Comput.* 2005;38(5):39–47.
55. Ruth P, et al. *Automatic Live Migration of Virtual Computational Environments in a Multi-domain Infrastructure* Purdue University 2006.
56. Sapuntzakis C, Chandra R, Pfaff B, et al. Optimizing the migration of virtual computers. In: *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*. Boston. 9–11 December 2002.
57. Shi L, Chen H, Sun J. vCUDA: GPU accelerated high performance computing in virtual machines. In: *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*. 2009.
58. Smith J, Nair R. *Virtual Machines: Versatile Platforms for Systems and Processes* Morgan Kaufmann 2005.
59. Smith J, Nair R. The architecture of virtual machines. *IEEE Comput.* 2005.
60. Song Y, Wang H, et al. Multi-tiered on-demand resource scheduling for VM-based data center. In: *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2009.
61. Sotomayor B, Keahey K, Foster I. Combining batch execution and leasing using virtual machines. In: *Proceedings of the 17th International Symposium on High-Performance Distributed Computing*. 2008.
62. Steinder M, Whalley I, et al. Server virtualization in autonomic management of heterogeneous workloads. *ACM SIGOPS Oper Syst Rev.* 2008;42(1):94–95.
63. Suleman M, Patt Y, Sprangle E, Rohillah A. *Asymmetric chip multiprocessors*:

*balancing hardware efficiency and programming efficiency* 2007.

64. Sun Microsystems. Solaris Containers: Server Virtualization and Manageability, Technical white paper, September 2004.
65. SWsoft, Inc. OpenVZ User's Guide, <http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf>, 2005.
66. Trivino F, et al. *Virtualizing network on chip resources in chip multiprocessors*. *J Microprocess Microsyst*. 2010;35:245–230 <http://www.elsevier.com/locate/micro>; 2010.
67. Xu J, Zhao M, et al. On the use of fuzzy modeling in virtualized datacenter management. In: *Proceedings of the 4th International Conference on Autonomic Computing (ICAC07)*. 2007.
68. Ublig R, et al. Intel virtualization technology. *IEEE Comput*. 2005.
69. Van H, Tran F. Autonomic virtual resource management for service hosting platforms. *CLOUD* 2009.
70. Verma A, Ahuja P, Neogi A. pMapper: Power and migration cost aware application placement in virtualized systems. In: *Proceedings of the 9th International Middleware Conference*. 2008;243–264.
71. VMware (white paper). Understanding Full Virtualization, Paravirtualization, and Hardware Assist, [www.vmware.com/files/pdf/VMware\\_paravirtualization.pdf](http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf).
72. VMware (white paper). The vSphere 4 Operating System for Virtualizing Datacenters, News release, February 2009, [www.vmware.com/products/vsphere/](http://www.vmware.com/products/vsphere/), April 2010.
73. Walters J, et al. A comparison of virtualization technologies for HPC. In: *Proceedings of Advanced Information Networking and Applications (AINA)*. 2008.
74. Wells P, Chakraborty K, Sohi GS. Dynamic heterogeneity and the need for multicore virtualization. *ACM SIGOPS Operat Syst Rev*. 2009;43(2):5–14.
75. Wood T, Levin G, Shenoy P. Memory buddies: Exploiting page sharing for smart collocation in virtualized data centers. In: *Proceedings of the 5th International Conference on Virtual Execution Environments (VEE)*. 2009.
76. Xun J, Chen K, Zheng W. Amigo file system: CAS based storage management for a virtual cluster system. In: *Proceedings of IEEE 9th International Conference on Computer and Information Technology (CIT)*. 2009.
77. Y. Yu, OS-level Virtualization and Its Applications, Ph.D. dissertation, Computer Science Department, SUNY, Stony Brook, New York, December 2007.
78. Yu Y, Guo F, et al. A feather-weight virtual machine for windows applications. In: *Proceedings of the 2nd International Conference on Virtual Execution Environments (VEE)*. Ottawa, Canada. 14–16 June 2006.
79. Zhao M, Zhang J, et al. Distributed file system support for virtual machines in grid computing. In: *Proceedings of High Performance Distributed Computing*. 2004.

## Homework Problems

### Problem 3.1

Briefly answer the following questions on virtualization levels. Highlight the key points and identify the distinctions in different approaches. Discuss their relative advantages, shortcomings and limitations. Also identify example systems implemented at each level.

## Problem 3.2

Explain the differences between hypervisor and para-virtualization and give one example VMM (virtual machine monitor), that was built in each of the two categories.

## Problem 3.3

Install the VMware Workstation on a Windows XP or Vista personal computer or laptop, and then install Red Hat Linux and Windows XP in the VMware Workstation. Configure the network settings of Red Hat Linux and Windows XP to get on the Internet. Write an installation and configuration guide for the VMware Workstation, Red Hat Linux, and Windows XP systems. Include any troubleshooting tips in the guide.

## Problem 3.4

Download a new kernel package from [www.kernel.org/](http://www.kernel.org/). Compile it in Red Hat Linux in the VMware Workstation installed in Problem 3.3 with Red Hat Linux on a real computer. Compare the time required for the two compilations. Which one takes longer to compile? What are their major differences?

## Problem 3.5

Install Xen on a Red Hat Linux machine in two methods from the binary code or from the source code. Compile installation guides for the two methods used. Describe the dependencies of utilities and packages along with troubleshooting tips.

## Problem 3.6

Install Red Hat Linux on the Xen you installed in Problem 3.5. Download nbench from [www.tux.org/~mayer/linux/bmark.html](http://www.tux.org/~mayer/linux/bmark.html). Run the nbench on the VM using Xen and on a real machine. Compare the performance of the programs on the two platforms.

## Problem 3.7

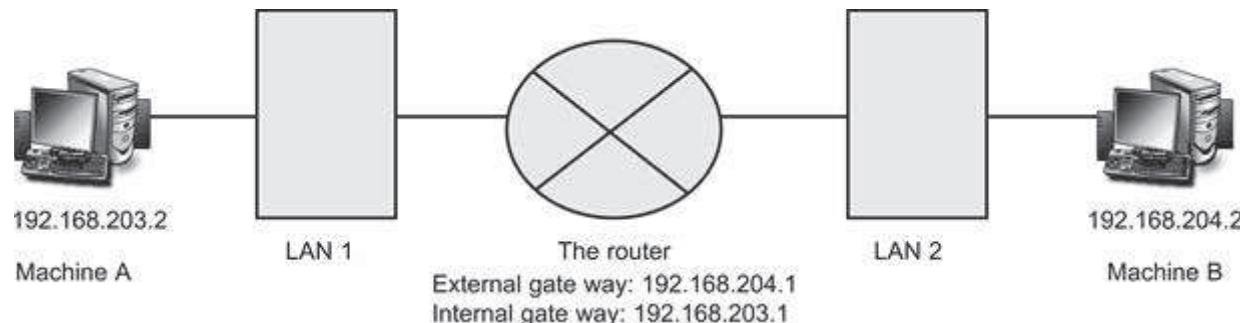
Use the utilities for easing deployment of Google enterprise applications in VMs. The Google-vm-deployment tool can be downloaded from <http://code.google.com/p/google-vm-deployment/>.

## Problem 3.8

Describe the approaches used to exchange data among the domains of Xen and design experiments to compare the performance of data communication between the domains. This is designed to familiarize you with the Xen programming environment. It may require a longer period of time to port the Xen code, implement the application code, perform the experiments, collect the performance data, and interpret the results.

## Problem 3.9

Build your own LAN by using the VMware Workstation. The topological structure of the LAN is specified in [Figure 3.31](#). Machine A is required to install Red Hat Linux while machine B is required to install Windows XP.



**FIGURE 3.31** The topological structure of the virtual LAN.

## Problem 3.10

Study the relevant papers [33,63,74] on asymmetric or heterogeneous chip multiprocessors (CMP). Write a study report to survey the area, identify the key research issues, review the current development and open research challenges lying ahead.

## Problem 3.11

Study the relevant papers [17,28,30,66] on network on chip (NoC) and virtualization of NoC resources for multi-core CMP design and applications. Repeat [Problem 3.10](#) with a survey report after the research study.

## Problem 3.12

Hardware and software resource deployment are often complicated and time-consuming. Automatic VM deployment can significantly reduce the time to instantiate new services or reallocate resources depending on user needs. Visit the following web site for more information.

[http://wiki.systemimager.org/index.php/Automating\\_Xen\\_VM\\_deployment\\_with\\_SystemImager](http://wiki.systemimager.org/index.php/Automating_Xen_VM_deployment_with_SystemImager)  
Report your experience with automatic deployment using the SystemImager and Xen-

tools.

## Problem 3.13

Design an experiment to analyze the performance of Xen live migration for I/O read-intensive applications. The performance merits include the time consumed by the precopy phase, the downtime, the time used by the pull phase, and the total migration time.

## Problem 3.14

Design an experiment to test the performance of Xen live migration for I/O write-intensive applications. The performance metrics include the time consumed by the precopy phase, the downtime, the time used by the pull phase, and the total migration time. Compare the results with those from Problem 3.13.

## Problem 3.15

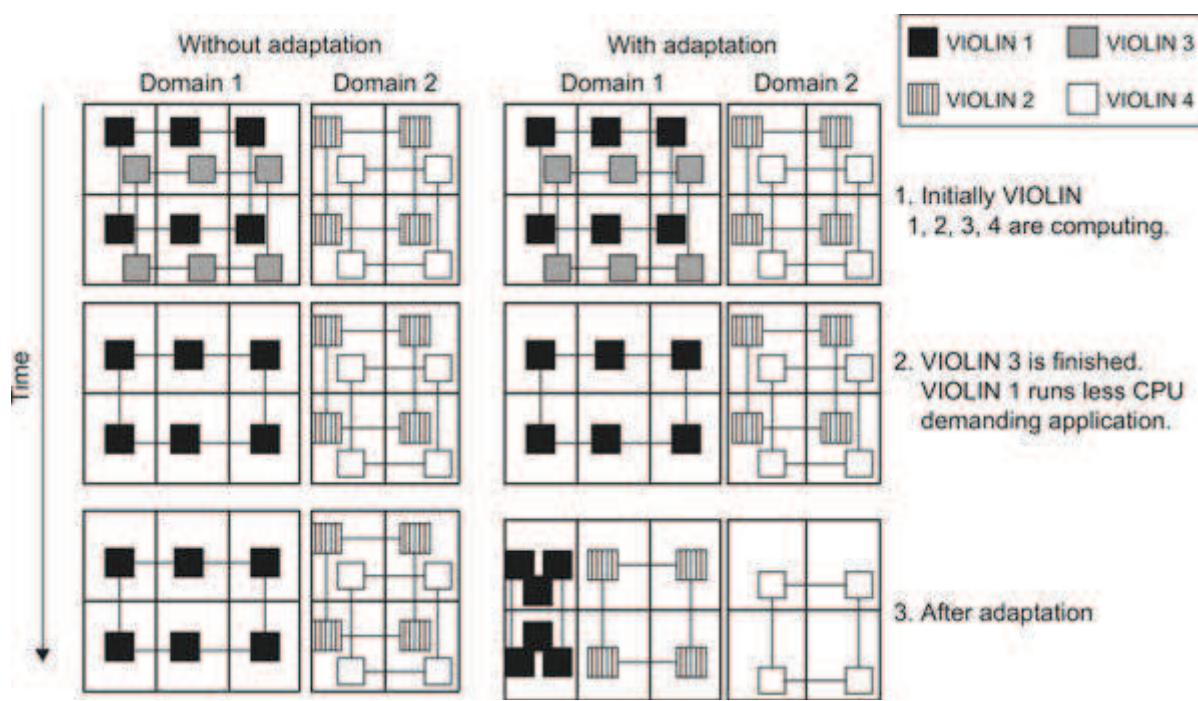
Design and implement a VM execution environment for grid computing based on VMware Server. The environment should enable grid users and resource providers to use services that are unique to a VM-based approach to distributed computing. Users can define customized execution environments which can then be archived, copied, shared, and instantiated as multiple runtime clones.

## Problem 3.16

Design a large-scale virtual cluster system. This problem may require three students to work together for a semester. Assume that users can create multiple VMs at one time. Users can also manipulate and configure multiple VMs at the same time. Common software such as OS or libraries are preinstalled as templates. These templates enable users to create a new execution environment rapidly. Finally, you can assume that users have their own profiles which store the identification of data blocks.

## Problem 3.17

Figure 3.32 shows another VIOLIN adaptation scenario for changes in virtual environments. There are four VIOLIN applications running in two cluster domains. Trace the three steps of VIOLIN job execution and discuss the gains in resource utilization after live migration of the virtual execution environment in the two cluster domains. You can check your results against the cited paper to compare your observations.



**FIGURE 3.32** An adaptation scenario with four VIOLINs running in two cluster domains in the VIOLIN Virtual clustering experiments. Courtesy of P. Ruth, et al. [55]

## Problem 3.18

After studying the material presented in Section 3.3.5, plus reading the papers by Wells, et al. [74] and by Marty and Hill in [39] answer the following two questions:

- Distinguish virtual cores from physical cores and discuss the mapping technique in Wells's paper on upgrading resource utilization and fault tolerance in using virtualized multicore processors.
- Study the cache coherence protocol presented in the Marty and Hill paper and discuss its feasibility and advantages to implement on many-core CMP in the future.

## PART 2

# Computing Clouds, Service-Oriented Architecture, and Programming

## OUTLINE

Introduction

Chapter 4 Cloud Platform Architecture over Virtualized Data Centers

Chapter 5 Service-Oriented Architectures for Distributed Computing

Chapter 6 Cloud Programming and Software Environments

# Introduction

---

Three chapters in Part 2 are devoted to cloud computing, including various cloud platforms for IaaS (infrastructure as a service), PaaS (platform as a service), and SaaS (software as a service) applications. We present service-oriented architectures developed in recent years. Parallel and distributed computing paradigms and their software support, language tools, and programming environments are covered for various cloud computing services.

## Chapter 4: Cloud Platform Architecture over Virtualized Data Centers

This chapter covers the design principles and enabling technologies for cloud architecture and data center design. We start with data center design and management. Then we present the design principles of cloud platforms. We cover layered platform designs from infrastructure clouds including compute clouds, storage clouds, and application clouds. In particular, we study the center automation technology and interconnection networks. We apply virtualization support in resource provisioning and infrastructure management. Major public cloud platforms covered include Google AppEngine (GAE), Amazon Web Service (AWS), and Microsoft Windows Azure. The cloud security and trust management issues are studied.

This chapter is authored by Kai Hwang with partial contributions by Rajkumar Buyya and Kang Chen and the final manuscript was edited by Geoffrey Fox.

# Chapter 5: Service-Oriented Architectures for Distributed Computing

We cover two major forms of distributed services—REST and web services—and the extension of the latter in Grid systems. Workflow is used to orchestrate or integrate multiple services and we describe the general approach used in workflow, the BPEL Web Service standard including Pegasus, Taverna, Kepler, Trident, and Swift. Example web interfaces or portals are described using Web Service (portlet) and Web 2.0 (gadget). We handle data and metadata in distributed systems using service registries and semantic web/grid.

This chapter is authored by Geoffrey Fox and Albert Zomaya with partial contributions by Rajkumar Buyya and technical assistance from Ali Javadzadeh Boloori, Chen Wang, Shrideep Pallickara, Marlon Pierce, Suresh Marru, Michael McLennan, George Adams, III, Gerhard Klimeck, and Michael Wilde. The final manuscript was edited by Kai Hwang.

# Chapter 6: Cloud Programming and Software Environments

We introduce major cloud programming paradigms: MapReduce, BigTable, Twister, Dryad, DryadLINQ, Hadoop, Sawzall, and Pig Latin. We use concrete service examples to explain the implementation and application requirements in the cloud. We review core service models and access technologies. Cloud services provided by Google App Engine, Amazon Web Service, and Microsoft Windows Azure are illustrated by example applications. In particular, we illustrate how-to programming the GAE, AWS EC2, S3, EBS, and others. We review the open-source Eucalyptus, Nimbus, and OpenNebula and the startup Manjrasoft Aneka system for cloud computing.

This chapter is authored by Geoffrey Fox and Albert Zomaya with partial contributions by Rajkumar Buyya ([Section 6.5.3](#)) and Judy Qiu ([Section 6.2.6](#)). Special technical assistances are from Gregor von Laszewski, Javier Diaz, Archit Kulshrestha, Andrew Younge, Reza Moravaeji, Javid Teheri, and Renato Figueiredo. The final manuscript was edited by Kai Hwang.

# CHAPTER 4

# Cloud Platform Architecture over Virtualized Data Centers

## CHAPTER OUTLINE

### Summary

#### 4.1 Cloud Computing and Service Models

- 4.1.1 Public, Private, and Hybrid Clouds
- 4.1.2 Cloud Ecosystem and Enabling Technologies
- 4.1.3 Infrastructure-as-a-Service (IaaS)
- 4.1.4 Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS)

#### 4.2 Data-Center Design and Interconnection Networks

- 4.2.1 Warehouse-Scale Data-Center Design
- 4.2.2 Data-Center Interconnection Networks
- 4.2.3 Modular Data Center in Shipping Containers
- 4.2.4 Interconnection of Modular Data Centers
- 4.2.5 Data-Center Management Issues

#### 4.3 Architectural Design of Compute and Storage Clouds

- 4.3.1 A Generic Cloud Architecture Design
- 4.3.2 Layered Cloud Architectural Development
- 4.3.3 Virtualization Support and Disaster Recovery
- 4.3.4 Architectural Design Challenges

#### 4.4 Public Cloud Platforms: GAE, AWS, and Azure

- 4.4.1 Public Clouds and Service Offerings
- 4.4.2 Google App Engine (GAE)
- 4.4.3 Amazon Web Services (AWS)
- 4.4.4 Microsoft Windows Azure

#### 4.5 Inter-cloud Resource Management

- 4.5.1 Extended Cloud Computing Services
- 4.5.2 Resource Provisioning and Platform Deployment
- 4.5.3 Virtual Machine Creation and Management
- 4.5.4 Global Exchange of Cloud Resources

## 4.6 Cloud Security and Trust Management

- 4.6.1 Cloud Security Defense Strategies
- 4.6.2 Distributed Intrusion/Anomaly Detection
- 4.6.3 Data and Software Protection Techniques
- 4.6.4 Reputation-Guided Protection of Data Centers

## 4.7 Bibliographic Notes and Homework Problems

### Acknowledgements

### References

### Homework Problems

This chapter covers design principles, architectures, and enabling technologies of cloud platforms. We begin with a discussion of data-center design and management. Next, we present design choices for building compute and storage cloud platforms. We cover layered platform structure, virtualization support, resource provisioning, and infrastructure management. Several public cloud platforms are also studied, including Amazon Web Services, the Google App Engine, and Microsoft Azure. Subsequent chapters are devoted to service-oriented architectures, cloud computing paradigms, programming environments, and future cloud extensions.

cloud computing, Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), data centers, cloud architecture, public clouds, resource management, cloud security

## Summary

This chapter covers design principles, architectures, and enabling technologies of cloud platforms. We begin with a discussion of data-center design and management. Next, we present design choices for building compute and storage cloud platforms. We cover layered platform structure, virtualization support, resource provisioning, and infrastructure management. Several public cloud platforms are also studied, including Amazon Web Services, the Google App Engine, and Microsoft Azure. Subsequent chapters are devoted to service-oriented architectures, cloud computing paradigms, programming environments, and future cloud extensions.

## 4.1 Cloud Computing and Service Models

Over the past two decades, the world economy has rapidly moved from manufacturing to more service-oriented. In 2010, 80 percent of the U.S. economy was driven by the service industry, leaving only 15 percent in manufacturing and 5 percent in agriculture and other areas. Cloud computing benefits the service industry most and advances business computing with a new paradigm. In 2009, the global cloud service marketplace reached

\$17.4 billion. IDC predicted in 2010 that the cloud-based economy may increase to \$44.2 billion by 2013. Developers of innovative cloud applications no longer acquire large capital equipment in advance. They just rent the resources from some large data centers that have been automated for this purpose.

In this and the next two chapters, we will study the cloud platform architecture, service models, and programming environments. Users can access and deploy cloud applications from anywhere in the world at very competitive costs. Virtualized cloud platforms are often built on top of large data centers. With that in mind, we examine first the server cluster in a data center and its interconnection issues. In other words, clouds aim to power the next generation of data centers by architecting them as virtual resources over automated hardware, databases, user interfaces, and application environments. In this sense, clouds grow out of the desire to build better data centers through automated resource provisioning.

## 4.1.1 Public, Private, and Hybrid Clouds

The concept of cloud computing has evolved from cluster, grid, and utility computing. Cluster and grid computing leverage the use of many computers in parallel to solve problems of any size. Utility and Software as a Service (SaaS) provide computing resources as a service with the notion of pay per use. Cloud computing leverages dynamic resources to deliver large numbers of services to end users. Cloud computing is a high-throughput computing (HTC) paradigm whereby the infrastructure provides the services through a large data center or server farms. The cloud computing model enables users to share access to resources from anywhere at any time through their connected devices.

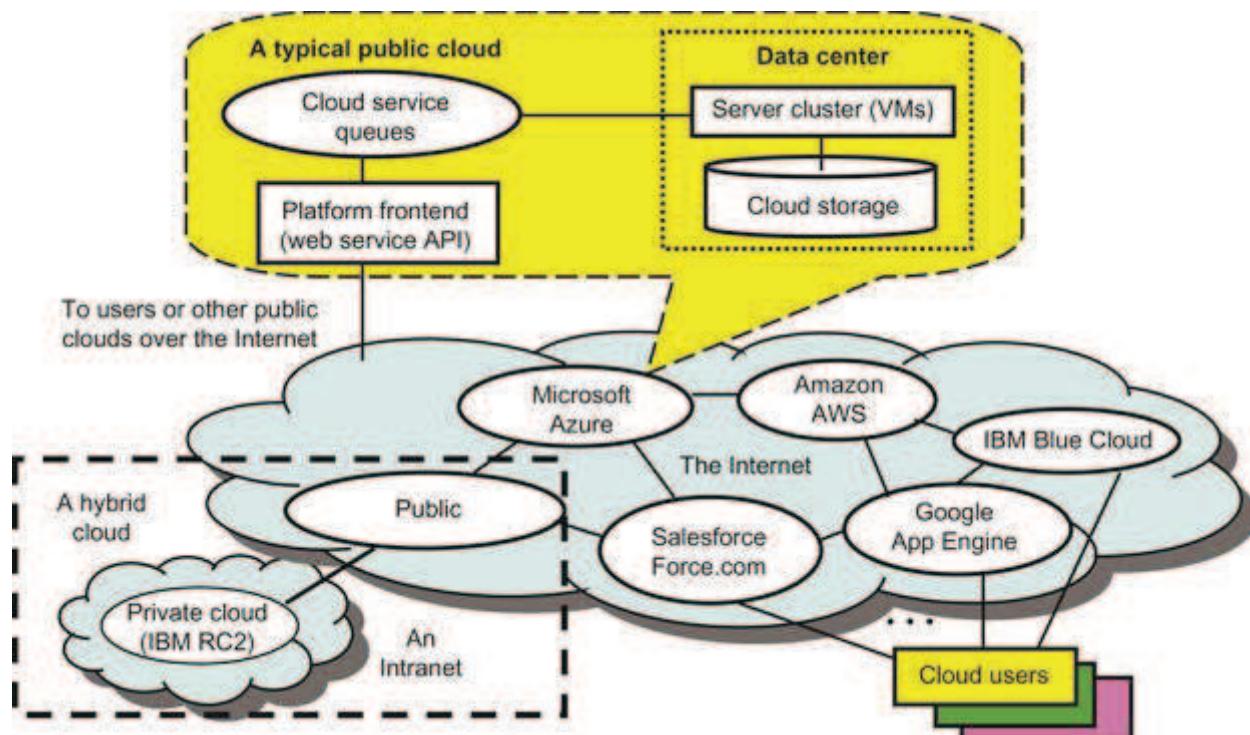
Recall the introduction in [Chapter 1](#) in which we said that the cloud will free users to focus on user application development and create business value by outsourcing job execution to cloud providers. In this scenario, the computations (programs) are sent to where the data is located, rather than copying the data to millions of desktops as in the traditional approach. Cloud computing avoids large data movement, resulting in much better network bandwidth utilization. Furthermore, machine virtualization has enhanced resource utilization, increased application flexibility, and reduced the total cost of using virtualized data-center resources.

The cloud offers significant benefit to IT companies by freeing them from the low-level task of setting up the hardware (servers) and managing the system software. Cloud computing applies a virtual platform with elastic resources put together by on-demand provisioning of hardware, software, and data sets, dynamically. The main idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers. Cloud computing leverages its low cost and simplicity to both providers and users. According to Ian Foster [\[25\]](#), cloud computing intends to leverage multitasking to achieve higher throughput by serving many heterogeneous applications, large or small, simultaneously.

#### 4.1.1.1 Centralized versus Distributed Computing

Some people argue that cloud computing is centralized computing at data centers. Others claim that cloud computing is the practice of distributed parallel computing over data-center resources. These represent two opposite views of cloud computing. All computations in cloud applications are distributed to servers in a data center. These are mainly *virtual machines* (VMs) in virtual clusters created out of data-center resources. In this sense, cloud platforms are systems distributed through virtualization.

As Figure 4.1 shows, both *public clouds* and *private clouds* are developed in the Internet. As many clouds are generated by commercial providers or by enterprises in a distributed manner, they will be interconnected over the Internet to achieve scalable and efficient computing services. Commercial cloud providers such as Amazon, Google, and Microsoft created their platforms to be distributed geographically. This distribution is partially attributed to fault tolerance, response latency reduction, and even legal reasons. Intranet-based private clouds are linked to public clouds to get additional resources. Nevertheless, users in Europe may not feel comfortable using clouds in the United States, and vice versa, until extensive *service-level agreements* (SLAs) are developed between the two user communities.



**FIGURE 4.1** Public, private, and hybrid clouds illustrated by functional architecture and connectivity of representative clouds available by 2011.

#### 4.1.1.2 Public Clouds

A *public cloud* is built over the Internet and can be accessed by any user who has paid for the service. Public clouds are owned by service providers and are accessible through a subscription. The callout box in top of Figure 4.1 shows the architecture of a typical

public cloud. Many public clouds are available, including Google App Engine (GAE), Amazon Web Services (AWS), Microsoft Azure, IBM Blue Cloud, and Salesforce.com's [Force.com](#). The providers of the aforementioned clouds are commercial providers that offer a publicly accessible remote interface for creating and managing VM instances within their proprietary infrastructure. A public cloud delivers a selected set of business processes. The application and infrastructure services are offered on a flexible price-per-use basis.

#### **4.1.1.3 Private Clouds**

A *private cloud* is built within the domain of an intranet owned by a single organization. Therefore, it is client owned and managed, and its access is limited to the owning clients and their partners. Its deployment was not meant to sell capacity over the Internet through publicly accessible interfaces. Private clouds give local users a flexible and agile private infrastructure to run service workloads within their administrative domains. A private cloud is supposed to deliver more efficient and convenient cloud services. It may impact the cloud standardization, while retaining greater customization and organizational control.

#### **4.1.1.4 Hybrid Clouds**

A *hybrid cloud* is built with both public and private clouds, as shown at the lower-left corner of [Figure 4.1](#). Private clouds can also support a hybrid cloud model by supplementing local infrastructure with computing capacity from an external public cloud. For example, the *Research Compute Cloud* (RC2) is a private cloud, built by IBM, that interconnects the computing and IT resources at eight IBM Research Centers scattered throughout the United States, Europe, and Asia. A hybrid cloud provides access to clients, the partner network, and third parties. In summary, public clouds promote standardization, preserve capital investment, and offer application flexibility. Private clouds attempt to achieve customization and offer higher efficiency, resiliency, security, and privacy. Hybrid clouds operate in the middle, with many compromises in terms of resource sharing.

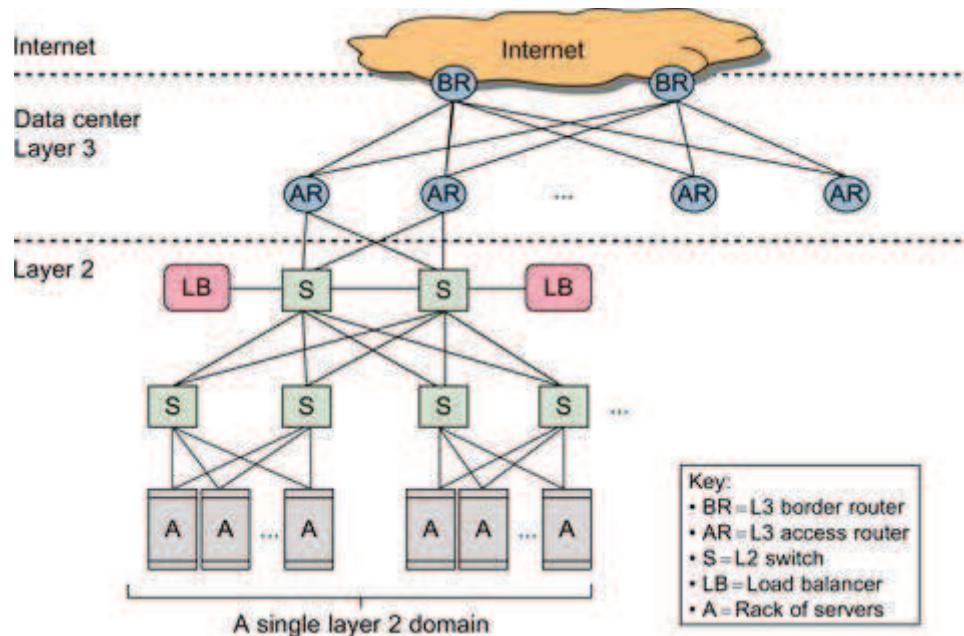
#### **4.1.1.5 Data-Center Networking Structure**

The core of a cloud is the server cluster (or VM cluster). Cluster nodes are used as compute nodes. A few control nodes are used to manage and monitor cloud activities. The scheduling of user jobs requires that you assign work to virtual clusters created for users. The gateway nodes provide the access points of the service from the outside world. These gateway nodes can be also used for security control of the entire cloud platform. In physical clusters and traditional grids, users expect static demand of resources. Clouds are designed to handle fluctuating workloads, and thus demand variable resources dynamically. Private clouds will satisfy this demand if properly designed and managed.

Data centers and supercomputers have some similarities as well as fundamental differences. We discussed supercomputers in [Chapter 2](#). In the case of data centers,

scaling is a fundamental requirement. Data-center server clusters are typically built with large number of servers, ranging from thousands to millions of servers (nodes). For example, Microsoft has a data center in the Chicago area that has 100,000 eight-core servers, housed in 50 containers. In supercomputers, a separate data farm is used, while a data center uses disks on server nodes plus memory cache and databases.

Data centers and supercomputers also differ in networking requirements, as illustrated in [Figure 4.2](#). Supercomputers use custom-designed high-bandwidth networks such as fat trees or 3D torus networks (which we discussed in [Chapter 2](#)). Data-center networks are mostly IP-based commodity networks, such as the 10 Gbps Ethernet network, which is optimized for Internet access. [Figure 4.2](#) shows a multilayer structure for accessing the Internet. The server racks are at the bottom Layer 2, and they are connected through fast switches (S) as the hardware core. The data center is connected to the Internet at Layer 3 with many *access routers* (ARs) and *border routers* (BRs).



**FIGURE 4.2** Standard data-center networking for the cloud to access the Internet Courtesy of Dennis Gannon, 2010 [26]

An example of a private cloud is the one the U.S. National Aeronautics and Space Administration (NASA) is building to enable researchers to run climate models on remote systems it provides. This can save users the capital expense of HPC machines at local sites. Furthermore, NASA can build the complex weather models around its data centers, which is more cost-effective. Another good example is the cloud built by the European Council for Nuclear Research (CERN). This is a very big private cloud designed to distribute data, applications, and computing resources to thousands of scientists around the world.

These cloud models demand different levels of performance, data protection, and security enforcement. In this case, different SLAs may be applied to satisfy both providers and paid users. Cloud computing exploits many existing technologies. For example, grid computing is the backbone of cloud computing in that the grid has the

same goals of resource sharing with better utilization of research facilities. Grids are more focused on delivering storage and computing resources while cloud computing aims to achieve economies of scale with abstracted services and resources.

#### 4.1.1.6 Cloud Development Trends

Although most clouds built in 2010 are large public clouds, the authors believe private clouds will grow much faster than public clouds in the future. Private clouds are easier to secure and more trustworthy within a company or organization. Once private clouds become mature and better secured, they could be open or converted to public clouds. Therefore, the boundary between public and private clouds could be blurred in the future. Most likely, most future clouds will be hybrid in nature.

For example, an e-mail application can run in the service-access nodes and provide the user interface for outside users; the application can get the service from the internal cloud computing services (e.g., the e-mail storage service). There are also some service nodes designed to support the proper functioning of cloud computing clusters. These nodes are called runtime supporting service nodes. For example, there might be distributed locking services for supporting specific applications. Finally, it is possible that there will be some independent service nodes. Those nodes would provide independent services for other nodes in the cluster. For example, a news service need geographical information under service-access nodes.

With cost-effective performance as the key concept of clouds, we will consider the public cloud in this chapter, unless otherwise specified. Many executable application codes are much smaller than the web-scale data sets they process. Cloud computing avoids large data movement during execution. This will result in less traffic on the Internet and better network utilization. Clouds also alleviate the petascale I/O problem. Cloud performance and its *Quality of Service* (QoS) are yet to be proven in more real-life applications. We will model the performance of cloud computing in [Chapter 9](#), along with data protection, security measures, service availability, fault tolerance, and operating cost.

#### 4.1.2 Cloud Ecosystem and Enabling Technologies

Cloud computing platforms differ from conventional computing platforms in many aspects. In this section, we will identify their differences in computing paradigms and cost models applied. The traditional computing model is specified below by the process on the left, which involves buying the hardware, acquiring the necessary system software, installing the system, testing the configuration, and executing the application code and management of resources. What is even worse is that this cycle repeats itself in about every 18 months, meaning the machine we bought becomes obsolete every 18 months.

The cloud computing paradigm is shown on the right. This computing model follows a pay-as-you-go model. Therefore the cost is significantly reduced, because we simply rent computer resources without buying the computer in advance. All hardware and software resources are leased from the cloud provider without capital investment on the part of

the users. Only the execution phase costs some money. The experts at IBM have estimated that an 80 percent to 95 percent saving results from cloud computing, compared with the conventional computing paradigm. This is very much desired, especially for small businesses, which requires limited computing power and thus avoid the purchase of expensive computers or servers repeatedly every few years.

<b>Classical Computing</b>	<b>Cloud Computing</b>
(Repeat the following cycle every 18 months)	(Pay as you go per each service provided)
<b>Buy and own</b>	<b>Subscribe</b>
Hardware, system software, applications to meet peak needs	- - - -
<b>Install, configure, test, verify, evaluate, manage</b>	<b>Use</b> (Save about 80-95% of the total cost)
- - - -	- - - -
<b>Use</b>	(Finally)
- - - -	<b>\$ - Pay for what you use</b>
<b>Pay \$\$\$\$\$ (High cost)</b>	based on the QoS

For example, IBM has estimated that the worldwide cloud service market may reach \$126 billion by 2012, including components, infrastructure services, and business services. Internet clouds work as service factories built around multiple data centers. To formalize the above cloud computing model, we characterize the cloud cost model, the cloud ecosystems, and enabling technologies. These topics help our readers understand the motivations behind cloud computing. The intention is to remove the barriers of cloud computing

#### **4.1.2.1 Cloud Design Objectives**

Despite the controversy surrounding the replacement of desktop or deskside computing by centralized computing and storage services at data centers or big IT companies, the cloud computing community has reached some consensus on what has to be done to make cloud computing universally acceptable. The following list highlights six design objectives for cloud computing:

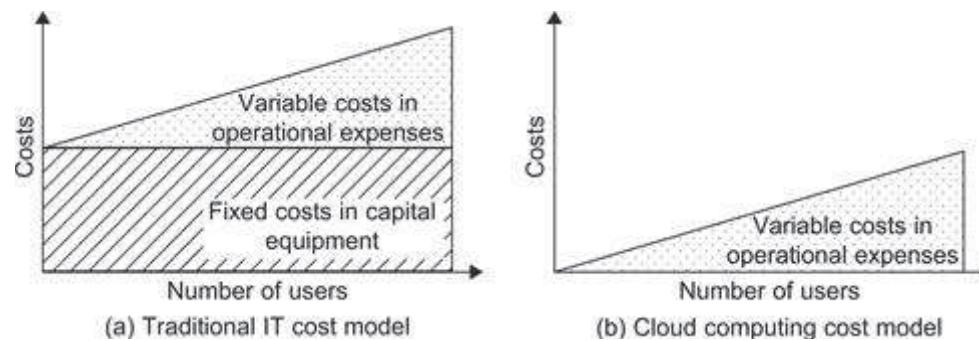
- **Shifting computing from desktops to data centers** Computer processing, storage, and software delivery is shifted away from desktops and local servers and toward data centers over the Internet.
- **Service provisioning and cloud economics** Providers supply cloud services by signing SLAs with consumers and end users. The services must be efficient in terms of computing, storage, and power consumption. Pricing is based on a pay-as-you-go policy.
- **Scalability in performance** The cloud platforms and software and infrastructure services must be able to scale in performance as the number of users increases.
- **Data privacy protection** Can you trust data centers to handle your private data and records? This concern must be addressed to make clouds successful as trusted services.
- **High quality of cloud services** The QoS of cloud computing must be standardized to

make clouds interoperable among multiple providers.

- **New standards and interfaces** This refers to solving the data lock-in problem associated with data centers or cloud providers. Universally accepted APIs and access protocols are needed to provide high portability and flexibility of virtualized applications.

#### 4.1.2.2 Cost Model

In traditional IT computing, users must acquire their own computer and peripheral equipment as capital expenses. In addition, they have to face operational expenditures in operating and maintaining the computer systems, including personnel and service costs. Figure 4.3(a) shows the addition of variable operational costs on top of fixed capital investments in traditional IT. Note that the fixed cost is the main cost, and that it could be reduced slightly as the number of users increases. However, the operational costs may increase sharply with a larger number of users. Therefore, the total cost escalates quickly with massive numbers of users. On the other hand, cloud computing applies a pay-per-use business model, in which user jobs are outsourced to data centers. To use the cloud, one has no up-front cost in hardware acquisitions. Only variable costs are experienced by cloud users, as demonstrated in Figure 4.3(b).



**FIGURE 4.3** Computing economics between traditional IT users and cloud users.

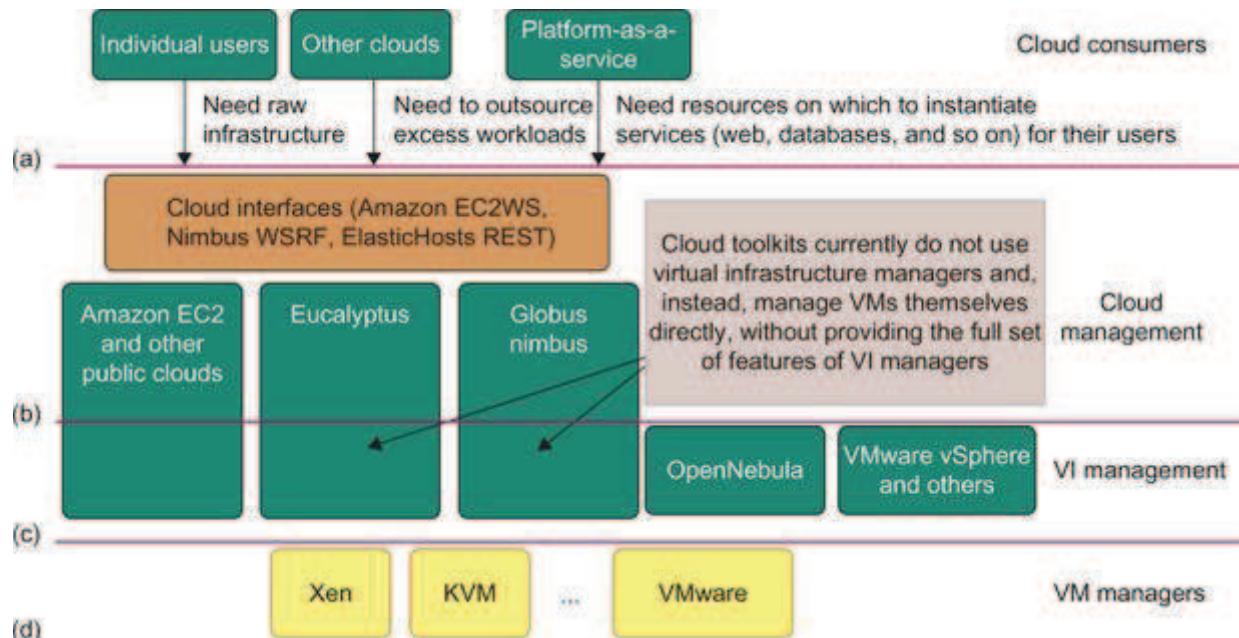
Overall, cloud computing will reduce computing costs significantly for both small users and large enterprises. Computing economics does show a big gap between traditional IT users and cloud users. The savings in acquiring expensive computers up front releases a lot of burden for startup companies. The fact that cloud users only pay for operational expenses and do not have to invest in permanent equipment is especially attractive to massive numbers of small users. This is a major driving force for cloud computing to become appealing to most enterprises and heavy computer users. In fact, any IT users whose capital expenses are under more pressure than their operational expenses should consider sending their overflow work to utility computing or cloud service providers.

#### 4.1.2.3 Cloud Ecosystems

With the emergence of various Internet clouds, an ecosystem of providers, users, and

technologies has appeared. This ecosystem has evolved around public clouds. Strong interest is growing in open source cloud computing tools that let organizations build their own IaaS clouds using their internal infrastructures. Private and hybrid clouds are not exclusive, since public clouds are involved in both cloud types. A private/hybrid cloud allows remote access to its resources over the Internet using remote web service interfaces such as that used in Amazon EC2.

An ecosystem was suggested by Sotomayor, et al. [39] (Figure 4.4) for building private clouds. They suggested four levels of ecosystem development in a private cloud. At the user end, consumers demand a flexible platform. At the cloud management level, the cloud manager provides virtualized resources over an IaaS platform. At the virtual infrastructure (VI) management level, the manager allocates VMs over multiple server clusters. Finally, at the VM management level, the VM managers handle VMs installed on individual host machines. An ecosystem of cloud tools attempts to span both cloud management and VI management. Integrating these two layers is complicated by the lack of open and standard interfaces between them.



**FIGURE 4.4** Cloud ecosystem for building private clouds: (a) Consumers demand a flexible platform; (b) Cloud manager provides virtualized resources over an IaaS platform; (c) VI manager allocates VMs; (d) VM managers handle VMs installed on servers Courtesy of Sotomayor, et al. © IEEE [68]

An increasing number of startup companies are now basing their IT strategies on cloud resources, spending little or no capital to manage their own IT infrastructures. We desire a flexible and open architecture that enables organizations to build private/hybrid clouds. VI management is aimed at this goal. Example VI tools include oVirt (<https://fedorahosted.org/ovirt/>), vSphere/4 ([www.vmware.com/products/vsphere/](http://www.vmware.com/products/vsphere/)) from VMWare, and VM Orchestrator ([www.platform.com/Products/platform-vm-orchestrator](http://www.platform.com/Products/platform-vm-orchestrator)) from Platfom Computing.

These tools support dynamic placement and VM management on a pool of physical resources, automatic load balancing, server consolidation, and dynamic infrastructure

resizing and partitioning. In addition to public clouds such as Amazon EC2, Eucalyptus and Globus Nimbus are open source tools for virtualization of cloud infrastructure. To access these cloud management tools, one can use the Amazon EC2WS, Nimbus WSRF, and ElasticHost REST cloud interfaces. For VI management, OpenNebula and VMware vSphere can be used to manage all VM generation including Xen, KVM, and VMware tools.

#### 4.1.2.4 Surge of Private Clouds

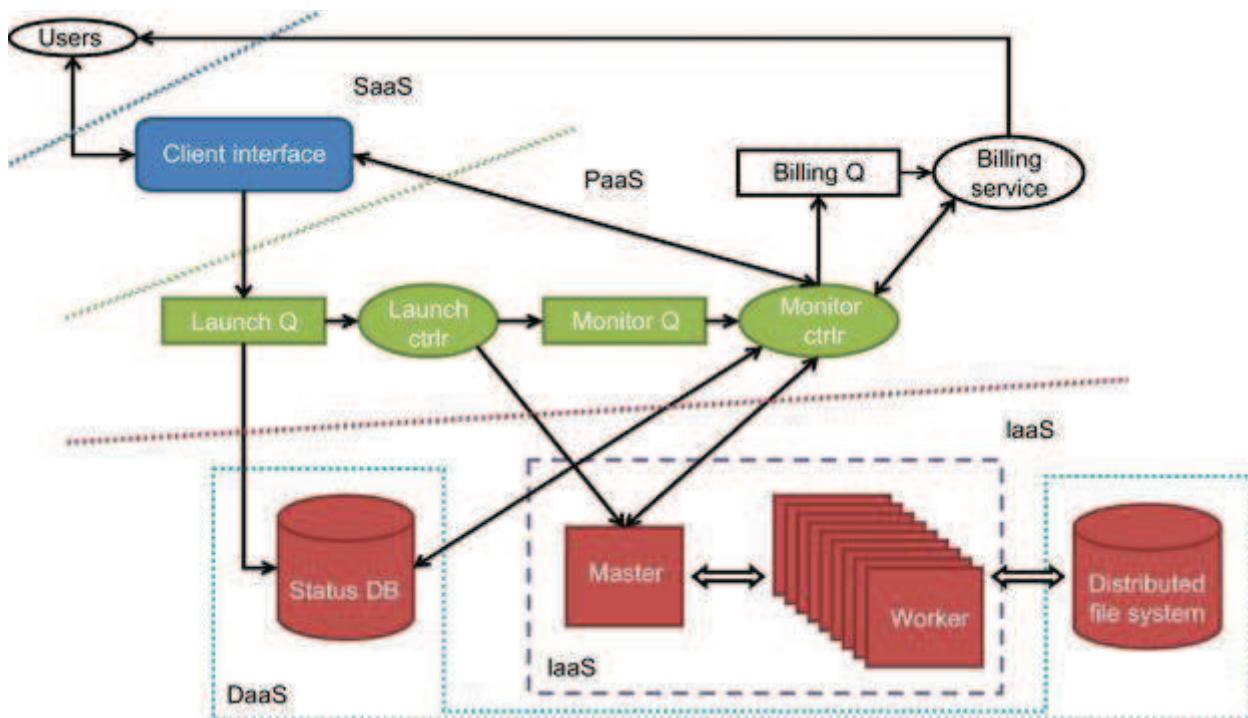
In general, private clouds leverage existing IT infrastructure and personnel within an enterprise or government organization. Both public and private clouds handle workloads dynamically. However, public clouds should be designed to handle workloads without communication dependency. Both types of clouds distribute data and VM resources. However, private clouds can balance workloads to exploit IT resources more efficiently within the same intranet. Private clouds can also provide preproduction testing and enforce data privacy and security policies more effectively. In a public cloud, the surge workload is often offloaded. The major advantage of public clouds lies in the avoidance of capital expenses by users in IT investments in hardware, software, and personnel.

Most companies start with virtualization of their computing machines to lower the operating costs. Companies such as Microsoft, Oracle, and SAP may want to establish policy-driven management of their computing resources, mainly to improve QoS to their employees and customers. By integrating virtualized data centers and company IT resources, they offer *IT as a service* to improve the agility of their company operations. This approach avoids replacement of a large number of servers every 18 months. As a result, these companies can upgrade their IT efficiency significantly.

### 4.1.3 Infrastructure-as-a-Service (IaaS)

Cloud computing delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. The services provided over the cloud can be generally categorized into three different service models: namely IaaS, Platform as a Service (PaaS), and Software as a Service (SaaS). These form the three pillars on top of which cloud computing solutions are delivered to end users. All three models allow users to access services over the Internet, relying entirely on the infrastructures of cloud service providers.

These models are offered based on various SLAs between providers and users. In a broad sense, the SLA for cloud computing is addressed in terms of service availability, performance, and data protection and security. [Figure 4.5](#) illustrates three cloud models at different service levels of the cloud. SaaS is applied at the application end using special interfaces by users or clients. At the PaaS layer, the cloud platform must perform billing services and handle job queuing, launching, and monitoring services. At the bottom layer of the IaaS services, databases, compute instances, the file system, and storage must be provisioned to satisfy user demands.



**FIGURE 4.5** The IaaS, PaaS, and SaaS cloud service models at different service levels. Courtesy of J. Suh and S. Kang, USC

#### 4.1.3.1 Infrastructure as a Service

This model allows users to use virtualized IT resources for computing, storage, and networking. In short, the service is performed by rented cloud infrastructure. The user can deploy and run his applications over his chosen OS environment. The user does not manage or control the underlying cloud infrastructure, but has control over the OS, storage, deployed applications, and possibly select networking components. This IaaS model encompasses *storage as a service*, *compute instances as a service*, and *communication as a service*. The *Virtual Private Cloud (VPC)* in [Example 4.1](#) shows how to provide Amazon EC2 clusters and S3 storage to multiple users. Many startup cloud providers have appeared in recent years. GoGrid, FlexiScale, and Aneka are good examples. [Table 4.1](#) summarizes the IaaS offerings by five public cloud providers. Interested readers can visit the companies' web sites for updated information. More examples can be also found in two recent cloud books [10,18].

**Table 4.1**

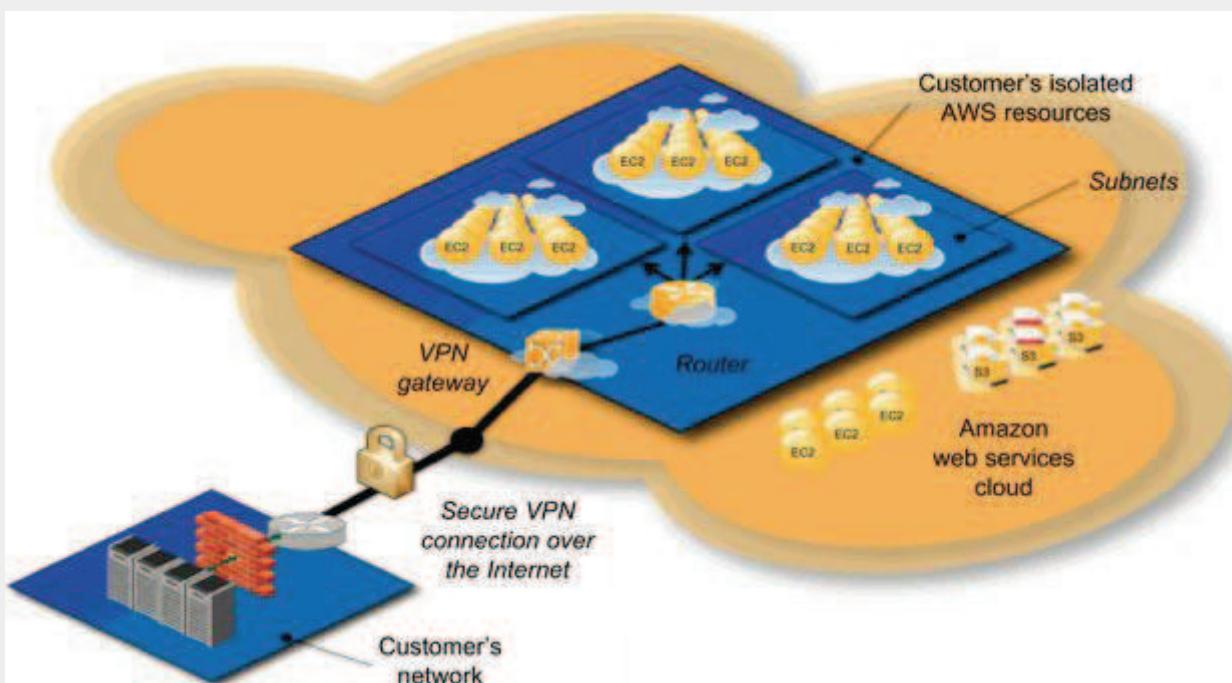
**Public Cloud Offerings of IaaS [10,18]**

Cloud Name	VM Instance Capacity	API and Access Tools	Hypervisor, Guest OS
<b>Amazon EC2</b>	Each instance has 1–20 EC2 processors, 1.7–15 GB of memory, and 160–1.69 TB of storage.	CLI or web Service (WS) portal	Xen, Linux, Windows
<b>GoGrid</b>	Each instance has 1–6 CPUs, 0.5–8 GB of memory, and 30–480 GB of storage.	REST, Java, PHP, Python, Ruby	Xen, Linux, Windows
<b>Rackspace Cloud</b>	Each instance has a four-core CPU, 0.25–16 GB of memory, and 10–620 GB of storage.	REST, Python, PHP, Java, C#, .NET	Xen, Linux
<b>FlexiScale in the UK</b>	Each instance has 1–4 CPUs, 0.5–16 GB of memory, and 20–270 GB of storage.	web console	Xen, Linux, Windows
<b>Joyent Cloud</b>	Each instance has up to eight CPUs, 0.25–32 GB of memory, and 30–480 GB of storage.	No specific API, SSH, Virtual/Mon	OS-level virtualization, OpenSolaris

## Example 4.1

### Amazon VPC for Multiple Tenants

A user can use a private facility for basic computations. When he must meet a specific workload requirement, he can use the Amazon VPC to provide additional EC2 instances or more storage (S3) to handle urgent applications. Figure 4.6 shows VPC which is essentially a private cloud designed to address the privacy concerns of public clouds that hamper their application when sensitive data and software are involved.



**FIGURE 4.6** Amazon VPC (virtual private cloud) Courtesy of VMWare, <http://aws.amazon.com/vpc/>

Amazon EC2 provides the following services: resources from multiple data centers globally distributed, CL1, web services (SOAP and Query), web-based console user interfaces, access to VM instances via SSH and Windows, 99.5 percent available agreements, per-hour pricing, Linux and Windows OSes, and automatic scaling and load balancing. We will illustrate the use of EC2 in more detail in [Chapter 6](#). VPC allows the user to isolate provisioned AWS processors, memory, and storage from interference by other users. Both *auto-scaling* and *elastic load balancing* services can support related demands. Auto-scaling enables users to automatically scale their VM instance capacity up or down. With auto-scaling, one can ensure that a sufficient number of Amazon EC2 instances are provisioned to meet desired performance. Or one can scale down the VM instance capacity to reduce costs, when the workload is reduced.

## 4.1.4 Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS)

In this section, we will introduce the PaaS and SaaS models for cloud computing. SaaS is often built on top of the PaaS, which is in turn built on top of the IaaS.

### 4.1.4.1 Platform as a Service (PaaS)

To be able to develop, deploy, and manage the execution of applications using provisioned resources demands a cloud platform with the proper software environment. Such a platform includes operating system and runtime library support. This has triggered the creation of the PaaS model to enable users to develop and deploy their user applications. [Table 4.2](#) highlights cloud platform services offered by five PaaS services. Further details of some of these PaaS offerings are provided in [Section 4.4](#) and in [Chapter 6](#). Some illustrative examples and case studies can be found in [\[10,18\]](#).

---

**Table 4.2**

**Five Public Cloud Offerings of PaaS [10,18]**

---

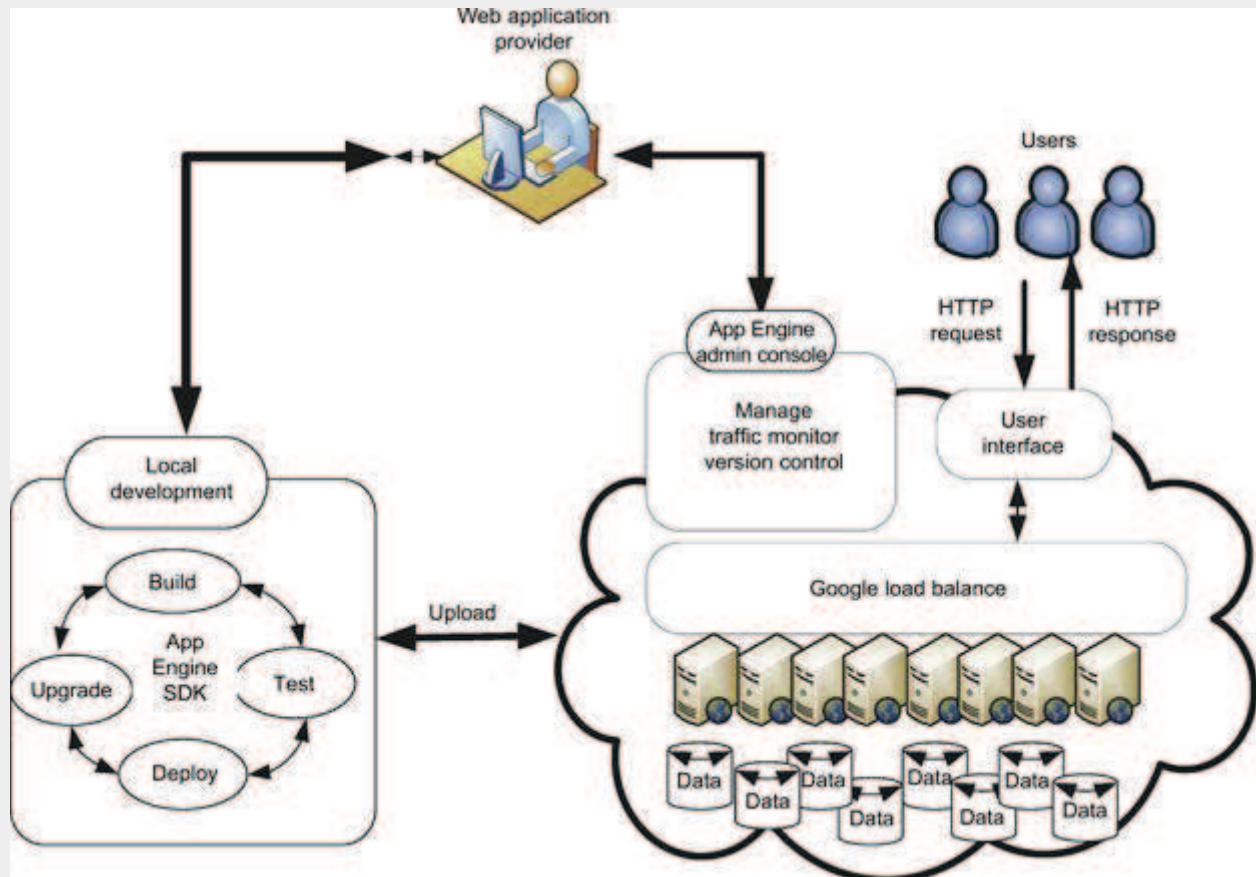
Cloud Name	Languages and Developer Tools	Programming Models Supported by Provider	Target Applications and Storage Option
Google App Engine	Python, Java, and Eclipse-based IDE	MapReduce, web programming on demand	Web applications and BigTable storage
Salesforce.com's Force.com	Apex, Eclipse-based IDE, web-based Wizard	Workflow, Excel-like formula, Web programming on demand	Business applications such as CRM
Microsoft Azure	.NET, Azure tools for MS Visual Studio	Unrestricted model	Enterprise and web applications
Amazon Elastic MapReduce	Hive, Pig, Cascading, Java, Ruby, Perl, Python, PHP, R, C++	MapReduce	Data processing and e-commerce
Aneka	.NET, stand-alone SDK	Threads, task, MapReduce	.NET enterprise applications, HPC

The platform cloud is an integrated computer system consisting of both hardware and software infrastructure. The user application can be developed on this virtualized cloud platform using some programming languages and software tools supported by the provider (e.g., Java, Python, .NET). The user does not manage the underlying cloud infrastructure. The cloud provider supports user application development and testing on a well-defined service platform. This PaaS model enables a collaborated software development platform for users from different parts of the world. This model also encourages third parties to provide software management, integration, and service monitoring solutions.

## Example 4.2

### Google App Engine for PaaS Applications

As web applications are running on Google's server clusters, they share the same capability with many other users. The applications have features such as automatic scaling and load balancing which are very convenient while building web applications. The distributed scheduler mechanism can also schedule tasks for triggering events at specified times and regular intervals. Figure 4.7 shows the operational model for GAE. To develop applications using GAE, a development environment must be provided.



**FIGURE 4.7** Google App Engine platform for PaaS operations Courtesy of Yangting Wu, USC

Google provides a fully featured local development environment that simulates GAE on the developer's computer. All the functions and application logic can be implemented locally which is quite similar to traditional software development. The coding and debugging stages can be performed locally as well. After these steps are finished, the SDK provided provides a tool for uploading the user's application to Google's infrastructure where the applications are actually deployed. Many additional third-party capabilities, including software management, integration, and service monitoring solutions, are also provided.

Here are some useful links when logging on to the GAE system:

- Google App Engine home page: <http://code.google.com/appengine/>
- Sign up for an account or use your Gmail account name:  
<https://appengine.google.com/>
- Download GAE SDK: <http://code.google.com/appengine/downloads.html>
- Python Getting Started Guide:  
<http://code.google.com/appengine/docs/python/gettingstarted/>
- Java Getting Started Guide:  
<http://code.google.com/appengine/docs/java/gettingstarted/>
- Quota page for free service:  
<http://code.google.com/appengine/docs/quotas.html#Resources>
- Billing page if you go over the quota:  
<http://code.google.com/appengine/docs/quotas.html#Billing>

#### 4.1.4.2 Software as a Service (SaaS)

This refers to browser-initiated application software over thousands of cloud customers. Services and tools offered by PaaS are utilized in construction of applications and management of their deployment on resources offered by IaaS providers. The SaaS model provides software applications as a service. As a result, on the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are kept rather low, compared with conventional hosting of user applications. Customer data is stored in the cloud that is either vendor proprietary or publicly hosted to support PaaS and IaaS.

The best examples of SaaS services include Google Gmail and docs, Microsoft SharePoint, and the CRM software from Salesforce.com. They are all very successful in promoting their own business or are used by thousands of small businesses in their day-to-day operations. Providers such as Google and Microsoft offer integrated IaaS and PaaS services, whereas others such as Amazon and GoGrid offer pure IaaS services and expect third-party PaaS providers such as Manjrasoft to offer application development and deployment services on top of their infrastructure services. To identify important cloud applications in enterprises, the success stories of three real-life cloud applications are presented in [Example 4.3](#) for HTC, news media, and business transactions. The benefits of using cloud services are evident in these SaaS applications.

#### **Example 4.3**

#### **Three Success Stories on SaaS Applications**

1. To discover new drugs through DNA sequence analysis, Eli Lilly Company has used Amazon's AWS platform with provisioned server and storage clusters to conduct high-performance biological sequence analysis without using an expensive supercomputer. The benefit of this IaaS application is reduced drug deployment time with much lower costs.
2. The *New York Times* has applied Amazon's EC2 and S3 services to retrieve useful pictorial information quickly from millions of archival articles and newspapers. The *New York Times* has significantly reduced the time and cost in getting the job done.
3. Pitney Bowes, an e-commerce company, offers clients the opportunity to perform B2B transactions using the Microsoft Azure platform, along with .NET and SQL services. These offerings have significantly increased

the company's client base.

#### 4.1.4.3 Mashup of Cloud Services

At the time of this writing, public clouds are in use by a growing number of users. Due to the lack of trust in leaking sensitive data in the business world, more and more enterprises, organizations, and communities are developing private clouds that demand deep customization. An enterprise cloud is used by multiple users within an organization. Each user may build some strategic applications on the cloud, and demands customized partitioning of the data, logic, and database in the metadata representation. More private clouds may appear in the future.

Based on a 2010 Google search survey, interest in grid computing is declining rapidly. *Cloud mashups* have resulted from the need to use multiple clouds simultaneously or in sequence. For example, an industrial supply chain may involve the use of different cloud resources or services at different stages of the chain. Some public repository provides thousands of service APIs and mashups for web commerce services. Popular APIs are provided by Google Maps, Twitter, YouTube, Amazon eCommerce, Salesforce.com, etc.

## 4.2 Data-Center Design and Interconnection Networks

A data center is often built with a large number of servers through a huge interconnection network. In this section, we will study the design of large-scale data centers and small modular data centers that can be housed in a 40-ft truck container. Then we will take a look at interconnection of modular data centers and their management issues and solutions.

### 4.2.1 Warehouse-Scale Data-Center Design

Dennis Gannon claims: "The cloud is built on massive datacenters" [26]. Figure 4.8 shows a data center that is as large as a shopping mall (11 times the size of a football field) under one roof. Such a data center can house 400,000 to 1 million servers. The data centers are built economics of scale—meaning lower unit cost for larger data centers. A small data center could have 1,000 servers. The larger the data center, the lower the operational cost. The approximate monthly cost to operate a huge 400-server data center is estimated by network cost \$13/Mbps; storage cost \$0.4/GB; and administration costs. These unit costs are greater than those of a 1,000-server data center. The network cost to operate a small data center is about seven times greater and the storage cost is 5.7 times greater. Microsoft has about 100 data centers, large or small, which are distributed around the globe.



**FIGURE 4.8** A huge data center that is 11 times the size of a football field, housing 400,000 to 1 million servers  
Courtesy of Dennis Gannon [26]

#### 4.2.1.1 Data-Center Construction Requirements

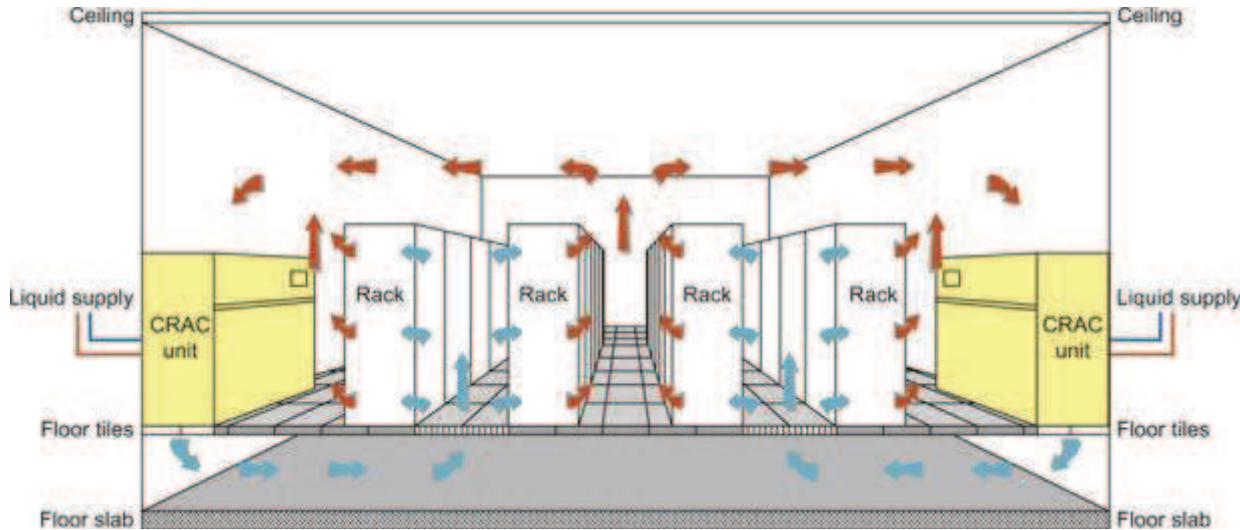
Most data centers are built with commercially available components. An off-the-shelf server consists of a number of processor sockets, each with a multicore CPU and its internal cache hierarchy, local shared and coherent DRAM, and a number of directly attached disk drives. The DRAM and disk resources within the rack are accessible through first-level rack switches and all resources in all racks are accessible via a cluster-level switch. Consider a data center built with 2,000 servers, each with 8 GB of DRAM and four 1 TB disk drives. Each group of 40 servers is connected through a 1 Gbps link to a rack-level switch that has an additional eight 1 Gbps ports used for connecting the rack to the cluster-level switch.

It was estimated [9] that the bandwidth available from local disks is 200 MB/s, whereas the bandwidth from off-rack disks is 25 MB/s via shared rack uplinks. The total disk storage in the cluster is almost 10 million times larger than local DRAM. A large application must deal with large discrepancies in latency, bandwidth, and capacity. In a very large-scale data center, components are relatively cheaper. The components used in data centers are very different from those in building supercomputer systems.

With a scale of thousands of servers, concurrent failure, either hardware failure or software failure, of 1 percent of nodes is common. Many failures can happen in hardware; for example, CPU failure, disk I/O failure, and network failure. It is even quite possible that the whole data center does not work in the case of a power crash. Also, some failures are brought on by software. The service and data should not be lost in a failure situation. Reliability can be achieved by redundant hardware. The software must keep multiple copies of data in different locations and keep the data accessible while facing hardware or software errors.

#### 4.2.1.2 Cooling System of a Data-Center Room

**Figure 4.9** shows the layout and cooling facility of a warehouse in a data center. The data-center room has raised floors for hiding cables, power lines, and cooling supplies. The cooling system is somewhat simpler than the power system. The raised floor has a steel grid resting on stanchions about 2–4 ft above the concrete floor. The under-floor area is often used to route power cables to racks, but its primary use is to distribute cool air to the server rack. The CRAC (*computer room air conditioning*) unit pressurizes the raised floor plenum by blowing cold air into the plenum.



**FIGURE 4.9** The cooling system in a raised-floor data center with hot-cold air circulation supporting water heat exchange facilities Courtesy of DLB Associates, D. Dyer [22]

The cold air escapes from the plenum through perforated tiles that are placed in front of server racks. Racks are arranged in long aisles that alternate between cold aisles and hot aisles to avoid mixing hot and cold air. The hot air produced by the servers circulates back to the intakes of the CRAC units that cool it and then exhaust the cool air into the raised floor plenum again. Typically, the incoming coolant is at 12–14°C and the warm coolant returns to a chiller. Newer data centers often insert a cooling tower to pre-cool the condenser water loop fluid. Water-based free cooling uses cooling towers to dissipate heat. The cooling towers use a separate cooling loop in which water absorbs the coolant's heat in a heat exchanger.

## 4.2.2 Data-Center Interconnection Networks

A critical core design of a data center is the interconnection network among all servers in the data-center cluster. This network design must meet five special requirements: low latency, high bandwidth, low cost, *message-passing interface* (MPI) communication support, and fault tolerance. The design of an inter-server network must satisfy both point-to-point and collective communication patterns among all server nodes. Specific design considerations are given in the following sections.

### 4.2.2.1 Application Traffic Support

The network topology should support all MPI communication patterns. Both point-to-point and collective MPI communications must be supported. The network should have high bisection bandwidth to meet this requirement. For example, one-to-many communications are used for supporting distributed file access. One can use one or a few servers as metadata master servers which need to communicate with slave server nodes in the cluster. To support the MapReduce programming paradigm, the network must be designed to perform the map and reduce functions (to be treated in [Chapter 7](#)) at a high speed. In other words, the underlying network structure should support various network traffic patterns demanded by user applications.

### **4.2.2.2 Network Expandability**

The interconnection network should be expandable. With thousands or even hundreds of thousands of server nodes, the cluster network interconnection should be allowed to expand once more servers are added to the data center. The network topology should be restructured while facing such expected growth in the future. Also, the network should be designed to support load balancing and data movement among the servers. None of the links should become a bottleneck that slows down application performance. The topology of the interconnection should avoid such bottlenecks.

The fat-tree and crossbar networks studied in [Chapter 2](#) could be implemented with low-cost Ethernet switches. However, the design could be very challenging when the number of servers increases sharply. The most critical issue regarding expandability is support of modular network growth for building data-center containers, as discussed in [Section 4.2.3](#). One single data-center container contains hundreds of servers and is considered to be the building block of large-scale data centers. The network interconnection among many containers will be explained in [Section 4.2.4](#). Cluster networks need to be designed for data-center containers. Cable connections are then needed among multiple data-center containers.

Data centers are not built by piling up servers in multiple racks today. Instead, data-center owners buy server containers while each container contains several hundred or even thousands of server nodes. The owners can just plug in the power supply, outside connection link, and cooling water, and the whole system will just work. This is quite efficient and reduces the cost of purchasing and maintaining servers. One approach is to establish the connection backbone first and then extend the backbone links to reach the end servers. One can also connect multiple containers through external switching and cabling.

### **4.2.2.3 Fault Tolerance and Graceful Degradation**

The interconnection network should provide some mechanism to tolerate link or switch failures. In addition, multiple paths should be established between any two server nodes in a data center. Fault tolerance of servers is achieved by replicating data and computing among redundant servers. Similar redundancy technology should apply to the network structure. Both software and hardware network redundancy apply to cope with potential

failures. On the software side, the software layer should be aware of network failures. Packet forwarding should avoid using broken links. The network support software drivers should handle this transparently without affecting cloud operations.

In case of failures, the network structure should degrade gracefully amid limited node failures. Hot-swappable components are desired. There should be no critical paths or critical points which may become a single point of failure that pulls down the entire system. Most design innovations are in the topology structure of the network. The network structure is often divided into two layers. The lower layer is close to the end servers, and the upper layer establishes the backbone connections among the server groups or sub-clusters. This hierarchical interconnection approach appeals to building data centers with modular containers.

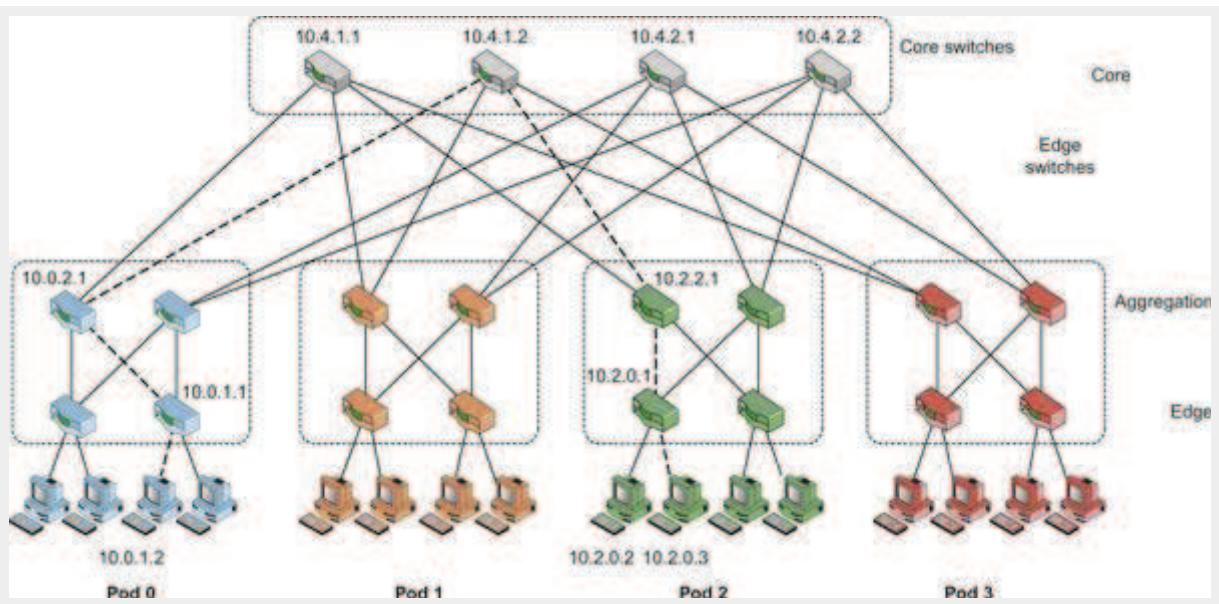
#### **4.2.2.4 Switch-centric Data-Center Design**

At the time of this writing, there are two approaches to building data-center-scale networks: One is switch-centric and the other is server-centric. In a switch-centric network, the switches are used to connect the server nodes. The switch-centric design does not affect the server side. No modifications to the servers are needed. The server-centric design does modify the operating system running on the servers. Special drivers are designed for relaying the traffic. Switches still have to be organized to achieve the connections.

#### **Example 4.4**

#### **A Fat-Tree Interconnection Network for Data Centers**

Figure 4.10 shows a fat-tree switch network design for data-center construction. The fat-tree topology is applied to interconnect the server nodes. The topology is organized into two layers. Server nodes are in the bottom layer, and *edge switches* are used to connect the nodes in the bottom layer. The upper layer aggregates the lower-layer edge switches. A group of aggregation switches, edge switches, and their leaf nodes form a *pod*. *Core switches* provide paths among different pods. The fat-tree structure provides multiple paths between any two server nodes. This provides fault-tolerant capability with an alternate path in case of some isolated link failures.



**FIGURE 4.10** A fat-tree interconnection topology for scalable data-center construction Courtesy of M. Al-Fares, et al. [2]

The failure of an aggregation switch and core switch will not affect the connectivity of the whole network. The failure of any edge switch can only affect a small number of end server nodes. The extra switches in a pod provide higher bandwidth to support cloud applications in massive data movement. The building blocks used are the low-cost Ethernet switches. This reduces the cost quite a bit. The routing table provides extra routing paths in case of failure. The routing algorithms are built inside the switches. The end server nodes in the data center are not affected during a switch failure, as long as the alternate routing path does not fail at the same time.

### 4.2.3 Modular Data Center in Shipping Containers

A modern data center is structured as a shipyard of server clusters housed in truck-towed containers. Figure 4.11 shows the housing of multiple sever racks in a truck-towed container in the SGI ICE Cube modular data center. Inside the container, hundreds of blade servers are housed in racks surrounding the container walls. An array of fans forces the heated air generated by the server racks to go through a heat exchanger, which cools the air for the next rack (detail in callout) on a continuous loop. The SGI ICE Cube container can house 46,080 processing cores or 30 PB of storage per container.





**FIGURE 4.11** A modular data center built in a truck-towed ICE Cube container, that can be cooled by chilled air circulation with cold-water heat exchanges Courtesy of SGI, Inc., <http://www.sgi.com/icecube>

Large-scale data center built with modular containers appear as a big shipping yard of container trucks. This container-based data center was motivated by demand for lower power consumption, higher computer density, and mobility to relocate data centers to better locations with lower electricity costs, better cooling water supplies, and cheaper housing for maintenance engineers. Sophisticated cooling technology enables up to 80% reduction in cooling costs compared with traditional warehouse data centers. Both chilled air circulation and cold water are flowing through the heat exchange pipes to keep the server racks cool and easy to repair.

Data centers usually are built at a site where leases and utilities for electricity are cheaper, and cooling is more efficient. Both warehouse-scale and modular data centers in containers are needed. In fact, the modular truck containers can be used to put together a large-scale data center like a container shipping yard. In addition to location selection and power savings in data-center operations, one must consider data integrity, server monitoring, and security management in data centers. These problems are easier to handle if the data center is centralized in a single large building.

#### **4.2.3.1 Container Data-Center Construction**

The data-center module is housed in a truck-towable container. The modular container design includes the network, computer, storage, and cooling gear. One needs to increase cooling efficiency by varying the water and airflow with better airflow management. Another concern is to meet seasonal load requirements. The construction of a container-based data center may start with one system (server), then move to a rack system design, and finally to a container system. This staged development may take different amounts of time and demand increasing costs. Building a rack of 40 servers may take half a day. Extending this to a whole container system with multiple racks for 1,000 servers requires the layout of the floor space with power, networking, cooling, and complete testing.

The container must be designed to be weatherproof and easy to transport. Modular data-center construction and testing may take a few days to complete if all components are available and power and water supplies are handy. The modular data-center approach supports many cloud service applications. For example, the health care industry will benefit by installing a data center at all clinic sites. However, how to exchange information with the central database and maintain periodic consistency becomes a rather challenging design issue in a hierarchically structured data center. The security of collocation cloud services may involve multiple data centers.

#### **4.2.4 Interconnection of Modular Data Centers**

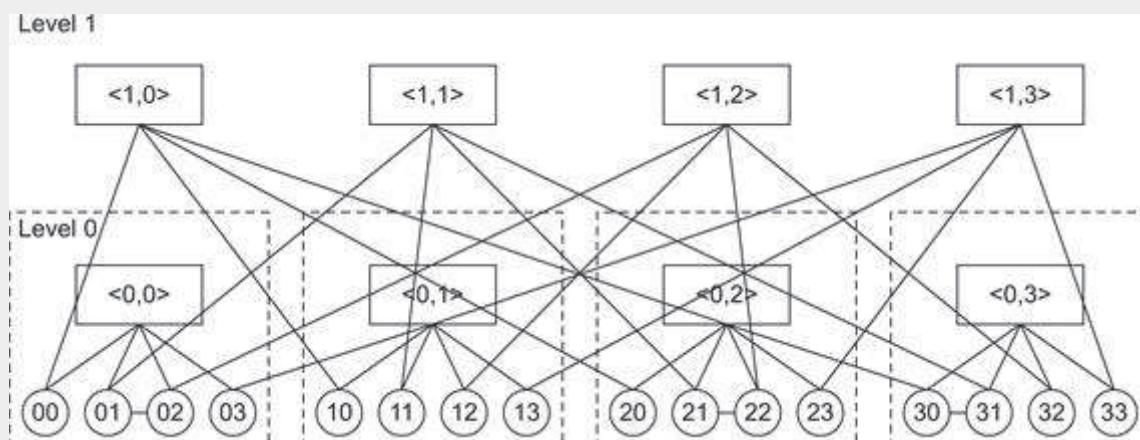
Container-based data-center modules are meant for construction of even larger data centers using a farm of container modules. Some proposed designs of container modules

are presented in this section. Their interconnections are shown for building scalable data centers. The following example is a server-centric design of the data-center module.

### Example 4.5

#### A Server-Centric Network for a Modular Data Center

Guo, et al. [30] have developed a server-centric BCube network (Figure 4.12) for interconnecting modular data centers. The servers are represented by circles, and switches by rectangles. The BCube provides a layered structure. The bottom layer contains all the server nodes and they form Level 0. Level 1 switches form the top layer of  $\text{BCube}_0$ . BCube is a recursively constructed structure. The  $\text{BCube}_0$  consists of  $n$  servers connecting to an  $n$ -port switch. The  $\text{BCube}_k$  ( $k \geq 1$ ) is structured from  $n$   $\text{BCube}_{k-1}$  with  $n^k$   $n$ -port switches. The example of  $\text{BCube}_1$  is illustrated in Figure 4.12, where the connection rule is that the  $i$ -th server in the  $j$ -th  $\text{BCube}_0$  connects to the  $j$ -th port of the  $i$ -th Level 1 switch. The servers in the BCube have multiple ports attached. This allows extra devices to be used in the server.

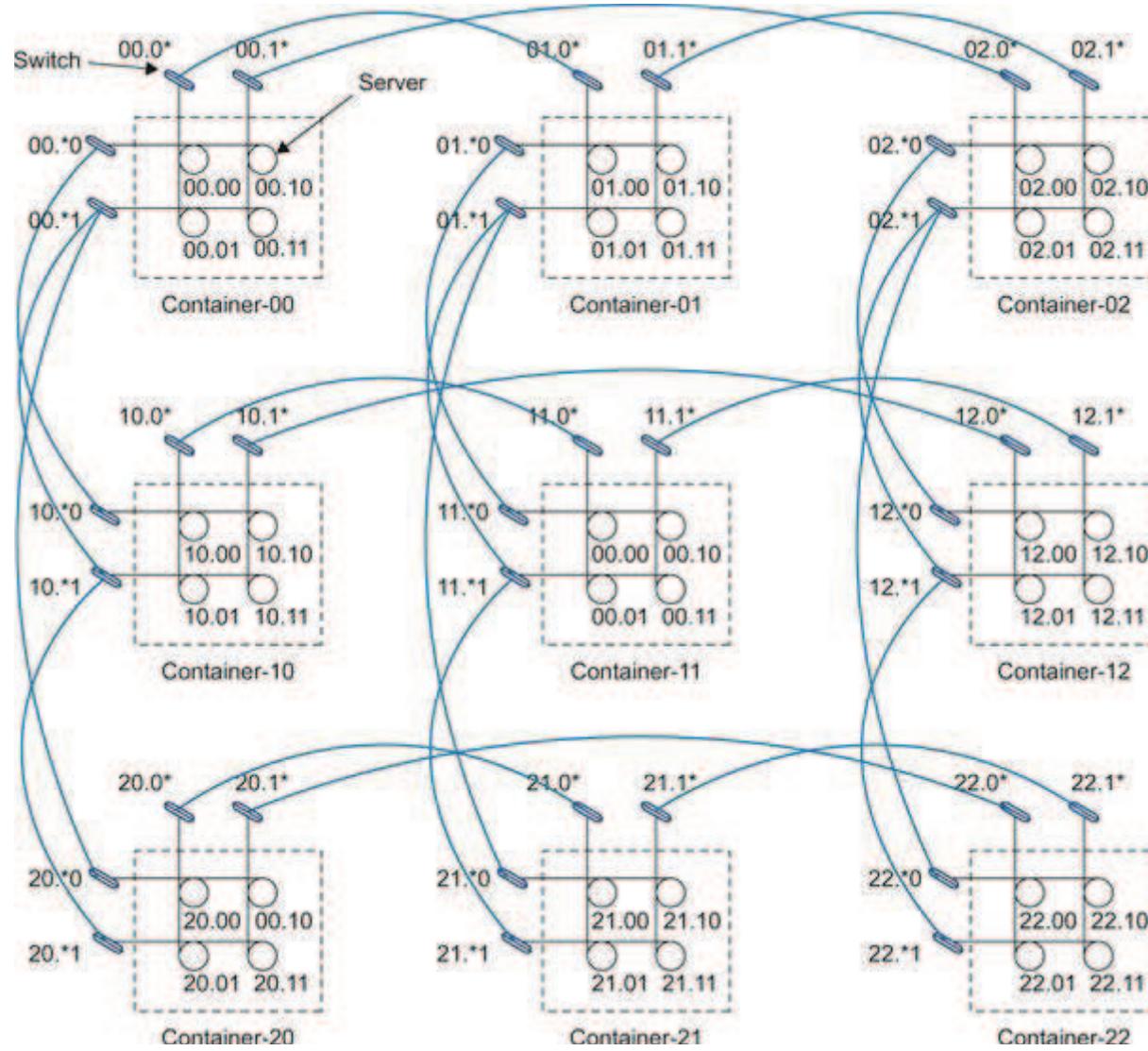


**FIGURE 4.12** BCube, a high-performance, server-centric network for building modular data centers Courtesy of C. Guo, et al. [30]

The BCube provides multiple paths between any two nodes. Multiple paths provide extra bandwidth to support communication patterns in different cloud applications. The BCube provides a kernel module in the server OS to perform routing operations. The kernel module supports packet forwarding while the incoming packets are not destined to the current node. Such modification of the kernel will not influence the upper layer applications. Thus, the cloud application can still run on top of the BCube network structure without any modification.

#### 4.2.4.1 Inter-Module Connection Networks

The BCube is commonly used inside a server container. The containers are considered the building blocks for data centers. Thus, despite the design of the inner container network, one needs another level of networking among multiple containers. In Figure 4.13, Wu, et al. [82] have proposed a network topology for intercontainer connection using the aforementioned BCube network as building blocks. The proposed network was named MDCube (for Modularized Datacenter Cube). This network connects multiple BCube containers by using high-speed switches in the BCube. Similarly, the MDCube is constructed by shuffling networks with multiple containers. Figure 4.13 shows how a 2D MDCube is constructed from nine BCube<sub>1</sub> containers.



**FIGURE 4.13** A 2D MDCube constructed from nine BCube containers Courtesy of Wu, et al. [82]

The architecture builds a virtual hypercube at the container level, in addition to the cube structure inside the container (BCube). With the server container built with the BCube network, the MDCube is used to build a large-scale data center for supporting cloud application communication patterns. Readers are referred to the article at [45] for detailed implementation and simulation results of this interconnection network over

multiple modular data centers built in containers. In fact, there are many other ways to use MDCube to build the network. Essentially, this network architecture builds a virtual hypercube at the container level, in addition to the cube structure inside the container (BCube). With the server container built with the BCube network, the MDCube is used to build a large-scale data center for supporting cloud application communication patterns [82].

## 4.2.5 Data-Center Management Issues

Here are basic requirements for managing the resources of a data center. These suggestions have resulted from the design and operational experiences of many data centers in the IT and service industries.

- **Making common users happy** The data center should be designed to provide quality service to the majority of users for at least 30 years.
- **Controlled information flow** Information flow should be streamlined. Sustained services and high availability (HA) are the primary goals.
- **Multiuser manageability** The system must be managed to support all functions of a data center, including traffic flow, database updating, and server maintenance.
- **Scalability to prepare for database growth** The system should allow growth as workload increases. The storage, processing, I/O, power, and cooling subsystems should be scalable.
- **Reliability in virtualized infrastructure** Failover, fault tolerance, and VM live migration should be integrated to enable recovery of critical applications from failures or disasters.
- **Low cost to both users and providers** The cost to users and providers of the cloud system built over the data centers should be reduced, including all operational costs.
- **Security enforcement and data protection** Data privacy and security defense mechanisms must be deployed to protect the data center against network attacks and system interrupts and to maintain data integrity from user abuses or network attacks.
- **Green information technology** Saving power consumption and upgrading energy efficiency are in high demand when designing and operating current and future data centers.

### 4.2.5.1 Marketplaces in Cloud Computing Services

Container-based data-center implementation can be done more efficiently with factory racking, stacking, and packing. One should avoid layers of packaging at the customer site. However, the data centers are still custom-crafted rather than prefab units. The modular approach is more space-efficient with power densities in excess of 1250 W/sq ft. Rooftop or parking lot installation is acceptable. One should leave sufficient redundancy to allow upgrades over time.

## 4.3 Architectural Design of Compute and Storage Clouds

This section presents basic cloud design principles. We start with basic cloud architecture to process massive amounts of data with a high degree of parallelism. Then we study virtualization support, resource provisioning, infrastructure management, and performance modeling.

## 4.3.1 A Generic Cloud Architecture Design

An Internet cloud is envisioned as a public cluster of servers provisioned on demand to perform collective web services or distributed applications using data-center resources. In this section, we will discuss cloud design objectives and then present a basic cloud architecture design.

### 4.3.1.1 Cloud Platform Design Goals

Scalability, virtualization, efficiency, and reliability are four major design goals of a cloud computing platform. Clouds support Web 2.0 applications. Cloud management receives the user request, finds the correct resources, and then calls the provisioning services which invoke the resources in the cloud. The cloud management software needs to support both physical and virtual machines. Security in shared resources and shared access of data centers also pose another design challenge.

The platform needs to establish a very large-scale HPC infrastructure. The hardware and software systems are combined to make it easy and efficient to operate. System scalability can benefit from cluster architecture. If one service takes a lot of processing power, storage capacity, or network traffic, it is simple to add more servers and bandwidth. System reliability can benefit from this architecture. Data can be put into multiple locations. For example, user e-mail can be put in three disks which expand to different geographically separate data centers. In such a situation, even if one of the data centers crashes, the user data is still accessible. The scale of the cloud architecture can be easily expanded by adding more servers and enlarging the network connectivity accordingly.

### 4.3.1.2 Enabling Technologies for Clouds

The key driving forces behind cloud computing are the ubiquity of broadband and wireless networking, falling storage costs, and progressive improvements in Internet computing software. Cloud users are able to demand more capacity at peak demand, reduce costs, experiment with new services, and remove unneeded capacity, whereas service providers can increase system utilization via multiplexing, virtualization, and dynamic resource provisioning. Clouds are enabled by the progress in hardware, software, and networking technologies summarized in [Table 4.3](#).

---

**Table 4.3**

**Cloud-Enabling Technologies in Hardware, Software, and Networking**

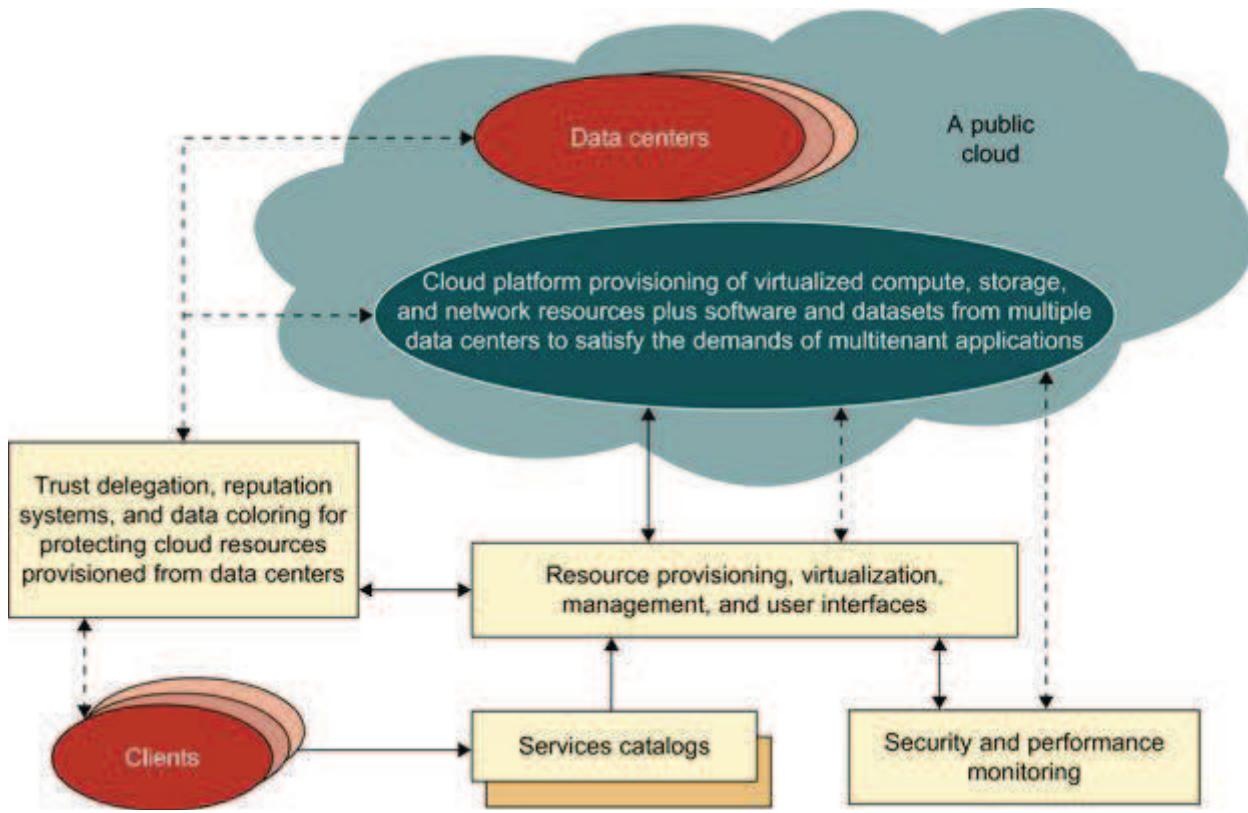
Technology	Requirements and Benefits
Fast platform deployment	Fast, efficient, and flexible deployment of cloud resources to provide dynamic computing environment to users
Virtual clusters on demand	Virtualized cluster of VMs provisioned to satisfy user demand and virtual cluster reconfigured as workload changes
Multitenant techniques	SaaS for distributing software to a large number of users for their simultaneous use and resource sharing if so desired
Massive data processing	Internet search and web services which often require massive data processing, especially to support personalized services
Web-scale communication	Support for e-commerce, distance education, telemedicine, social networking, digital government, and digital entertainment applications
Distributed storage	Large-scale storage of personal records and public archive information which demands distributed storage over the clouds
Licensing and billing services	License management and billing services which greatly benefit all types of cloud services in utility computing

These technologies play instrumental roles in making cloud computing a reality. Most of these technologies are mature today to meet increasing demand. In the hardware area, the rapid progress in multicore CPUs, memory chips, and disk arrays has made it possible to build faster data centers with huge amounts of storage space. Resource virtualization enables rapid cloud deployment and disaster recovery. *Service-oriented architecture* (SOA) also plays a vital role.

Progress in providing SaaS, Web 2.0 standards, and Internet performance have all contributed to the emergence of cloud services. Today's clouds are designed to serve a large number of tenants over massive volumes of data. The availability of large-scale, distributed storage systems is the foundation of today's data centers. Of course, cloud computing is greatly benefitted by the progress made in license management and automatic billing techniques in recent years.

#### 4.3.1.3 A Generic Cloud Architecture

Figure 4.14 shows a security-aware cloud architecture. The Internet cloud is envisioned as a massive cluster of servers. These servers are provisioned on demand to perform collective web services or distributed applications using data-center resources. The cloud platform is formed dynamically by provisioning or deprovisioning servers, software, and database resources. Servers in the cloud can be physical machines or VMs. User interfaces are applied to request services. The provisioning tool carves out the cloud system to deliver the requested service.



**FIGURE 4.14** A security-aware cloud platform built with a virtual cluster of VMs, storage, and networking resources over the data-center servers operated by providers Courtesy of K. Hwang and D. Li, 2010 [36]

In addition to building the server cluster, the cloud platform demands distributed storage and accompanying services. The cloud computing resources are built into the data centers, which are typically owned and operated by a third-party provider. Consumers do not need to know the underlying technologies. In a cloud, software becomes a service. The cloud demands a high degree of trust of massive amounts of data retrieved from large data centers. We need to build a framework to process large-scale data stored in the storage system. This demands a distributed file system over the database system. Other cloud resources are added into a cloud platform, including storage area networks (SANs), database systems, firewalls, and security devices. Web service providers offer special APIs that enable developers to exploit Internet clouds. Monitoring and metering units are used to track the usage and performance of provisioned resources.

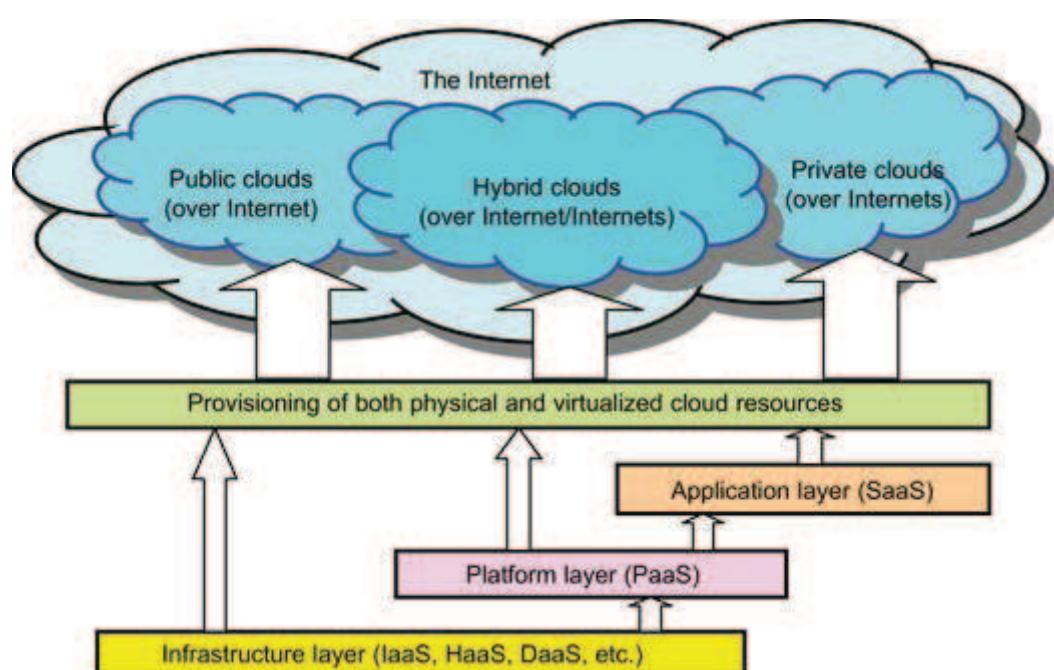
The software infrastructure of a cloud platform must handle all resource management and do most of the maintenance automatically. Software must detect the status of each node server joining and leaving, and perform relevant tasks accordingly. Cloud computing providers, such as Google and Microsoft, have built a large number of data centers all over the world. Each data center may have thousands of servers. The location of the data center is chosen to reduce power and cooling costs. Thus, the data centers are often built around hydroelectric power. The cloud physical platform builder is more concerned about the performance/price ratio and reliability issues than sheer speed performance.

In general, private clouds are easier to manage, and public clouds are easier to access. The trends in cloud development are that more and more clouds will be hybrid. This is

because many cloud applications must go beyond the boundary of an intranet. One must learn how to create a private cloud and how to interact with public clouds in the open Internet. Security becomes a critical issue in safeguarding the operation of all cloud types. We will study cloud security and privacy issues at the end of this chapter.

### 4.3.2 Layered Cloud Architectural Development

The architecture of a cloud is developed at three layers: infrastructure, platform, and application, as demonstrated in Figure 4.15. These three development layers are implemented with virtualization and standardization of hardware and software resources provisioned in the cloud. The services to public, private, and hybrid clouds are conveyed to users through networking support over the Internet and intranets involved. It is clear that the infrastructure layer is deployed first to support IaaS services. This infrastructure layer serves as the foundation for building the platform layer of the cloud for supporting PaaS services. In turn, the platform layer is a foundation for implementing the application layer for SaaS applications. Different types of cloud services demand application of these resources separately.



**FIGURE 4.15** Layered architectural development of the cloud platform for IaaS, PaaS, and SaaS applications over the Internet.

The infrastructure layer is built with virtualized compute, storage, and network resources. The abstraction of these hardware resources is meant to provide the flexibility demanded by users. Internally, virtualization realizes automated provisioning of resources and optimizes the infrastructure management process. The platform layer is for general-purpose and repeated usage of the collection of software resources. This layer provides users with an environment to develop their applications, to test operation flows, and to monitor execution results and performance. The platform should be able to assure

users that they have scalability, dependability, and security protection. In a way, the virtualized cloud platform serves as a “system middleware” between the infrastructure and application layers of the cloud.

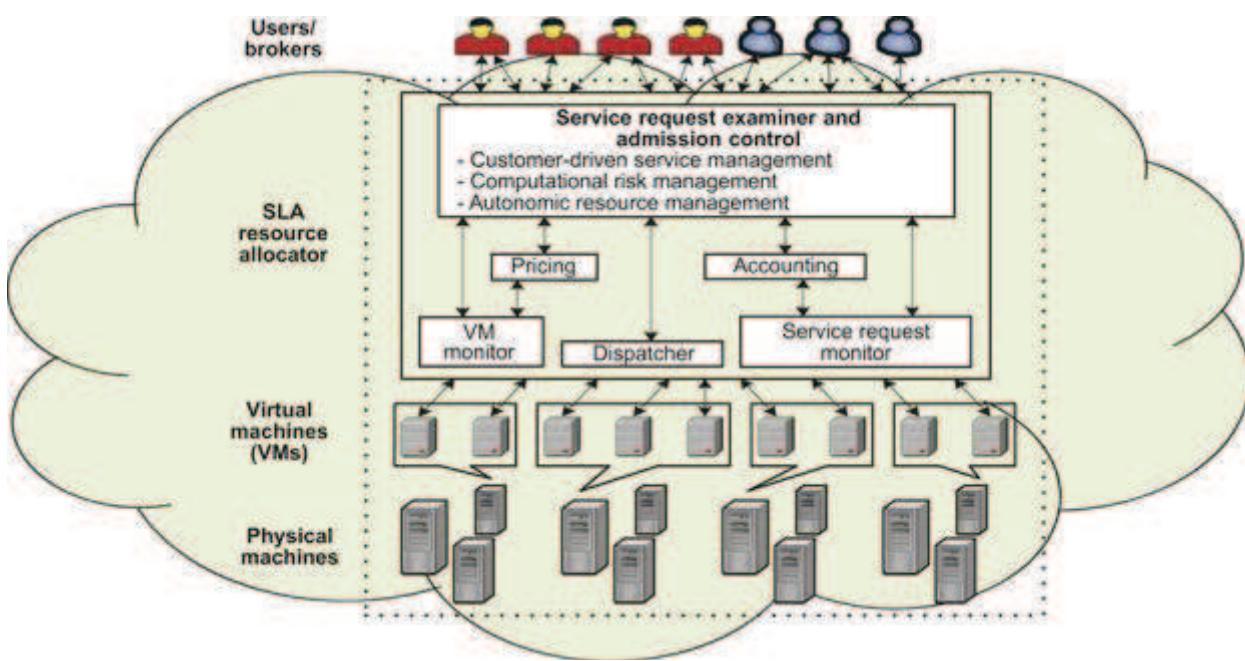
The application layer is formed with a collection of all needed software modules for SaaS applications. Service applications in this layer include daily office management work, such as information retrieval, document processing, and calendar and authentication services. The application layer is also heavily used by enterprises in business marketing and sales, consumer relationship management (CRM), financial transactions, and supply chain management. It should be noted that not all cloud services are restricted to a single layer. Many applications may apply resources at mixed layers. After all, the three layers are built from the bottom up with a dependence relationship.

From the provider’s perspective, the services at various layers demand different amounts of functionality support and resource management by providers. In general, SaaS demands the most work from the provider, PaaS is in the middle, and IaaS demands the least. For example, Amazon EC2 provides not only virtualized CPU resources to users, but also management of these provisioned resources. Services at the application layer demand more work from providers. The best example of this is the Salesforce.com CRM service, in which the provider supplies not only the hardware at the bottom layer and the software at the top layer, but also the platform and software tools for user application development and monitoring.

#### **4.3.2.1 Market-Oriented Cloud Architecture**

As consumers rely on cloud providers to meet more of their computing needs, they will require a specific level of QoS to be maintained by their providers, in order to meet their objectives and sustain their operations. Cloud providers consider and meet the different QoS parameters of each individual consumer as negotiated in specific SLAs. To achieve this, the providers cannot deploy traditional system-centric resource management architecture. Instead, market-oriented resource management is necessary to regulate the supply and demand of cloud resources to achieve market equilibrium between supply and demand.

The designer needs to provide feedback on economic incentives for both consumers and providers. The purpose is to promote QoS-based resource allocation mechanisms. In addition, clients can benefit from the potential cost reduction of providers, which could lead to a more competitive market, and thus lower prices. [Figure 4.16](#) shows the high-level architecture for supporting market-oriented resource allocation in a cloud computing environment. This cloud is basically built with the following entities:



**FIGURE 4.16** Market-oriented cloud architecture to expand/shrink leasing of resources with variation in QoS/demand from users Courtesy of Raj Buyya, et al. [11]

Users or brokers acting on user's behalf submit service requests from anywhere in the world to the data center and cloud to be processed. The SLA resource allocator acts as the interface between the data center/cloud service provider and external users/brokers. It requires the interaction of the following mechanisms to support SLA-oriented resource management. When a service request is first submitted the service request examiner interprets the submitted request for QoS requirements before determining whether to accept or reject the request.

The request examiner ensures that there is no overloading of resources whereby many service requests cannot be fulfilled successfully due to limited resources. It also needs the latest status information regarding resource availability (from the VM Monitor mechanism) and workload processing (from the Service Request Monitor mechanism) in order to make resource allocation decisions effectively. Then it assigns requests to VMs and determines resource entitlements for allocated VMs.

The Pricing mechanism decides how service requests are charged. For instance, requests can be charged based on submission time (peak/off-peak), pricing rates (fixed/changing), or availability of resources (supply/demand). Pricing serves as a basis for managing the supply and demand of computing resources within the data center and facilitates in prioritizing resource allocations effectively. The Accounting mechanism maintains the actual usage of resources by requests so that the final cost can be computed and charged to users. In addition, the maintained historical usage information can be utilized by the Service Request Examiner and Admission Control mechanism to improve resource allocation decisions.

The VM Monitor mechanism keeps track of the availability of VMs and their resource entitlements. The Dispatcher mechanism starts the execution of accepted service requests on allocated VMs. The Service Request Monitor mechanism keeps track of the execution progress of service requests. Multiple VMs can be started and stopped on

demand on a single physical machine to meet accepted service requests, hence providing maximum flexibility to configure various partitions of resources on the same physical machine to different specific requirements of service requests. In addition, multiple VMs can concurrently run applications based on different operating system environments on a single physical machine since the VMs are isolated from one another on the same physical machine.

#### **4.3.2.2 Quality of Service Factors**

The data center comprises multiple computing servers that provide resources to meet service demands. In the case of a cloud as a commercial offering to enable crucial business operations of companies, there are critical QoS parameters to consider in a service request, such as time, cost, reliability, and trust/security. In particular, QoS requirements cannot be static and may change over time due to continuing changes in business operations and operating environments. In short, there should be greater importance on customers since they pay to access services in clouds. In addition, the state of the art in cloud computing has no or limited support for dynamic negotiation of SLAs between participants and mechanisms for automatic allocation of resources to multiple competing requests. Negotiation mechanisms are needed to respond to alternate offers protocol for establishing SLAs [72].

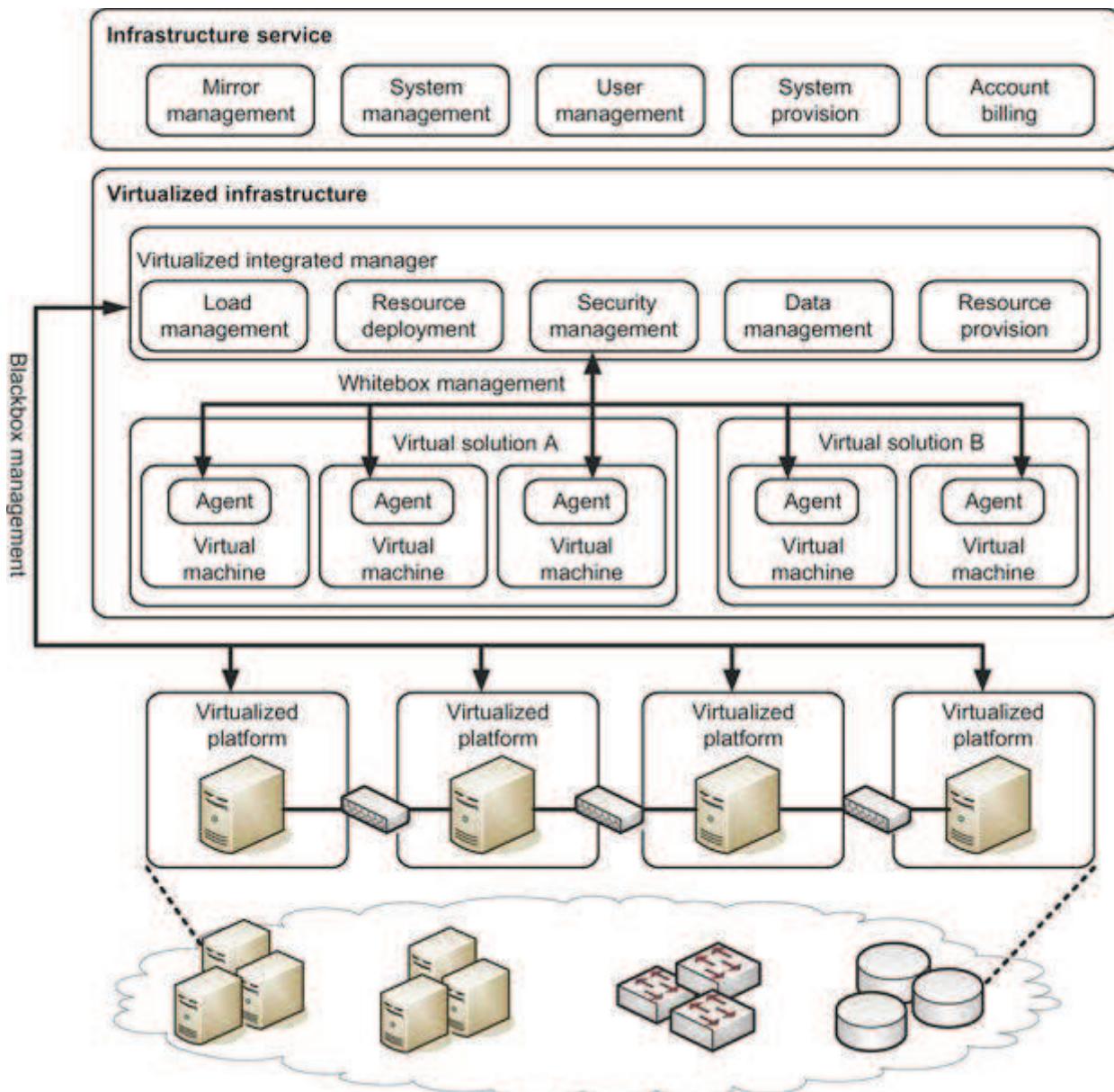
Commercial cloud offerings must be able to support customer-driven service management based on customer profiles and requested service requirements. Commercial clouds define computational risk management tactics to identify, assess, and manage risks involved in the execution of applications with regard to service requirements and customer needs. The cloud also derives appropriate market-based resource management strategies that encompass both customer-driven service management and computational risk management to sustain SLA-oriented resource allocation. The system incorporates autonomic resource management models that effectively self-manage changes in service requirements to satisfy both new service demands and existing service obligations, and leverage VM technology to dynamically assign resource shares according to service requirements.

#### **4.3.3 Virtualization Support and Disaster Recovery**

One very distinguishing feature of cloud computing infrastructure is the use of system virtualization and the modification to provisioning tools. Virtualization of servers on a shared cluster can consolidate web services. As the VMs are the containers of cloud services, the provisioning tools will first find the corresponding physical machines and deploy the VMs to those nodes before scheduling the service to run on the virtual nodes.

In addition, in cloud computing, virtualization also means the resources and fundamental infrastructure are virtualized. The user will not care about the computing resources that are used for providing the services. Cloud users do not need to know and have no way to discover physical resources that are involved while processing a service request. Also, application developers do not care about some infrastructure issues such

as scalability and fault tolerance (i.e., they are virtualized). Application developers focus on service logic. Figure 4.17 shows the infrastructure needed to virtualize the servers in a data center for implementing specific cloud applications.



**FIGURE 4.17** Virtualized servers, storage, and network for cloud platform construction Courtesy of Zhong-Yuan Qin, SouthEast University, China

#### 4.3.3.1 Hardware Virtualization

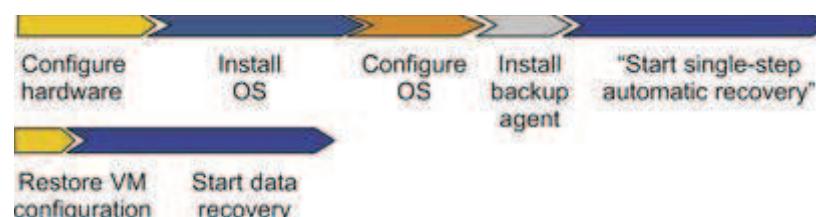
In many cloud computing systems, virtualization software is used to virtualize the hardware. System virtualization software is a special kind of software which simulates the execution of hardware and runs even unmodified operating systems. Cloud computing systems use virtualization software as the running environment for legacy software such as old operating systems and unusual applications. Virtualization software is also used as the platform for developing new cloud applications that enable developers to use any operating systems and programming environments they like. The

development environment and deployment environment can now be the same, which eliminates some runtime problems.

Some cloud computing providers have used virtualization technology to provide this service for developers. As mentioned before, system virtualization software is considered the hardware analog mechanism to run an unmodified operating system, usually on bare hardware directly, on top of software. [Table 4.4](#) lists some of the system virtualization software in wide use at the time of this writing. Currently, the VMs installed on a cloud computing platform are mainly used for hosting third-party programs. VMs provide flexible runtime services to free users from worrying about the system environment.

Using VMs in a cloud computing platform ensures extreme flexibility for users. As the computing resources are shared by many users, a method is required to maximize the users' privileges and still keep them separated safely. Traditional sharing of cluster resources depends on the user and group mechanism on a system. Such sharing is not flexible. Users cannot customize the system for their special purposes. Operating systems cannot be changed. The separation is not complete. An environment that meets one user's requirements often cannot satisfy another user. Virtualization allows users to have full privileges while keeping them separate.

Users have full access to their own VMs, which are completely separate from other users' VMs. Multiple VMs can be mounted on the same physical server. Different VMs may run with different OSes. We also need to establish the virtual disk storage and virtual networks needed by the VMs. The virtualized resources form a resource pool. The virtualization is carried out by special servers dedicated to generating the virtualized resource pool. The virtualized infrastructure (black box in the middle) is built with many *virtualizing integration managers*. These managers handle loads, resources, security, data, and provisioning functions. [Figure 4.18](#) shows two VM platforms. Each platform carries out a virtual solution to a user job. All cloud services are managed in the boxes at the top.



**FIGURE 4.18** Recovery overhead of a conventional disaster recovery scheme, compared with that required to recover from live migration of VMs.

#### 4.3.3.2 Virtualization Support in Public Clouds

Armbrust, et al. [4] have assessed in [Table 4.4](#) three public clouds in the context of virtualization support: AWS, Microsoft Azure, and GAE. AWS provides extreme flexibility (VMs) for users to execute their own applications. GAE provides limited application-level virtualization for users to build applications only based on the services that are created by Google. Microsoft provides programming-level virtualization (.NET

virtualization) for users to build their applications.

**Table 4.4**

### **Virtualized Resources in Compute, Storage, and Network Clouds [4]**

Provider	AWS	Microsoft Azure	GAE
<b>Compute cloud with virtual cluster of servers</b>	x86 instruction set, Xen VMs, resource elasticity allows scalability through virtual cluster, or a third party such as RightScale must provide the cluster	Common language runtime VMs provisioned by declarative descriptions	Predefined application framework handlers written in Python, automatic scaling up and down, server failover inconsistent with the web applications
<b>Storage cloud with virtual storage</b>	Models for block store (EBS) and augmented key/blob store (SimpleDB), automatic scaling varies from EBS to fully automatic (SimpleDB, S3)	SQL Data Services (restricted view of SQL Server), Azure storage service	MegaStore/BigTable
<b>Network cloud services</b>	Declarative IP-level topology; placement details hidden, security groups restricting communication, availability zones isolate network failure, elastic IP applied	Automatic with user's declarative descriptions or roles of app. components	Fixed topology to accommodate three-tier web app. structure, scaling up and down is automatic and programmer-invisible

The VMware tools apply to workstations, servers, and virtual infrastructure. The Microsoft tools are used on PCs and some special servers. The XenEnterprise tool applies only to Xen-based servers. Everyone is interested in the cloud; the entire IT industry is moving toward the vision of the cloud. Virtualization leads to HA, disaster recovery, dynamic load leveling, and rich provisioning support. Both cloud computing and utility computing leverage the benefits of virtualization to provide a scalable and autonomous computing environment.

#### **4.3.3.3 Storage Virtualization for Green Data Centers**

IT power consumption in the United States has more than doubled to 3 percent of the total energy consumed in the country. The large number of data centers in the country has contributed to this energy crisis to a great extent. More than half of the companies in the Fortune 500 are actively implementing new corporate energy policies. Recent surveys from both IDC and Gartner confirm the fact that virtualization had a great impact on cost reduction from reduced power consumption in physical computing systems. This alarming situation has made the IT industry become more energy-aware. With little evolution of alternate energy resources, there is an imminent need to conserve power in all computers. Virtualization and server consolidation have already proven handy in this aspect. Green data centers and benefits of storage virtualization are considered to further strengthen the synergy of green computing.

#### **4.3.3.4 Virtualization for IaaS**

VM technology has increased in ubiquity. This has enabled users to create customized

environments atop physical infrastructure for cloud computing. Use of VMs in clouds has the following distinct benefits: (1) System administrators consolidate workloads of underutilized servers in fewer servers; (2) VMs have the ability to run legacy code without interfering with other APIs; (3) VMs can be used to improve security through creation of sandboxes for running applications with questionable reliability; And (4) virtualized cloud platforms can apply performance isolation, letting providers offer some guarantees and better QoS to customer applications.

#### **4.3.3.5 VM Cloning for Disaster Recovery**

VM technology requires an advanced disaster recovery scheme. One scheme is to recover one physical machine by another physical machine. The second scheme is to recover one VM by another VM. As shown in the top timeline of [Figure 4.18](#), traditional disaster recovery from one physical machine to another is rather slow, complex, and expensive. Total recovery time is attributed to the hardware configuration, installing and configuring the OS, installing the backup agents, and the long time to restart the physical machine. To recover a VM platform, the installation and configuration times for the OS and backup agents are eliminated. Therefore, we end up with a much shorter disaster recovery time, about 40 percent of that to recover the physical machines. Virtualization aids in fast disaster recovery by VM encapsulation.

We discussed disaster recovery in [Chapters 2](#) and [3](#). The cloning of VMs offers an effective solution. The idea is to make a clone VM on a remote server for every running VM on a local server. Among all the clone VMs, only one needs to be active. The remote VM should be in a suspended mode. A cloud control center should be able to activate this clone VM in case of failure of the original VM, taking a snapshot of the VM to enable live migration in a minimal amount of time. The migrated VM can run on a shared Internet connection. Only updated data and modified states are sent to the suspended VM to update its state. The *Recovery Property Objective (RPO)* and *Recovery Time Objective (RTO)* are affected by the number of snapshots taken. Security of the VMs should be enforced during live migration of VMs.

#### **4.3.4 Architectural Design Challenges**

In this section, we will identify six open challenges in cloud architecture development. Armbrust, et al. [\[4\]](#) have observed some of these topics as both obstacles and opportunities. Plausible solutions to meet these challenges are discussed shortly.

##### **4.3.4.1 Challenge 1—Service Availability and Data Lock-in Problem**

The management of a cloud service by a single company is often the source of single points of failure. To achieve HA, one can consider using multiple cloud providers. Even if a company has multiple data centers located in different geographic regions, it may have common software infrastructure and accounting systems. Therefore, using multiple cloud providers may provide more protection from failures. Another availability obstacle is distributed *denial of service (DDoS)* attacks. Criminals threaten to cut off the incomes of

SaaS providers by making their services unavailable. Some utility computing services offer SaaS providers the opportunity to defend against DDoS attacks by using quick scale-ups.

Software stacks have improved interoperability among different cloud platforms, but the APIs themselves are still proprietary. Thus, customers cannot easily extract their data and programs from one site to run on another. The obvious solution is to standardize the APIs so that a SaaS developer can deploy services and data across multiple cloud providers. This will rescue the loss of all data due to the failure of a single company. In addition to mitigating data lock-in concerns, standardization of APIs enables a new usage model in which the same software infrastructure can be used in both public and private clouds. Such an option could enable “surge computing,” in which the public cloud is used to capture the extra tasks that cannot be easily run in the data center of a private cloud.

#### **4.3.4.2 Challenge 2—Data Privacy and Security Concerns**

Current cloud offerings are essentially public (rather than private) networks, exposing the system to more attacks. Many obstacles can be overcome immediately with well-understood technologies such as encrypted storage, virtual LANs, and network middleboxes (e.g., firewalls, packet filters). For example, you could encrypt your data before placing it in a cloud. Many nations have laws requiring SaaS providers to keep customer data and copyrighted material within national boundaries.

Traditional network attacks include buffer overflows, DoS attacks, spyware, malware, rootkits, Trojan horses, and worms. In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM rootkits. Another type of attack is the man-in-the-middle attack for VM migrations. In general, passive attacks steal sensitive data or passwords. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers. We will study all of these security and privacy problems on clouds in [Section 4.5](#).

#### **4.3.4.3 Challenge 3—Unpredictable Performance and Bottlenecks**

Multiple VMs can share CPUs and main memory in cloud computing, but I/O sharing is problematic. For example, to run 75 EC2 instances with the STREAM benchmark requires a mean bandwidth of 1,355 MB/second. However, for each of the 75 EC2 instances to write 1 GB files to the local disk requires a mean disk write bandwidth of only 55 MB/second. This demonstrates the problem of I/O interference between VMs. One solution is to improve I/O architectures and operating systems to efficiently virtualize interrupts and I/O channels.

Internet applications continue to become more data-intensive. If we assume applications to be “pulled apart” across the boundaries of clouds, this may complicate data placement and transport. Cloud users and providers have to think about the implications of placement and traffic at every level of the system, if they want to minimize costs. This kind of reasoning can be seen in Amazon’s development of its new

CloudFront service. Therefore, data transfer bottlenecks must be removed, bottleneck links must be widened, and weak servers should be removed. We will study performance issues in [Chapter 8](#).

#### **4.3.4.4 Challenge 4—Distributed Storage and Widespread Software Bugs**

The database is always growing in cloud applications. The opportunity is to create a storage system that will not only meet this growth, but also combine it with the cloud advantage of scaling arbitrarily up and down on demand. This demands the design of efficient distributed SANs. Data centers must meet programmers' expectations in terms of scalability, data durability, and HA. Data consistency checking in SAN-connected data centers is a major challenge in cloud computing.

Large-scale distributed bugs cannot be reproduced, so the debugging must occur at a scale in the production data centers. No data center will provide such a convenience. One solution may be a reliance on using VMs in cloud computing. The level of virtualization may make it possible to capture valuable information in ways that are impossible without using VMs. Debugging over simulators is another approach to attacking the problem, if the simulator is well designed.

#### **4.3.4.5 Challenge 5—Cloud Scalability, Interoperability, and Standardization**

The pay-as-you-go model applies to storage and network bandwidth; both are counted in terms of the number of bytes used. Computation is different depending on virtualization level. GAE automatically scales in response to load increases and decreases; users are charged by the cycles used. AWS charges by the hour for the number of VM instances used, even if the machine is idle. The opportunity here is to scale quickly up and down in response to load variation, in order to save money, but without violating SLAs.

*Open Virtualization Format (OVF)* describes an open, secure, portable, efficient, and extensible format for the packaging and distribution of VMs. It also defines a format for distributing software to be deployed in VMs. This VM format does not rely on the use of a specific host platform, virtualization platform, or guest operating system. The approach is to address virtual platform-agnostic packaging with certification and integrity of packaged software. The package supports virtual appliances to span more than one VM.

OVF also defines a transport mechanism for VM templates, and can apply to different virtualization platforms with different levels of virtualization. In terms of cloud standardization, we suggest the ability for virtual appliances to run on any virtual platform. We also need to enable VMs to run on heterogeneous hardware platform hypervisors. This requires hypervisor-agnostic VMs. We also need to realize cross-platform live migration between x86 Intel and AMD technologies and support legacy hardware for load balancing. All these issue are wide open for further research.

#### **4.3.4.6 Challenge 6—Software Licensing and Reputation Sharing**

Many cloud computing providers originally relied on open source software because the

licensing model for commercial software is not ideal for utility computing. The primary opportunity is either for open source to remain popular or simply for commercial software companies to change their licensing structure to better fit cloud computing. One can consider using both pay-for-use and bulk-use licensing schemes to widen the business coverage.

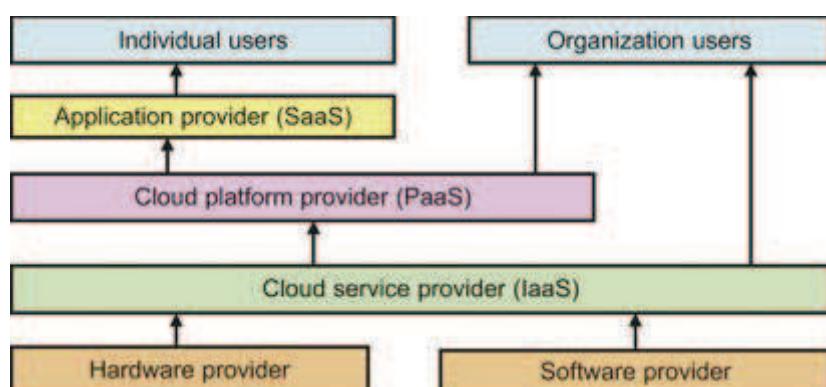
One customer's bad behavior can affect the reputation of the entire cloud. For instance, blacklisting of EC2 IP addresses by spam-prevention services may limit smooth VM installation. An opportunity would be to create reputation-guarding services similar to the "trusted e-mail" services currently offered (for a fee) to services hosted on smaller ISPs. Another legal issue concerns the transfer of legal liability. Cloud providers want legal liability to remain with the customer, and vice versa. This problem must be solved at the SLA level. We will study reputation systems for protecting data centers in the next section.

## 4.4 Public Cloud Platforms: GAE, AWS, and Azure

In this section, we will review the system architectures of four commercially available cloud platforms. These case studies will prepare readers for subsequent sections and chapters.

### 4.4.1 Public Clouds and Service Offerings

Cloud services are demanded by computing and IT administrators, software vendors, and end users. [Figure 4.19](#) introduces five levels of cloud players. At the top level, individual users and organizational users demand very different services. The application providers at the SaaS level serve mainly individual users. Most business organizations are serviced by IaaS and PaaS providers. The infrastructure services (IaaS) provide compute, storage, and communication resources to both applications and organizational users. The cloud environment is defined by the PaaS or platform providers. Note that the platform providers support both infrastructure services and organizational users directly.



**FIGURE 4.19** Roles of individual and organizational users and their interaction with cloud providers under various cloud service models.

Cloud services rely on new advances in machine virtualization, SOA, grid infrastructure management, and power efficiency. Consumers purchase such services in the form of IaaS, PaaS, or SaaS as described earlier. Also, many cloud entrepreneurs are selling value-added utility services to massive numbers of users. The cloud industry leverages the growing demand by many enterprises and business users to outsource their computing and storage jobs to professional providers. The provider service charges are often much lower than the cost for users to replace their obsolete servers frequently. Table 4.5 summarizes the profiles of five major cloud providers by 2010 standards.

**Table 4.5**

### Five Major Cloud Platforms and Their Service Offerings [36]

Model	IBM	Amazon	Google	Microsoft	Salesforce
PaaS	BlueCloud, WCA, RC2		App Engine (GAE)	Windows Azure	Force.com
IaaS	Ensembles	AWS		Windows Azure	
SaaS	Lotus Live		Gmail, Docs	.NET service, Dynamic CRM	Online CRM, Gifttag
Virtualization		OS and Xen	Application Container	OS level/ Hyper-V	
Service Offerings	SOA, B2, TSAM, RAD, Web 2.0	EC2, S3, SQS, SimpleDB	GFS, Chubby, BigTable, MapReduce	Live, SQL Hotmail	Apex, visual force, record security
Security Features	WebSphere2 and PowerVM tuned for protection	PKI, VPN, EBS to recover from failure	Chubby locks for security enforcement	Replicated data, rule- based access control	Admin./record security, uses metadata API
User Interfaces		EC2 command-line tools	Web-based admin. console	Windows Azure portal	
Web API	Yes	Yes	Yes	Yes	Yes
Programming Support	AMI		Python	.NET Framework	

**Note:** WCA: WebSphere CloudBurst Appliance; RC2: Research Compute Cloud; RAD: Rational Application Developer; SOA: Service-Oriented Architecture; TSAM: Tivoli Service Automation Manager; EC2: Elastic Compute Cloud; S3: Simple Storage Service; SQS: Simple Queue Service; GAE: Google App Engine; AWS: Amazon Web Services; SQL: Structured Query Language; EBS: Elastic Block Store; CRM: Consumer Relationship Management.

Amazon pioneered the IaaS business in supporting e-commerce and cloud applications by millions of customers simultaneously. The elasticity in the Amazon cloud comes from the flexibility provided by the hardware and software services. EC2 provides an environment for running virtual servers on demand. S3 provides unlimited online storage space. Both EC2 and S3 are supported in the AWS platform. Microsoft offers the Azure platform for cloud applications. It has also supported the .NET service, dynamic CRM, Hotmail, and SQL applications. Salesforce.com offers extensive SaaS applications for online CRM applications using its Force.com platforms.

As Table 4.5 shows, all IaaS, PaaS, and SaaS models allow users to access services over

the Internet, relying entirely on the infrastructures of the cloud service providers. These models are offered based on various SLAs between the providers and the users. SLAs are more common in network services as they account for the QoS characteristics of network services. For cloud computing services, it is difficult to find a reasonable precedent for negotiating an SLA. In a broader sense, the SLAs for cloud computing address service availability, data integrity, privacy, and security protection. Blank spaces in the table refer to unknown or underdeveloped features.

## 4.4.2 Google App Engine (GAE)

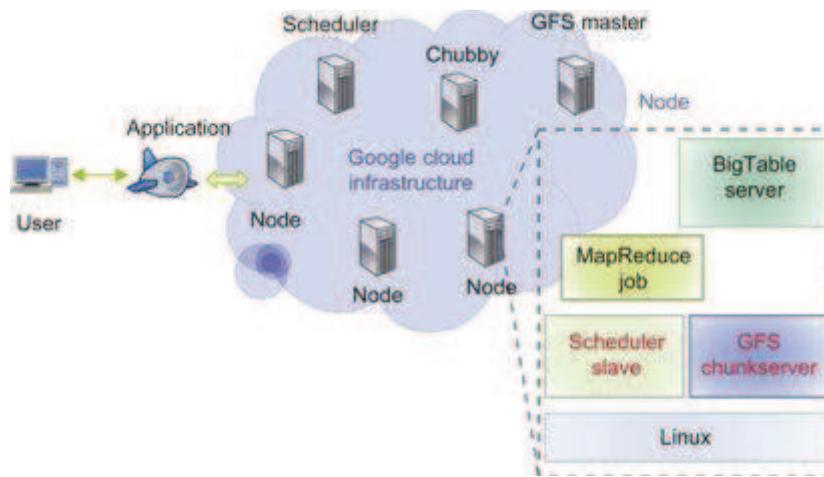
Google has the world's largest search engine facilities. The company has extensive experience in massive data processing that has led to new insights into data-center design (see [Chapter 3](#)) and novel programming models that scale to incredible sizes. The Google platform is based on its search engine expertise, but as discussed earlier with MapReduce, this infrastructure is applicable to many other areas. Google has hundreds of data centers and has installed more than 460,000 servers worldwide. For example, 200 Google data centers are used at one time for a number of cloud applications. Data items are stored in text, images, and video and are replicated to tolerate faults or failures. Here we discuss Google's App Engine (GAE) which offers a PaaS platform supporting various cloud and web applications.

### 4.4.2.1 Google Cloud Infrastructure

Google has pioneered cloud development by leveraging the large number of data centers it operates. For example, Google pioneered cloud services in Gmail, Google Docs, and Google Earth, among other applications. These applications can support a large number of users simultaneously with HA. Notable technology achievements include the Google File System (GFS), MapReduce, BigTable, and Chubby. In 2008, Google announced the GAE web application platform which is becoming a common platform for many small cloud service providers. This platform specializes in supporting scalable (elastic) web applications. GAE enables users to run their applications on a large number of data centers associated with Google's search engine operations.

### 4.4.2.2 GAE Architecture

Figure 4.20 shows the major building blocks of the Google cloud platform which has been used to deliver the cloud services highlighted earlier. GFS is used for storing large amounts of data. MapReduce is for use in application program development. Chubby is used for distributed application lock services. BigTable offers a storage service for accessing structured data. These technologies are described in more detail in [Chapter 8](#). Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications all run in data centers under tight management by Google engineers. Inside each data center, there are thousands of servers forming different clusters.



**FIGURE 4.20** Google cloud platform and major building blocks, the blocks shown are large clusters of low-cost servers  
Courtesy of Kang Chen, Tsinghua University, China

Google is one of the larger cloud application providers, although its fundamental service program is private and outside people cannot use the Google infrastructure to build their own service. The building blocks of Google's cloud computing application include the Google File System for storing large amounts of data, the MapReduce programming framework for application developers, Chubby for distributed application lock services, and BigTable as a storage service for accessing structural or semistructural data. With these building blocks, Google has built many cloud applications. Figure 4.20 shows the overall architecture of the Google cloud infrastructure. A typical cluster configuration can run the Google File System, MapReduce jobs, and BigTable servers for structure data. Extra services such as Chubby for distributed locks can also run in the clusters.

GAE runs the user program on Google's infrastructure. As it is a platform running third-party programs, application developers now do not need to worry about the maintenance of servers. GAE can be thought of as the combination of several software components. The frontend is an application framework which is similar to other web application frameworks such as ASP, J2EE, and JSP. At the time of this writing, GAE supports Python and Java programming environments. The applications can run similar to web application containers. The frontend can be used as the dynamic web serving infrastructure which can provide the full support of common technologies.

#### 4.4.2.3 Functional Modules of GAE

The GAE platform comprises the following five major components. The GAE is not an infrastructure platform, but rather an application development platform for users. We describe the component functionalities separately.

- The *datastore* offers object-oriented, distributed, structured data storage services based on BigTable techniques. The datastore secures data management operations.
- The *application runtime environment* offers a platform for scalable web programming and execution. It supports two development languages: Python and Java.
- The *software development kit (SDK)* is used for local application development. The SDK allows users to execute test runs of local applications and upload application code.

- d. The *administration console* is used for easy management of user application development cycles, instead of for physical resource management.
- e. The *GAE web service infrastructure* provides special interfaces to guarantee flexible use and management of storage and network resources by GAE.

Google offers essentially free GAE services to all Gmail account owners. You can register for a GAE account or use your Gmail account name to sign up for the service. The service is free within a quota. If you exceed the quota, the page instructs you on how to pay for the service. Then you download the SDK and read the Python or Java guide to get started. Note that GAE only accepts Python, Ruby, and Java programming languages. The platform does not provide any IaaS services, unlike Amazon, which offers IaaS and PaaS. This model allows the user to deploy user-built applications on top of the cloud infrastructure that are built using the programming languages and software tools supported by the provider (e.g., Java, Python). Azure does this similarly for .NET. The user does not manage the underlying cloud infrastructure. The cloud provider facilitates support of application development, testing, and operation support on a well-defined service platform.

#### **4.4.2.4 GAE Applications**

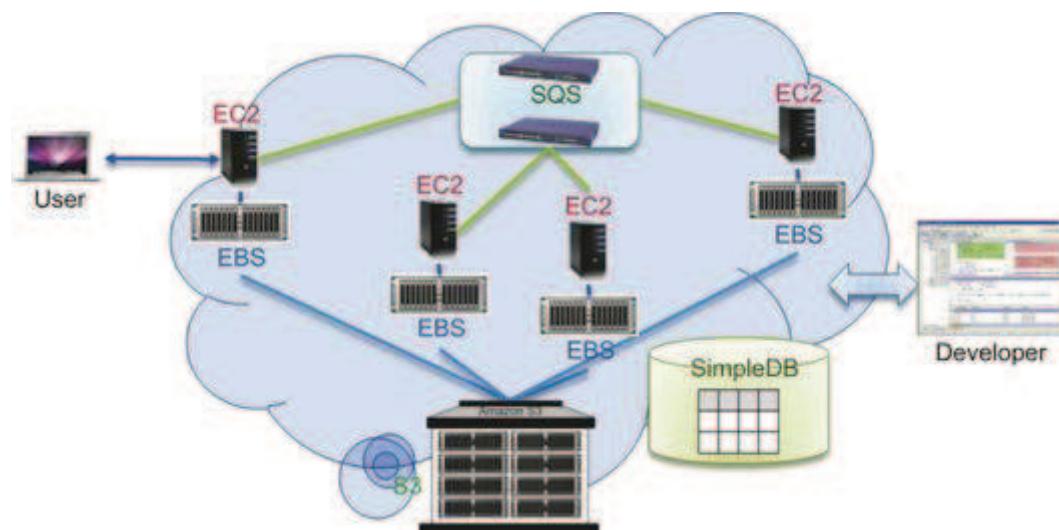
Well-known GAE applications include the Google Search Engine, Google Docs, Google Earth, and Gmail. These applications can support large numbers of users simultaneously. Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications are all run in the Google data centers. Inside each data center, there might be thousands of server nodes to form different clusters. (See the previous section.) Each cluster can run multipurpose servers.

GAE supports many web applications. One is a storage service to store application-specific data in the Google infrastructure. The data can be persistently stored in the backend storage server while still providing the facility for queries, sorting, and even transactions similar to traditional database systems. GAE also provides Google-specific services, such as the Gmail account service (which is the login service, that is, applications can use the Gmail account directly). This can eliminate the tedious work of building customized user management components in web applications. Thus, web applications built on top of GAE can use the APIs authenticating users and sending e-mail using Google accounts.

#### **4.4.3 Amazon Web Services (AWS)**

VMs can be used to share computing resources both flexibly and safely. Amazon has been a leader in providing public cloud services (<http://aws.amazon.com/>). Amazon applies the IaaS model in providing its services. [Figure 4.21](#) shows the AWS architecture. EC2 provides the virtualized platforms to the host VMs where the cloud application can run. S3 (Simple Storage Service) provides the object-oriented storage service for users. EBS (Elastic Block Service) provides the block storage interface which can be used to

support traditional applications. SQS stands for Simple Queue Service, and its job is to ensure a reliable message service between two processes. The message can be kept reliably even when the receiver processes are not running. Users can access their objects through SOAP with either browsers or other client programs which support the SOAP standard.



**FIGURE 4.21** Amazon cloud computing infrastructure (Key services are identified here; many more are listed in Table 4.6) Courtesy of Kang Chen, Tsinghua University, China

Table 4.6 summarizes the service offerings by AWS in 12 application tracks. Details of EC2, S3, and EBS are available in Chapter 6 where we discuss programming examples. Amazon offers queuing and notification services (SQS and SNS), which are implemented in the AWS cloud. Note brokering systems run very efficiently in clouds and offer a striking model for controlling sensors and providing office support of smartphones and tablets. Different from Google, Amazon provides a more flexible cloud computing platform for developers to build cloud applications. Small and medium-size companies can put their business on the Amazon cloud platform. Using the AWS platform, they can service large numbers of Internet users and make profits through those paid services.

---

## Table 4.6

### AWS Offerings in 2011

---

Service Area	Service Modules and Abbreviated Names
Compute	Elastic Compute Cloud (EC2), Elastic MapReduce, Auto Scaling
Messaging	Simple Queue Service (SQS), Simple Notification Service (SNS)
Storage	Simple Storage Service (S3), Elastic Block Storage (EBS), AWS Import/Export
Content Delivery	Amazon CloudFront
Monitoring	Amazon CloudWatch
Support	AWS Premium Support
Database	Amazon SimpleDB, Relational Database Service (RDS)

<b>Networking</b>	Virtual Private Cloud (VPC) ( <a href="#">Example 4.1, Figure 4.6</a> ), Elastic Load Balancing
<b>Web Traffic</b>	Alexa Web Information Service, Alexa Web Sites
<b>E-Commerce</b>	Fulfillment Web Service (FWS)
<b>Payments and Billing</b>	Flexible Payments Service (FPS), Amazon DevPay
<b>Workforce</b>	Amazon Mechanical Turk

Courtesy of Amazon, <http://aws.amazon.com> [3]

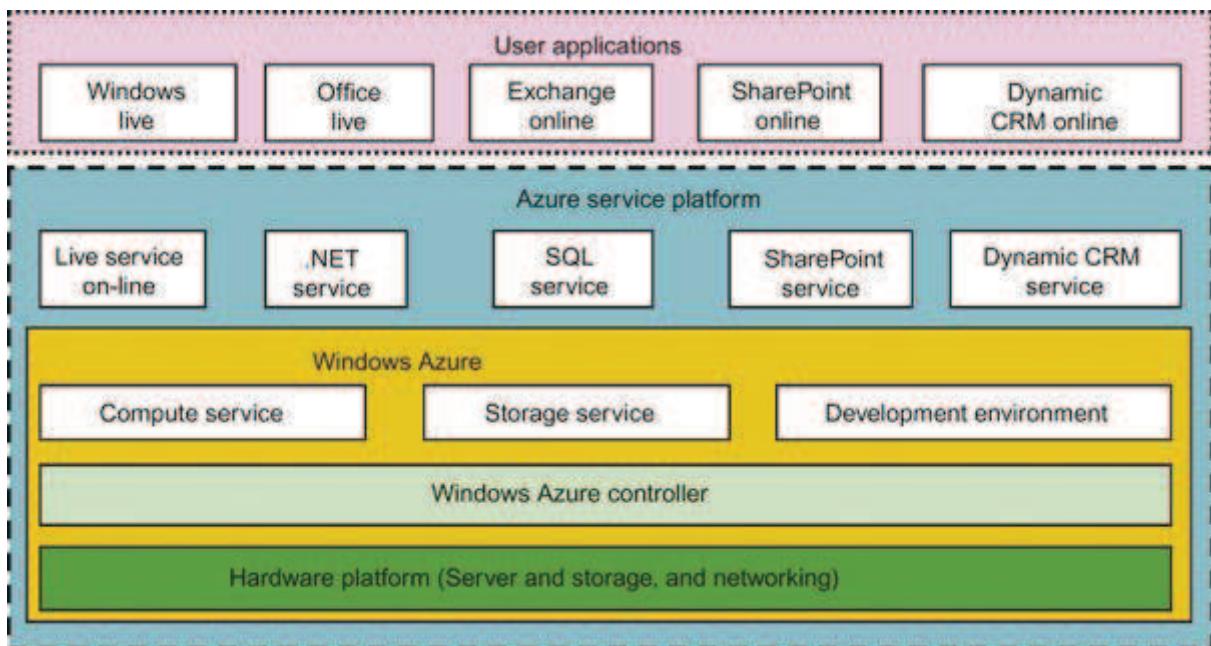
ELB automatically distributes incoming application traffic across multiple Amazon EC2 instances and allows user to avoid nonoperating nodes and to equalize load on functioning images. Both auto-scaling and ELB are enabled by CloudWatch which monitors running instances. CloudWatch is a web service that provides monitoring for AWS cloud resources, starting with Amazon EC2. It provides customers with visibility into resource utilization, operational performance, and overall demand patterns, including metrics such as CPU utilization, disk reads and writes, and network traffic.

Amazon (like Azure) offers a *Relational Database Service (RDS)* with a messaging interface to be covered in [Section 4.1](#). The Elastic MapReduce capability is equivalent to Hadoop running on the basic EC2 offering. AWS Import/Export allows one to ship large volumes of data to and from EC2 by shipping physical disks; it is well known that this is often the highest bandwidth connection between geographically distant systems. Amazon CloudFront implements a content distribution network. Amazon DevPay is a simple-to-use online billing and account management service that makes it easy for businesses to sell applications that are built into or run on top of AWS.

FPS provides developers of commercial systems on AWS with a convenient way to charge Amazon's customers that use such services built on AWS. Customers can pay using the same login credentials, shipping address, and payment information they already have on file with Amazon. The FWS allows merchants to access Amazon's fulfillment capabilities through a simple web service interface. Merchants can send order information to Amazon to fulfill customer orders on their behalf. In July 2010, Amazon offered MPI clusters and cluster compute instances. The AWS cluster compute instances use hardware-assisted virtualization instead of the para-virtualization used by other instance types and requires booting from the EBS. Users are freed to create a new AMI as needed.

#### 4.4.4 Microsoft Windows Azure

In 2008, Microsoft launched a Windows Azure platform to meet the challenges in cloud computing. This platform is built over Microsoft data centers. [Figure 4.22](#) shows the overall architecture of Microsoft's cloud platform. The platform is divided into three major component platforms. Windows Azure offers a cloud platform built on Windows OS and based on Microsoft virtualization technology. Applications are installed on VMs deployed on the data-center servers. Azure manages all servers, storage, and network resources of the data center. On top of the infrastructure are the various services for building different cloud applications. Cloud-level services provided by the Azure platform are introduced below. More details on Azure services are given in [Chapter 6](#).



**FIGURE 4.22** Microsoft Windows Azure platform for cloud computing Courtesy of Microsoft, 2010,  
<http://www.microsoft.com/windowsazure>

- **Live service** Users can visit Microsoft Live applications and apply the data involved across multiple machines concurrently.
- **.NET service** This package supports application development on local hosts and execution on cloud machines.
- **SQL Azure** This function makes it easier for users to visit and use the relational database associated with the SQL server in the cloud.
- **SharePoint service** This provides a scalable and manageable platform for users to develop their special business applications in upgraded web services.
- **Dynamic CRM service** This provides software developers a business platform in managing CRM applications in financing, marketing, and sales and promotions.

All these cloud services in Azure can interact with traditional Microsoft software applications, such as Windows Live, Office Live, Exchange online, SharePoint online, and dynamic CRM online. The Azure platform applies the standard web communication protocols SOAP and REST. The Azure service applications allow users to integrate the cloud application with other platforms or third-party clouds. You can download the Azure development kit to run a local version of Azure. The powerful SDK allows Azure applications to be developed and debugged on the Windows hosts.

## 4.5 Inter-cloud Resource Management

This section characterizes the various cloud service models and their extensions. The cloud service trends are outlined. Cloud resource management and intercloud resource exchange schemes are reviewed. We will discuss the defense of cloud resources against network threats in Section 4.6.

### 4.5.1 Extended Cloud Computing Services

Figure 4.23 shows six layers of cloud services, ranging from hardware, network, and collocation to infrastructure, platform, and software applications. We already introduced the top three service layers as SaaS, PaaS, and IaaS, respectively. The cloud platform provides PaaS, which sits on top of the IaaS infrastructure. The top layer offers SaaS. These must be implemented on the cloud platforms provided. Although the three basic models are dissimilar in usage, as shown in Table 4.7, they are built one on top of another. The implication is that one cannot launch SaaS applications with a cloud platform. The cloud platform cannot be built if compute and storage infrastructures are not there.

Cloud application (SaaS)			Concur, RightNOW, Teleo, Kenexa, Webex, Blackbaud, salesforce.com, Netsuite, Kenexa, etc.
Cloud software environment (PaaS)			Force.com, App Engine, Facebook, MS Azure, NetSuite, IBM BlueCloud, SGI Cyclone, eBay
Cloud software infrastructure			Amazon AWS, OpSource Cloud, IBM Ensembles, Rackspace cloud, Windows Azure, HP, Banknorth
Computational resources (IaaS)	Storage (DaaS)	Communications (CaaS)	
Collocation cloud services (LaaS)			Savvis, Internap, NTT Communications, Digital Realty Trust, 365 Main
Network cloud services (NaaS)			Owest, AT&T, AboveNet
Hardware/Virtualization cloud services (HaaS)			VMware, Intel, IBM, XenEnterprise

**FIGURE 4.23** A stack of six layers of cloud services and their providers Courtesy of T. Chou, Active Book Express, 2010 [16]

The bottom three layers are more related to physical requirements. The bottommost layer provides *Hardware as a Service (HaaS)*. The next layer is for interconnecting all the hardware components, and is simply called *Network as a Service (NaaS)*. Virtual LANs fall within the scope of NaaS. The next layer up offers *Location as a Service (LaaS)*, which provides a collocation service to house, power, and secure all the physical hardware and network resources. Some authors say this layer provides *Security as a Service ("SaaS")*. The cloud infrastructure layer can be further subdivided as *Data as a Service (DaaS)* and *Communication as a Service (CaaS)* in addition to compute and storage in IaaS.

We will examine commercial trends in cloud services in subsequent sections. Here we will mainly cover the top three layers with some success stories of cloud computing. As shown in Table 4.7, cloud players are divided into three classes: (1) cloud service providers and IT administrators, (2) software developers or vendors, and (3) end users or business users. These cloud players vary in their roles under the IaaS, PaaS, and SaaS models. The table entries distinguish the three cloud models as viewed by different players. From the software vendors' perspective, application performance on a given cloud platform is most important. From the providers' perspective, cloud infrastructure performance is the primary concern. From the end users' perspective, the quality of services, including security, is the most important.

**Table 4.7****Cloud Differences in Perspectives of Providers, Vendors, and Users**

<b>Cloud Players</b>	<b>IaaS</b>	<b>PaaS</b>	<b>SaaS</b>
IT administrators/cloud providers	Monitor SLAs	Monitor SLAs and enable service platforms	Monitor SLAs and deploy software
Software developers (vendors)	To deploy and store data	Enabling platforms via configurators and APIs	Develop and deploy software
End users or business users	To deploy and store data	To develop and test web software	Use business software

**4.5.1.1 Cloud Service Tasks and Trends**

Cloud services are introduced in five layers. The top layer is for SaaS applications, as further subdivided into the five application areas in [Figure 4.23](#), mostly for business applications. For example, CRM is heavily practiced in business promotion, direct sales, and marketing services. CRM offered the first SaaS on the cloud successfully. The approach is to widen market coverage by investigating customer behaviors and revealing opportunities by statistical analysis. SaaS tools also apply to distributed collaboration, and financial and human resources management. These cloud services have been growing rapidly in recent years.

PaaS is provided by Google, [Salesforce.com](#), and Facebook, among others. IaaS is provided by Amazon, Windows Azure, and RackRack, among others. Collocation services require multiple cloud providers to work together to support supply chains in manufacturing. Network cloud services provide communications such as those by AT&T, Qwest, and AboveNet. Details can be found in Clou's introductory book on business clouds [18]. The vertical cloud services in [Figure 4.25](#) refer to a sequence of cloud services that are mutually supportive. Often, cloud mashup is practiced in vertical cloud applications.

**4.5.1.2 Software Stack for Cloud Computing**

Despite the various types of nodes in the cloud computing cluster, the overall software stacks are built from scratch to meet rigorous goals (see [Table 4.7](#)). Developers have to consider how to design the system to meet critical requirements such as high throughput, HA, and fault tolerance. Even the operating system might be modified to meet the special requirement of cloud data processing. Based on the observations of some typical cloud computing instances, such as Google, Microsoft, and Yahoo!, the overall software stack structure of cloud computing software can be viewed as layers. Each layer has its own purpose and provides the interface for the upper layers just as the traditional software stack does. However, the lower layers are not completely transparent to the upper layers.

The platform for running cloud computing services can be either physical servers or virtual servers. By using VMs, the platform can be flexible, that is, the running services

are not bound to specific hardware platforms. This brings flexibility to cloud computing platforms. The software layer on top of the platform is the layer for storing massive amounts of data. This layer acts like the file system in a traditional single machine. Other layers running on top of the file system are the layers for executing cloud computing applications. They include the database storage system, programming for large-scale clusters, and data query language support. The next layers are the components in the software stack.

#### **4.5.1.3 Runtime Support Services**

As in a cluster environment, there are also some runtime supporting services in the cloud computing environment. Cluster monitoring is used to collect the runtime status of the entire cluster. One of the most important facilities is the cluster job management system introduced in [Chapter 2](#). The scheduler queues the tasks submitted to the whole cluster and assigns the tasks to the processing nodes according to node availability. The distributed scheduler for the cloud application has special characteristics that can support cloud applications, such as scheduling the programs written in MapReduce style. The runtime support system keeps the cloud cluster working properly with high efficiency.

Runtime support is software needed in browser-initiated applications applied by thousands of cloud customers. The SaaS model provides the software applications as a service, rather than letting users purchase the software. As a result, on the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are rather low, compared with conventional hosting of user applications. The customer data is stored in the cloud that is either vendor proprietary or a publicly hosted cloud supporting PaaS and IaaS.

### **4.5.2 Resource Provisioning and Platform Deployment**

The emergence of computing clouds suggests fundamental changes in software and hardware architecture. Cloud architecture puts more emphasis on the number of processor cores or VM instances. Parallelism is exploited at the cluster node level. In this section, we will discuss techniques to provision computer resources or VMs. Then we will talk about storage allocation schemes to interconnect distributed computing infrastructures by harnessing the VMs dynamically.

#### **4.5.2.1 Provisioning of Compute Resources (VMs)**

Providers supply cloud services by signing SLAs with end users. The SLAs must commit sufficient resources such as CPU, memory, and bandwidth that the user can use for a preset period. Underprovisioning of resources will lead to broken SLAs and penalties. Overprovisioning of resources will lead to resource underutilization, and consequently, a decrease in revenue for the provider. Deploying an autonomous system to efficiently provision resources to users is a challenging problem. The difficulty comes from the unpredictability of consumer demand, software and hardware failures, heterogeneity of

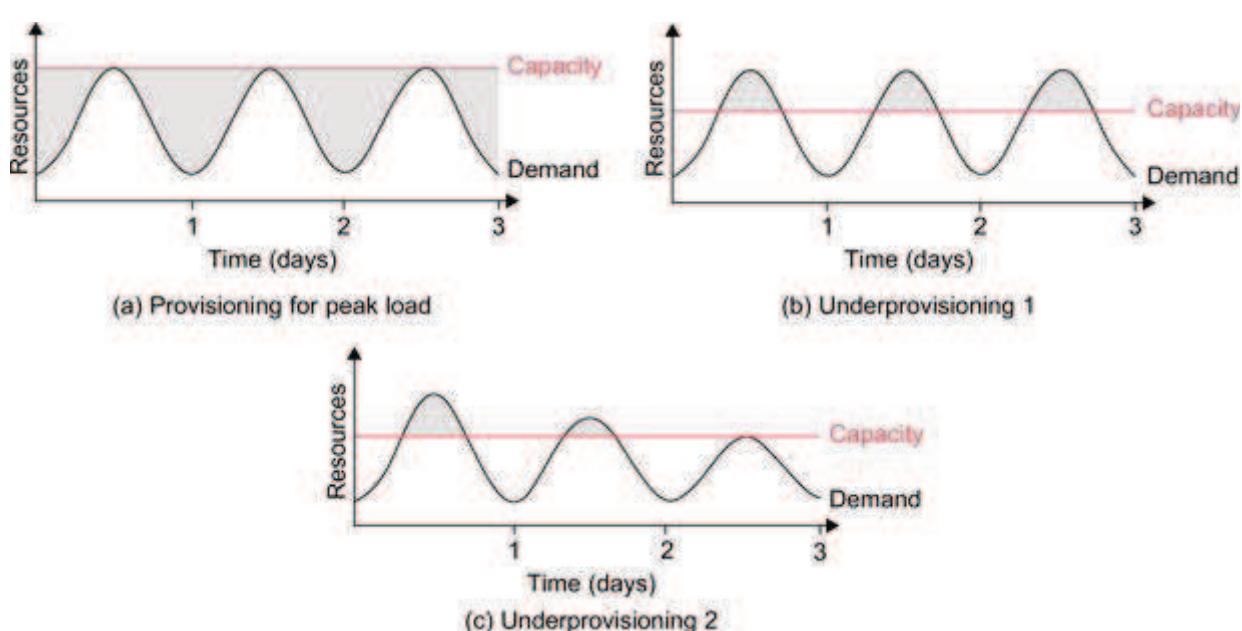
services, power management, and conflicts in signed SLAs between consumers and service providers.

Efficient VM provisioning depends on the cloud architecture and management of cloud infrastructures. Resource provisioning schemes also demand fast discovery of services and data in cloud computing infrastructures. In a virtualized cluster of servers, this demands efficient installation of VMs, live VM migration, and fast recovery from failures. To deploy VMs, users treat them as physical hosts with customized operating systems for specific applications. For example, Amazon's EC2 uses Xen as the virtual machine monitor (VMM). The same VMM is used in IBM's Blue Cloud.

In the EC2 platform, some predefined VM templates are also provided. Users can choose different kinds of VMs from the templates. IBM's Blue Cloud does not provide any VM templates. In general, any type of VM can run on top of Xen. Microsoft also applies virtualization in its Azure cloud platform. The provider should offer resource-economic services. Power-efficient schemes for caching, query processing, and thermal management are mandatory due to increasing energy waste by heat dissipation from data centers. Public or private clouds promise to streamline the on-demand provisioning of software, hardware, and data as a service, achieving economies of scale in IT deployment and operation.

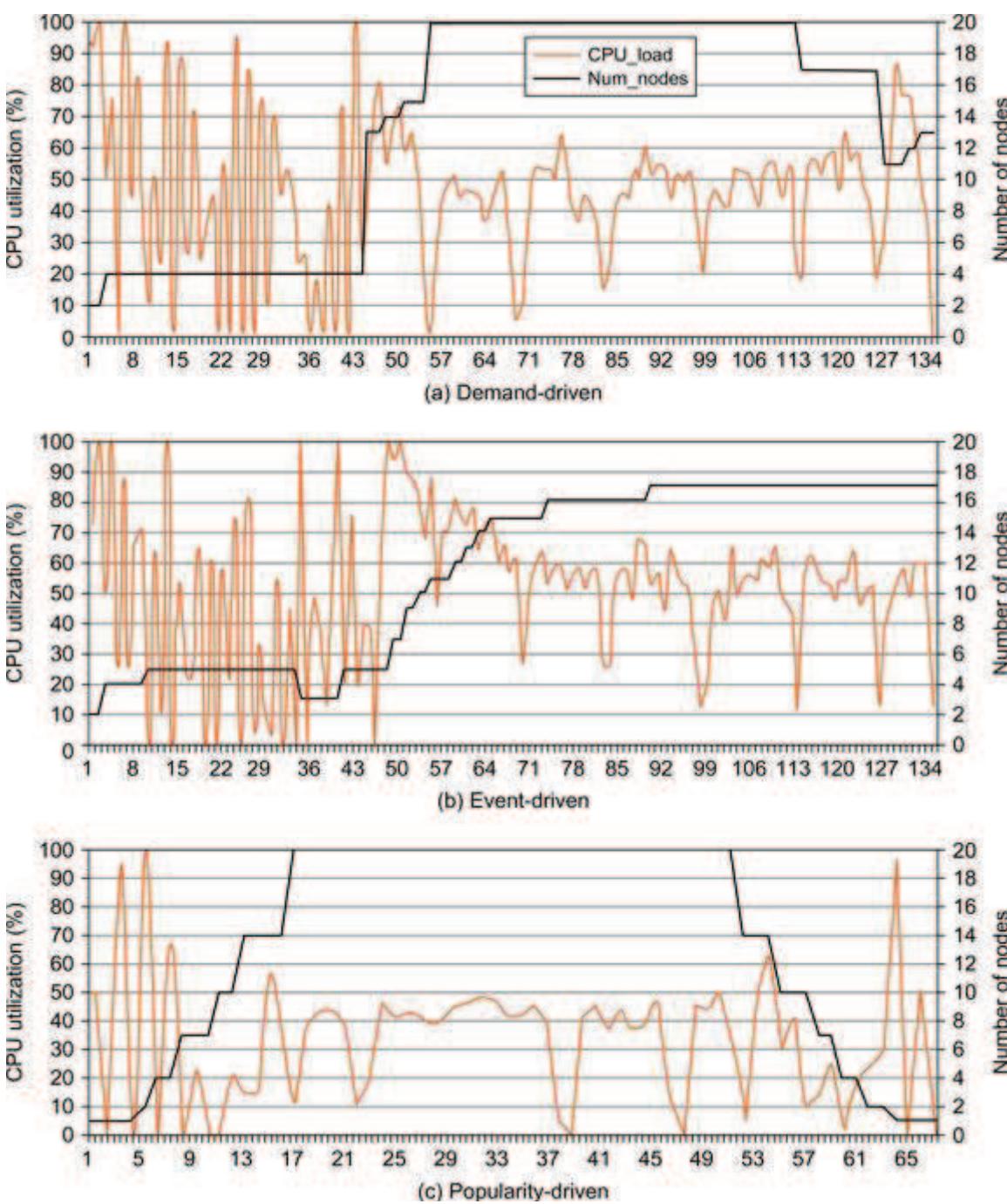
#### **4.5.2.2 Resource Provisioning Methods**

Figure 4.24 shows three cases of static cloud resource provisioning policies. In case (a), overprovisioning with the peak load causes heavy resource waste (shaded area). In case (b), underprovisioning (along the capacity line) of resources results in losses by both user and provider in that paid demand by the users (the shaded area above the capacity) is not served and wasted resources still exist for those demanded areas below the provisioned capacity. In case (c), the constant provisioning of resources with fixed capacity to a declining user demand could result in even worse resource waste. The user may give up the service by canceling the demand, resulting in reduced revenue for the provider. Both the user and provider may be losers in resource provisioning without elasticity.



**FIGURE 4.24** Three cases of cloud resource provisioning without elasticity: (a) heavy waste due to overprovisioning, (b) underprovisioning and (c) under- and then overprovisioning Courtesy of Armbrust, et al., UC Berkeley, 2009 [4]

Three resource-provisioning methods are presented in the following sections. The *demand-driven method* provides static resources and has been used in grid computing for many years. The *event-driven method* is based on predicted workload by time. The *popularity-driven method* is based on Internet traffic monitored. We characterize these resource provisioning methods as follows (see Figure 4.25).



**FIGURE 4.25** EC2 performance results on the AWS EC2 platform, collected from experiments at the University of Southern California using three resource provisioning methods Courtesy of Ken Wu, USC

#### 4.5.2.3 Demand-Driven Resource Provisioning

This method adds or removes computing instances based on the current utilization level of the allocated resources. The demand-driven method automatically allocates two Xeon processors for the user application, when the user was using one Xeon processor more than 60 percent of the time for an extended period. In general, when a resource has surpassed a threshold for a certain amount of time, the scheme increases that resource based on demand. When a resource is below a threshold for a certain amount of time, that resource could be decreased accordingly. Amazon implements such an auto-scale feature in its EC2 platform. This method is easy to implement. The scheme does not work out right if the workload changes abruptly.

The x-axis in [Figure 4.25](#) is the time scale in milliseconds. In the beginning, heavy fluctuations of CPU load are encountered. All three methods have demanded a few VM instances initially. Gradually, the utilization rate becomes more stabilized with a maximum of 20 VMs (100 percent utilization) provided for demand-driven provisioning in [Figure 4.25\(a\)](#). However, the event-driven method reaches a stable peak of 17 VMs toward the end of the event and drops quickly in [Figure 4.25\(b\)](#). The popularity provisioning shown in [Figure 4.25\(c\)](#) leads to a similar fluctuation with peak VM utilization in the middle of the plot.

#### **4.5.2.4 Event-Driven Resource Provisioning**

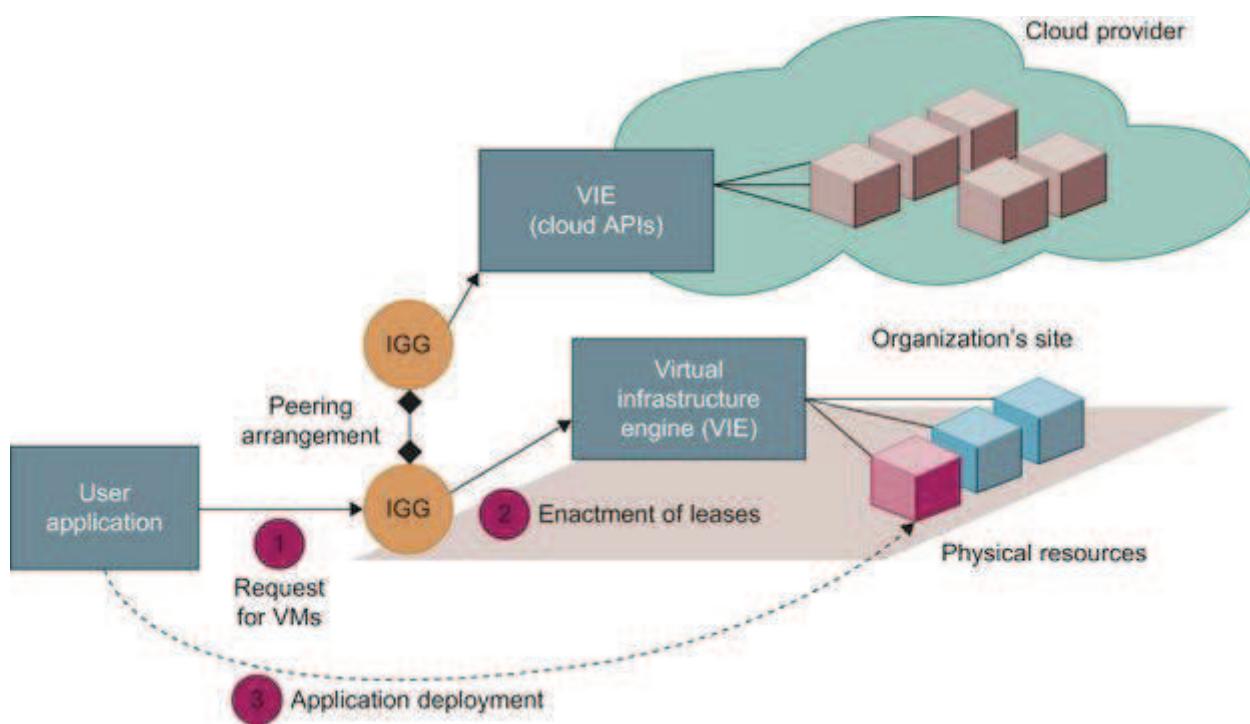
This scheme adds or removes machine instances based on a specific time event. The scheme works better for seasonal or predicted events such as Christmastime in the West and the Lunar New Year in the East. During these events, the number of users grows before the event period and then decreases during the event period. This scheme anticipates peak traffic before it happens. The method results in a minimal loss of QoS, if the event is predicted correctly. Otherwise, wasted resources are even greater due to events that do not follow a fixed pattern.

#### **4.5.2.5 Popularity-Driven Resource Provisioning**

In this method, the Internet searches for popularity of certain applications and creates the instances by popularity demand. The scheme anticipates increased traffic with popularity. Again, the scheme has a minimal loss of QoS, if the predicted popularity is correct. Resources may be wasted if traffic does not occur as expected. In [Figure 4.25\(c\)](#), EC2 performance by CPU utilization rate (the dark curve with the percentage scale shown on the left) is plotted against the number of VMs provisioned (the light curves with scale shown on the right, with a maximum of 20 VMs provisioned).

#### **4.5.2.6 Dynamic Resource Deployment**

The cloud uses VMs as building blocks to create an execution environment across multiple resource sites. The InterGrid-managed infrastructure was developed by a Melbourne University group [\[19\]](#). Dynamic resource deployment can be implemented to achieve scalability in performance. The InterGrid is a Java-implemented software system that lets users create execution cloud environments on top of all participating grid resources. Peering arrangements established between gateways enable the allocation of resources from multiple grids to establish the execution environment. In [Figure 4.26](#), a scenario is illustrated by which an *intergrid gateway* (IGG) allocates resources from a local cluster to deploy applications in three steps: (1) requesting the VMs, (2) enacting the leases, and (3) deploying the VMs as requested. Under peak demand, this IGG interacts with another IGG that can allocate resources from a cloud computing provider.



**FIGURE 4.26** Cloud resource deployment using an IGG (intergrid gateway) to allocate the VMs from a Local cluster to interact with the IGG of a public cloud provider. Courtesy of Constanzo, et al. © IEEE [21]

A grid has predefined peering arrangements with other grids, which the IGG manages. Through multiple IGGs, the system coordinates the use of InterGrid resources. An IGG is aware of the peering terms with other grids, selects suitable grids that can provide the required resources, and replies to requests from other IGGs. Request redirection policies determine which peering grid InterGrid selects to process a request and a price for which that grid will perform the task. An IGG can also allocate resources from a cloud provider. The cloud system creates a virtual environment to help users deploy their applications. These applications use the distributed grid resources.

The InterGrid allocates and provides a *distributed virtual environment* (DVE). This is a virtual cluster of VMs that runs isolated from other virtual clusters. A component called the *DVE manager* performs resource allocation and management on behalf of specific user applications. The core component of the IGG is a scheduler for implementing provisioning policies and peering with other gateways. The communication component provides an asynchronous message-passing mechanism. Received messages are handled in parallel by a thread pool.

#### 4.5.2.7 Provisioning of Storage Resources

The data storage layer is built on top of the physical or virtual servers. As the cloud computing applications often provide service to users, it is unavoidable that the data is stored in the clusters of the cloud provider. The service can be accessed anywhere in the world. One example is e-mail systems. A typical large e-mail system might have millions of users and each user can have thousands of e-mails and consume multiple gigabytes of disk space. Another example is a web searching application. In storage technologies, hard disk drives may be augmented with solid-state drives in the future. This will provide reliable and high-performance data storage. The biggest barriers to adopting flash

memory in data centers have been price, capacity, and, to some extent, a lack of sophisticated query-processing techniques. However, this is about to change as the I/O bandwidth of solid-state drives becomes too impressive to ignore.

A distributed file system is very important for storing large-scale data. However, other forms of data storage also exist. Some data does not need the namespace of a tree structure file system, and instead, databases are built with stored data files. In cloud computing, another form of data storage is (Key, Value) pairs. Amazon S3 service uses SOAP to access the objects stored in the cloud. [Table 4.8](#) outlines three cloud storage services provided by Google, Hadoop, and Amazon.

**Table 4.8**

### Storage Services in Three Cloud Computing Systems

Storage System	Features
GFS: Google File System	Very large sustainable reading and writing bandwidth, mostly continuous accessing instead of random accessing. The programming interface is similar to that of the POSIX file system accessing interface.
HDFS: Hadoop Distributed File System	The open source clone of GFS. Written in Java. The programming interfaces are similar to POSIX but not identical.
Amazon S3 and EBS	S3 is used for retrieving and storing data from/to remote servers. EBS is built on top of S3 for using virtual disks in running EC2 instances.

Many cloud computing companies have developed large-scale data storage systems to keep huge amount of data collected every day. For example, Google's GFS stores web data and some other data, such as geographic data for Google Earth. A similar system from the open source community is the Hadoop Distributed File System (HDFS) for Apache. Hadoop is the open source implementation of Google's cloud computing infrastructure. Similar systems include Microsoft's Cosmos file system for the cloud.

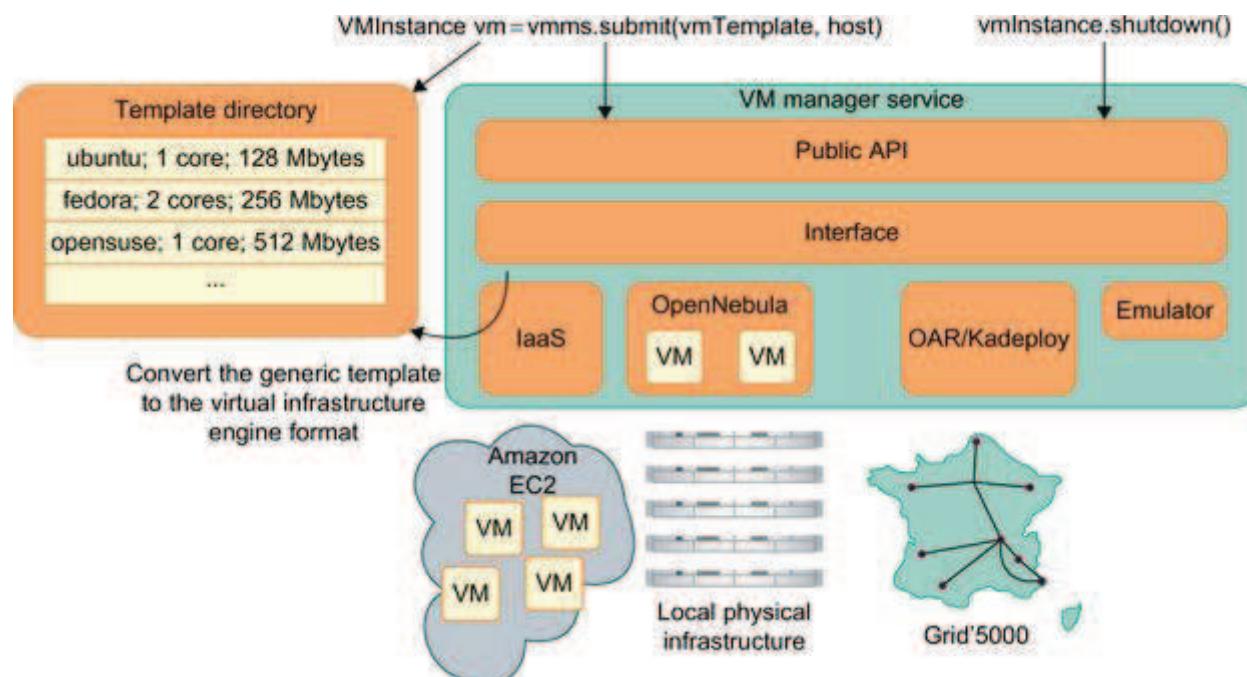
Despite the fact that the storage service or distributed file system can be accessed directly, similar to traditional databases, cloud computing does provide some forms of structure or semistructure database processing capability. For example, applications might want to process the information contained in a web page. Web pages are an example of semistructural data in HTML format. If some forms of database capability can be used, application developers will construct their application logic more easily. Another reason to build a database-like service in cloud computing is that it will be quite convenient for traditional application developers to code for the cloud platform. Databases are quite common as the underlying storage device for many applications.

Thus, such developers can think in the same way they do for traditional software development. Hence, in cloud computing, it is necessary to build databases like large-scale systems based on data storage or distributed file systems. The scale of such a database might be quite large for processing huge amounts of data. The main purpose is to store the data in structural or semi-structural ways so that application developers can use it easily and build their applications rapidly. Traditional databases will meet the performance bottleneck while the system is expanded to a larger scale. However, some

real applications do not need such strong consistency. The scale of such databases can be quite large. Typical cloud databases include BigTable from Google, SimpleDB from Amazon, and the SQL service from Microsoft Azure.

## 4.5.3 Virtual Machine Creation and Management

In this section, we will consider several issues for cloud infrastructure management. First, we will consider the resource management of independent service jobs. Then we will consider how to execute third-party cloud applications. Cloud-loading experiments are used by a Melbourne research group on the French Grid'5000 system. This experimental setting illustrates VM creation and management. This case study example reveals major VM management issues and suggests some plausible solutions for workload-balanced execution. Figure 4.27 shows the interactions among VM managers for cloud creation and management. The managers provide a public API for users to submit and control the VMs.



**FIGURE 4.27** Interactions among VM managers for cloud creation and management; the manager provides a public API for users to submit and control the VMs Courtesy of Constanzo, et al. © IEEE [21]

### 4.5.3.1 Independent Service Management

Independent services request facilities to execute many unrelated tasks. Commonly, the APIs provided are some web services that the developer can use conveniently. In Amazon cloud computing infrastructure, SQS is constructed for providing a reliable communication service between different providers. Even the endpoint does not run while another entity has posted a message in SQS. By using independent service providers, the cloud applications can run different services at the same time. Some other services are used for providing data other than the compute or storage services.

### **4.5.3.2 Running Third-Party Applications**

Cloud platforms have to provide support for building applications that are constructed by third-party application providers or programmers. As current web applications are often provided by using Web 2.0 forms (interactive applications with Ajax), the programming interfaces are different from the traditional programming interfaces such as functions in runtime libraries. The APIs are often in the form of services. Web service application engines are often used by programmers for building applications. The web browsers are the user interface for end users.

In addition to gateway applications, the cloud computing platform provides the extra capabilities of accessing backend services or underlying data. As examples, GAE and Microsoft Azure apply their own cloud APIs to get special cloud services. The WebSphere application engine is deployed by IBM for Blue Cloud. It can be used to develop any kind of web application written in Java. In EC2, users can use any kind of application engine that can run in VM instances.

### **4.5.3.3 Virtual Machine Manager**

The VM manager is the link between the gateway and resources. The gateway doesn't share physical resources directly, but relies on virtualization technology for abstracting them. Hence, the actual resources it uses are VMs. The manager manages VMs deployed on a set of physical resources. The VM manager implementation is generic so that it can connect with different VIEs. Typically, VIEs can create and stop VMs on a physical cluster. The Melbourne group has developed managers for OpenNebula, Amazon EC2, and French Grid'5000. The manager using the OpenNebula OS ([www.opennebula.org](http://www.opennebula.org)) to deploy VMs on local clusters.

OpenNebula runs as a daemon service on a master node, so the VMM works as a remote user. Users submit VMs on physical machines using different kinds of hypervisors, such as Xen ([www.xen.org](http://www.xen.org)), which enables the running of several operating systems on the same host concurrently. The VMM also manages VM deployment on grids and IaaS providers. The InterGrid supports Amazon EC2. The connector is a wrapper for the command-line tool Amazon provides. The VM manager for Grid'5000 is also a wrapper for its command-line tools. To deploy a VM, the manager needs to use its template.

### **4.5.3.4 Virtual Machine Templates**

A VM *template* is analogous to a computer's configuration and contains a description for a VM with the following static information:

- The number of cores or processors to be assigned to the VM
- The amount of memory the VM requires
- The kernel used to boot the VM's operating system
- The disk image containing the VM's file system
- The price per hour of using a VM

The gateway administrator provides the VM template information when the

infrastructure is set up. The administrator can update, add, and delete templates at any time. In addition, each gateway in the InterGrid network must agree on the templates to provide the same configuration on each site. To deploy an instance of a given VM, the VMM generates a descriptor from the template. This descriptor contains the same fields as the template and additional information related to a specific VM instance. Typically the additional information includes:

- The disk image that contains the VM's file system
- The address of the physical machine hosting the VM
- The VM's network configuration
- The required information for deployment on an IaaS provider

Before starting an instance, the scheduler gives the network configuration and the host's address; it then allocates MAC and IP addresses for that instance. The template specifies the disk image field. To deploy several instances of the same VM template in parallel, each instance uses a temporary copy of the disk image. Hence, the descriptor contains the path to the copied disk image. The descriptor's fields are different for deploying a VM on an IaaS provider. Network information is not needed, because Amazon EC2 automatically assigns a public IP to the instances. The IGG works with a repository of VM templates, called the *VM template directory*.

#### **4.5.3.5 Distributed VM Management**

Figure 4.30 illustrates the interactions between InterGrid's components. A distributed VM manager makes requests for VMs and queries their status. This manager requests VMs from the gateway on behalf of the user application. The manager obtains the list of requested VMs from the gateway. This list contains a tuple of public IP/private IP addresses for each VM with Secure Shell (SSH) tunnels. Users must specify which VM template they want to use and the number of VM instances needed, the deadline, the wall time, and the address for an alternative gateway.

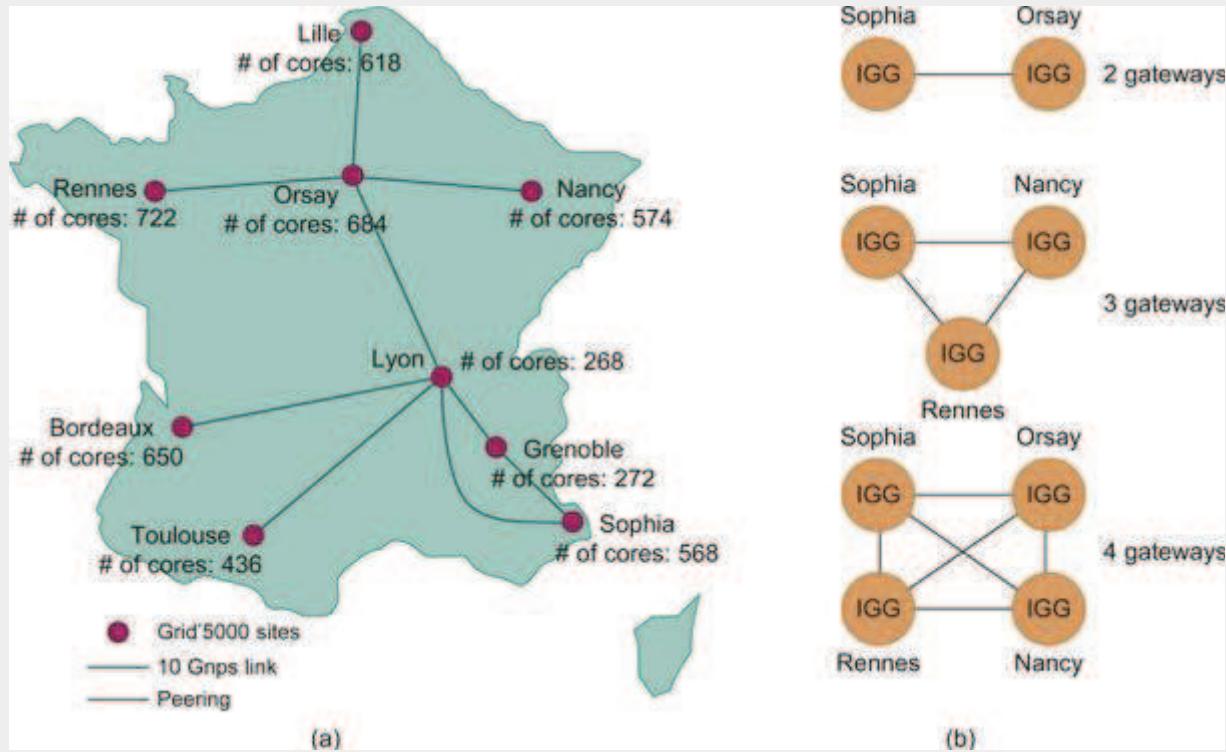
The local gateway tries to obtain resources from the underlying VIEs. When this is impossible, the local gateway starts a negotiation with any remote gateways to fulfill the request. When a gateway schedules the VMs, it sends the VM access information to the requester gateway. Finally, the manager configures the VM, sets up SSH tunnels, and executes the tasks on the VM. Under the peering policy, each gateway's scheduler uses conservative backfilling to schedule requests. When the scheduler can't start a request immediately using local resources, a redirection algorithm will be initiated.

#### **Example 4.6**

#### **Experiments on an InterGrid Test Bed over the Grid'5000**

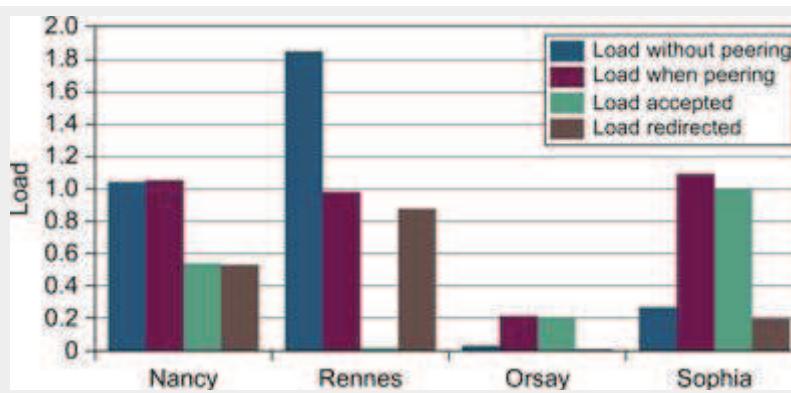
The Melbourne group conducted two experiments to evaluate the InterGrid architecture. The first one evaluates the performance of allocation decisions

by measuring how the IGG manages load via peering arrangements. The second considers its effectiveness in deploying a bag-of-tasks application. The experiment was conducted on the French experimental grid platform Grid'5000. Grid'5000 comprises 4,792 processor cores on nine grid sites across France. Each gateway represents one Grid'5000 site, as shown in Figure 4.28.



**FIGURE 4.28** The InterGrid test bed over the French Grid'5000 located in nine cities across France  
Courtesy of Constanzo, et al. © IEEE [21]

To prevent the gateways from interfering with real Grid'5000 users, emulated VM managers were implemented to instantiate fictitious VMs. The number of emulated hosts is limited by the core number at each site. A balanced workload was configured among the sites. The maximum number of VMs requested does not exceed the number of cores in any site. The load characteristics are shown in Figure 4.29 under a four-gateway scenario. The teal bars indicate each grid site's load. The magenta bars show the load when gateways redirect requests to one another. The green bars correspond to the amount of load each gateway accepts from other gateways. The brown bars represent the amount of load that is redirected. The results show that the loading policy can balance the load across the nine sites. Rennes, a site with a heavy load, benefits from peering with other gateways as the gateway redirects a great share of its load to other sites.

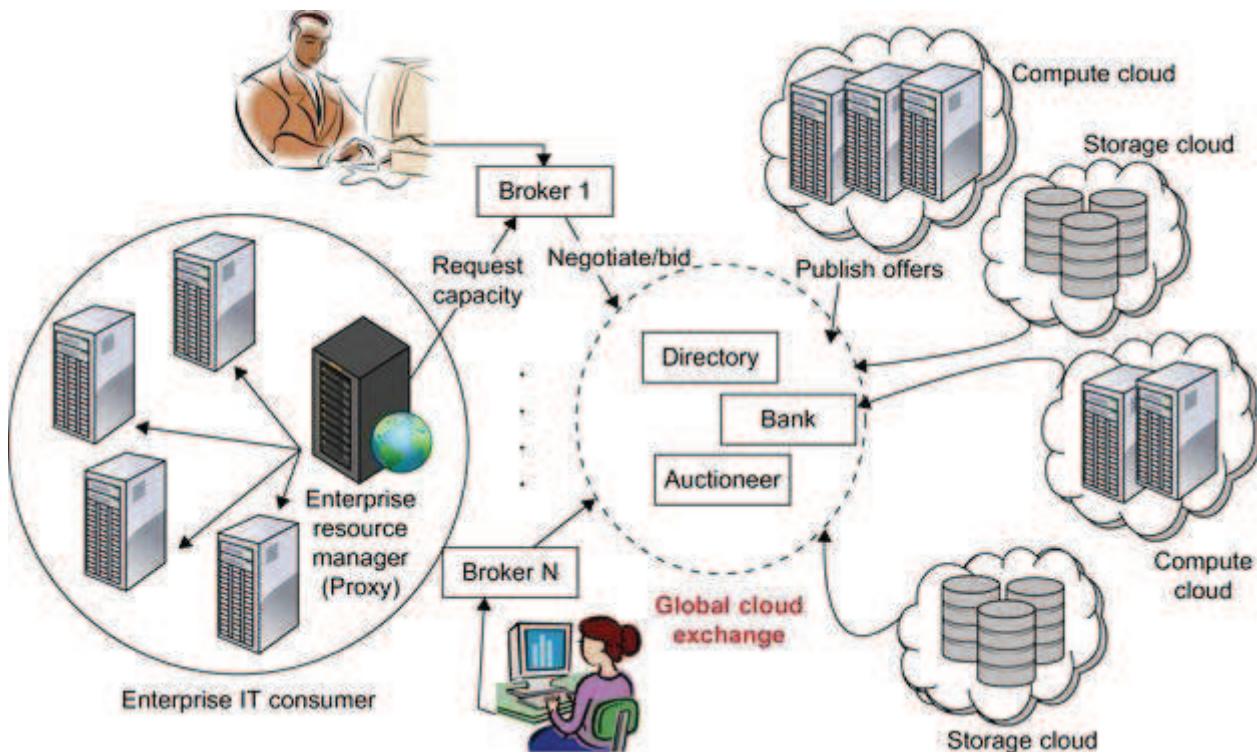


**FIGURE 4.29** Cloud loading results at four gateways at resource sites in the Grid'5000 system  
Courtesy of Constanzo, et al. © IEEE [21]

#### 4.5.4 Global Exchange of Cloud Resources

In order to support a large number of application service consumers from around the world, cloud infrastructure providers (i.e., IaaS providers) have established data centers in multiple geographical locations to provide redundancy and ensure reliability in case of site failures. For example, Amazon has data centers in the United States (e.g., one on the East Coast and another on the West Coast) and Europe. However, currently Amazon expects its cloud customers (i.e., SaaS providers) to express a preference regarding where they want their application services to be hosted. Amazon does not provide seamless/automatic mechanisms for scaling its hosted services across multiple geographically distributed data centers.

This approach has many shortcomings. First, it is difficult for cloud customers to determine in advance the best location for hosting their services as they may not know the origin of consumers of their services. Second, SaaS providers may not be able to meet the QoS expectations of their service consumers originating from multiple geographical locations. This necessitates building mechanisms for seamless federation of data centers of a cloud provider or providers supporting dynamic scaling of applications across multiple domains in order to meet QoS targets of cloud customers. [Figure 4.30](#) shows the high-level components of the Melbourne group's proposed InterCloud architecture.



**FIGURE 4.30** Inter-cloud exchange of cloud resources through brokering Courtesy of R. Buyya, et al., University of Melbourne [12]

In addition, no single cloud infrastructure provider will be able to establish its data centers at all possible locations throughout the world. As a result, cloud application service (SaaS) providers will have difficulty in meeting QoS expectations for all their consumers. Hence, they would like to make use of services of multiple cloud infrastructure service providers who can provide better support for their specific consumer needs. This kind of requirement often arises in enterprises with global operations and applications such as Internet services, media hosting, and Web 2.0 applications. This necessitates federation of cloud infrastructure service providers for seamless provisioning of services across different cloud providers. To realize this, the Cloudbus Project at the University of Melbourne has proposed InterCloud architecture [12] supporting brokering and exchange of cloud resources for scaling applications across multiple clouds.

By realizing InterCloud architectural principles in mechanisms in their offering, cloud providers will be able to dynamically expand or resize their provisioning capability based on sudden spikes in workload demands by leasing available computational and storage capabilities from other cloud service providers; operate as part of a market-driven resource leasing federation, where application service providers such as [Salesforce.com](#) host their services based on negotiated SLA contracts driven by competitive market prices; and deliver on-demand, reliable, cost-effective, and QoS-aware services based on virtualization technologies while ensuring high QoS standards and minimizing service costs. They need to be able to utilize market-based utility models as the basis for provisioning of virtualized software services and federated hardware infrastructure among users with heterogeneous applications.

They consist of client brokering and coordinator services that support utility-driven

federation of clouds: application scheduling, resource allocation, and migration of workloads. The architecture cohesively couples the administratively and topologically distributed storage and compute capabilities of clouds as part of a single resource leasing abstraction. The system will ease the cross-domain capability integration for on-demand, flexible, energy-efficient, and reliable access to the infrastructure based on virtualization technology [6,75].

The *Cloud Exchange (CEx)* acts as a market maker for bringing together service producers and consumers. It aggregates the infrastructure demands from application brokers and evaluates them against the available supply currently published by the cloud coordinators. It supports trading of cloud services based on competitive economic models such as commodity markets and auctions. CEx allows participants to locate providers and consumers with fitting offers. Such markets enable services to be commoditized, and thus will pave the way for creation of dynamic market infrastructure for trading based on SLAs. An SLA specifies the details of the service to be provided in terms of metrics agreed upon by all parties, and incentives and penalties for meeting and violating the expectations, respectively. The availability of a banking system within the market ensures that financial transactions pertaining to SLAs between participants are carried out in a secure and dependable environment.

## 4.6 Cloud Security and Trust Management

Lacking trust between service providers and cloud users has hindered the universal acceptance of cloud computing as a service on demand. In the past, trust models have been developed to protect mainly e-commerce and online shopping provided by eBay and Amazon. For web and cloud services, trust and security become even more demanding, because leaving user applications completely to the cloud providers has faced strong resistance by most PC and server users. Cloud platforms become worrisome to some users for lack of privacy protection, security assurance, and copyright protection. Trust is a social problem, not a pure technical issue. However, the social problem can be solved with a technical approach.

Common sense dictates that technology can enhance trust, justice, reputation, credit, and assurance in Internet applications. As a virtual environment, the cloud poses new security threats that are more difficult to contain than traditional client and server configurations. To solve these trust problems, a new data-protection model is presented in this section. In many cases, one can extend the trust models for P2P networks and grid systems to protect clouds and data centers.

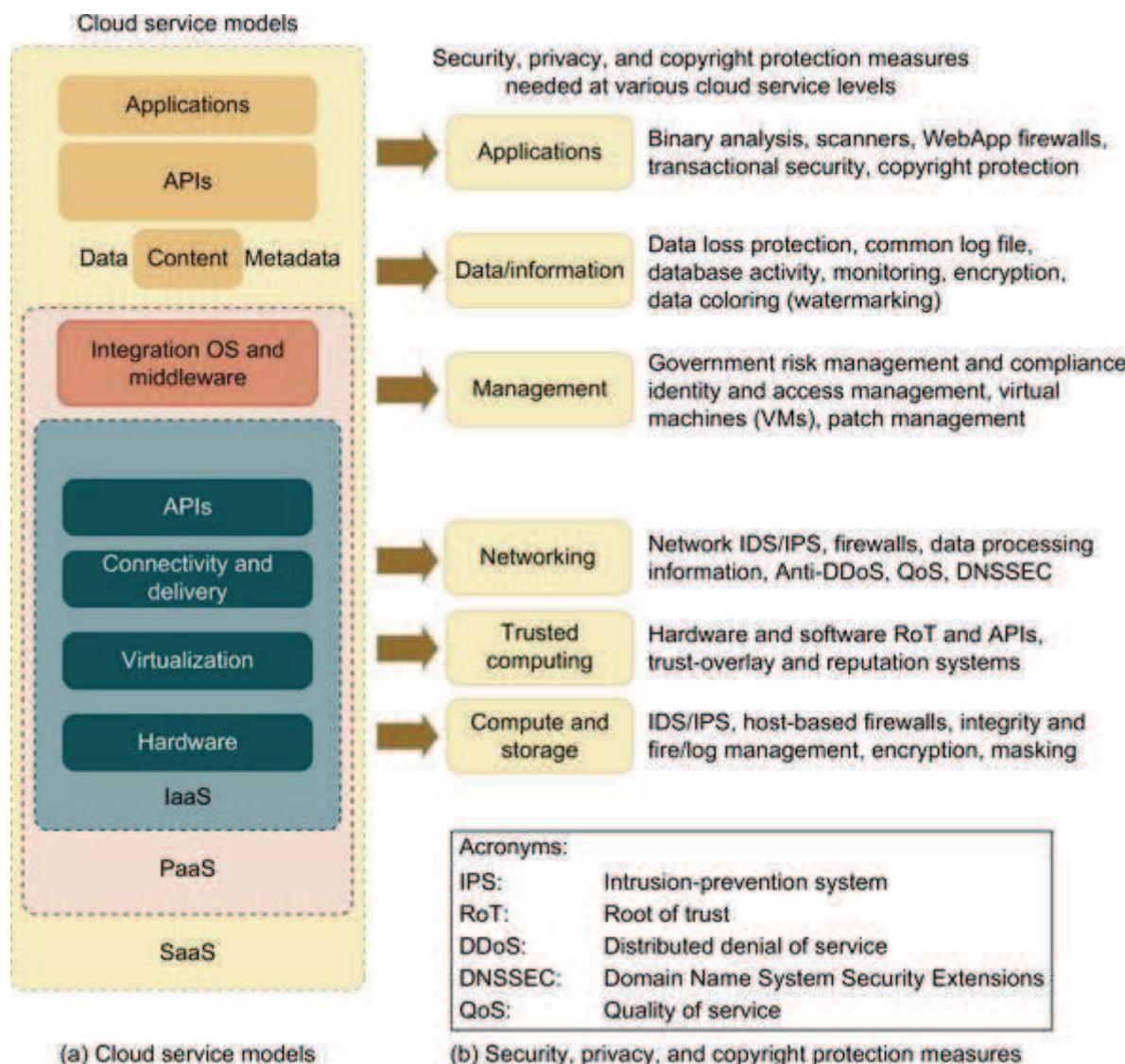
### 4.6.1 Cloud Security Defense Strategies

A healthy cloud ecosystem is desired to free users from abuses, violence, cheating, hacking, viruses, rumors, pornography, spam, and privacy and copyright violations. The security demands of three cloud service models, IaaS, PaaS, and SaaS, are described in this section. These security models are based on various SLAs between providers and

users.

#### 4.6.1.1 Basic Cloud Security

Three basic cloud security enforcements are expected. First, facility security in data centers demands on-site security year round. Biometric readers, CCTV (close-circuit TV), motion detection, and man traps are often deployed. Also, network security demands fault-tolerant external firewalls, intrusion detection systems (IDSe), and third-party vulnerability assessment. Finally, platform security demands SSL and data decryption, strict password policies, and system trust certification. [Figure 4.31](#) shows the mapping of cloud models, where special security measures are deployed at various cloud operating levels.



**FIGURE 4.31** Cloud service models on the left (a) and corresponding security measures on the right (b); the IaaS is at the innermost level, PaaS is at the middle level, and SaaS is at the outermost level, including all hardware, software, datasets, and networking resources Courtesy of Hwang and Li [36]

Servers in the cloud can be physical machines or VMs. User interfaces are applied to request services. The provisioning tool carves out the systems from the cloud to satisfy

the requested service. A security-aware cloud architecture demands security enforcement. Malware-based attacks such as network worms, viruses, and DDoS attacks exploit system vulnerabilities. These attacks compromise system functionality or provide intruders unauthorized access to critical information. Thus, security defenses are needed to protect all cluster servers and data centers. Here are some cloud components that demand special security protection:

- Protection of servers from malicious software attacks such as worms, viruses, and malware
- Protection of hypervisors or VM monitors from software-based attacks and vulnerabilities
- Protection of VMs and monitors from service disruption and DoS attacks
- Protection of data and information from theft, corruption, and natural disasters
- Providing authenticated and authorized access to critical data and services

#### **4.6.1.2 Security Challenges in VMs**

As we discussed earlier in this chapter, traditional network attacks include buffer overflows, DoS attacks, spyware, malware, rootkits, Trojan horses, and worms. In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM rootkits. Another type of attack is the man-in-the-middle attack for VM migrations. In general, passive attacks steal sensitive data or passwords. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers. An IDS can be a NIDS or a HIDS. Program shepherding can be applied to control and verify code execution. Other defense technologies include using the RIO dynamic optimization infrastructure, or VMware's vSafe and vShield tools, security compliance for hypervisors, and Intel vPro technology. Others apply a hardened OS environment or use isolated execution and sandboxing.

#### **4.6.1.3 Cloud Defense Methods**

Virtualization enhances cloud security. But VMs add an additional layer of software that could become a single point of failure. With virtualization, a single physical machine can be divided or partitioned into multiple VMs (e.g., server consolidation). This provides each VM with better security isolation and each partition is protected from DoS attacks by other partitions. Security attacks in one VM are isolated and contained from affecting the other VMs. [Table 4.9](#) lists eight protection schemes to secure public clouds and data centers. VM failures do not propagate to other VMs. The hypervisor provides visibility of the guest OS, with complete guest isolation. Fault containment and failure isolation of VMs provide a more secure and robust environment. Malicious intrusions may destroy valuable hosts, networks, and storage resources. Internet anomalies found in routers, gateways, and distributed hosts may stop cloud services. Trust negotiation is often done at the SLA level. Public Key Infrastructure (PKI) services could be augmented with data-center reputation systems. Worm and DDoS attacks must be contained. It is harder to establish security in the cloud because all data and software are shared by default.

**Table 4.9****Physical and Cyber Security Protection at Cloud/Data Centers**

Protection Schemes	Brief Description and Deployment Suggestions
Secure data centers and computer buildings	Choose hazard-free location, enforce building safety. Avoid windows, keep buffer zone around the site, bomb detection, camera surveillance, earthquake-proof, etc.
Use redundant utilities at multiple sites	Multiple power and supplies, alternate network connections, multiple databases at separate sites, data consistency, data watermarking, user authentication, etc.
Trust delegation and negotiation	Cross certificates to delegate trust across PKI domains for various data centers, trust negotiation among certificate authorities (CAs) to resolve policy conflicts
Worm containment and DDoS defense	Internet worm containment and distributed defense against DDoS attacks to secure all data centers and cloud platforms
Reputation system for data centers	Reputation system could be built with P2P technology; one can build a hierarchy of reputation systems from data centers to distributed file systems
Fine-grained file access control	Fine-grained access control at the file or object level; this adds to security protection beyond firewalls and IDSes
Copyright protection and piracy prevention	Piracy prevention achieved with peer collusion prevention, filtering of poisoned content, nondestructive read, alteration detection, etc.
Privacy protection	Uses double authentication, biometric identification, intrusion detection and disaster recovery, privacy enforcement by data watermarking, data classification, etc.

**4.6.1.4 Defense with Virtualization**

The VM is decoupled from the physical hardware. The entire VM can be represented as a software component and can be regarded as binary or digital data. The VM can be saved, cloned, encrypted, moved, or restored with ease. VMs enable HA and faster disaster recovery. Live migration of VMs was suggested by many researchers [36] for building *distributed intrusion detection systems (DIDSes)*. Multiple IDS VMs can be deployed at various resource sites including data centers. DIDS design demands trust negation among PKI domains. Security policy conflicts must be resolved at design time and updated periodically.

**4.6.1.5 Privacy and Copyright Protection**

The user gets a predictable configuration before actual system integration. Yahoo!'s Pipes is a good example of a lightweight cloud platform. With shared files and data sets, privacy, security, and copyright data could be compromised in a cloud computing environment. Users desire to work in a software environment that provides many useful tools to build cloud applications over large data sets. Google's platform essentially applies in-house software to protect resources. The Amazon EC2 applies HMEC and X.509 certificates in securing resources. It is necessary to protect browser-initiated application software in the cloud environment. Here are several security features desired in a secure cloud:

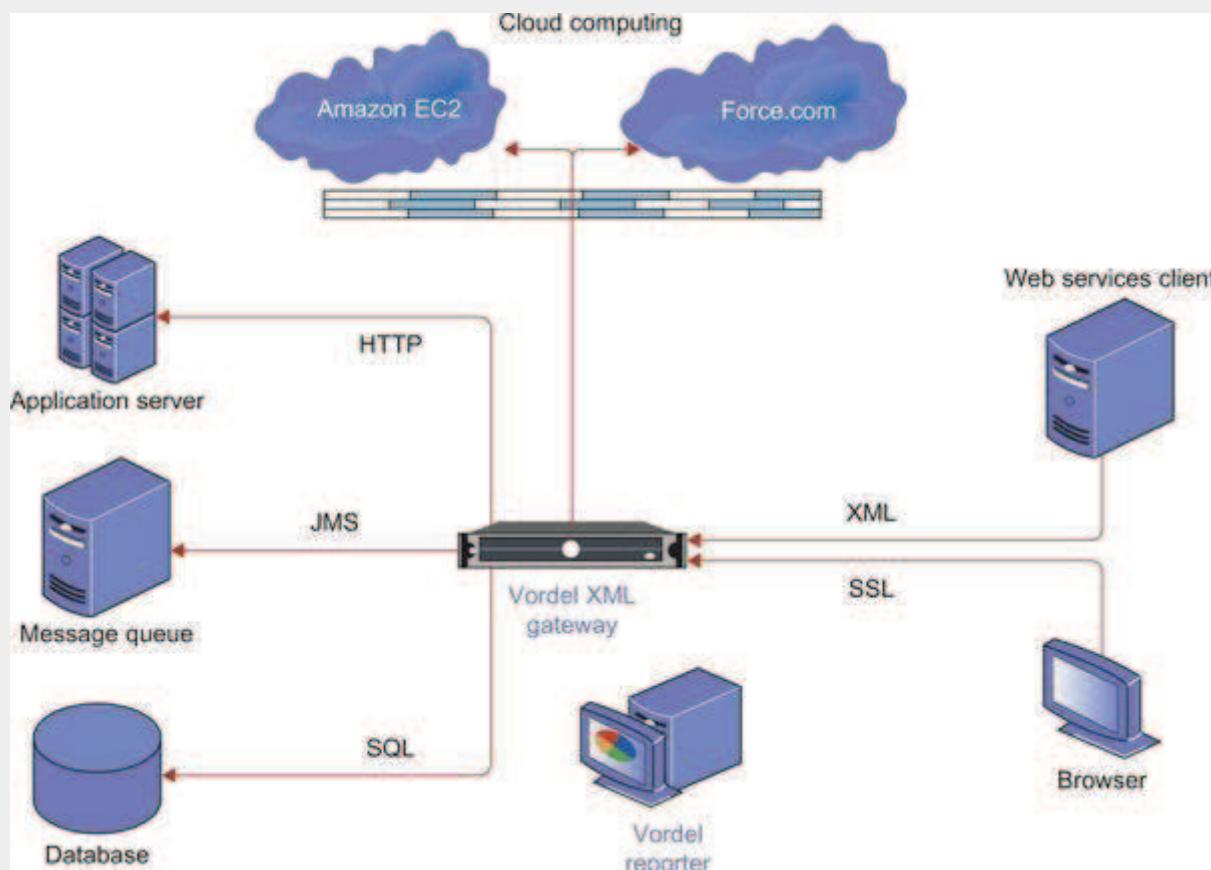
- Dynamic web services with full support from secure web technologies
- Established trust between users and providers through SLAs and reputation systems

- Effective user identity management and data-access management
- Single sign-on and single sign-off to reduce security enforcement overhead
- Auditing and copyright compliance through proactive enforcement
- Shifting of control of data operations from the client environment to cloud providers
- Protection of sensitive and regulated information in a shared environment

### **Example 4.7**

#### **Cloud Security Safeguarded by Gateway and Firewalls**

Figure 4.32 shows a security defense system for a typical private cloud environment. The gateway is fully secured to protect access to commercial clouds that are wide open to the general public. The firewall provides an external shield. The gateway secures the application server, message queue, database, web service client, and browser with HTTP, JMS, SQL, XML, and SSL security protocols, etc. The defense scheme is needed to protect user data from server attacks. A user's private data must not be leaked to other users without permission.



**FIGURE 4.32** The typical security structure coordinated by a secured gateway plus external firewalls to safeguard the access of public or private clouds Courtesy of Vordel Company

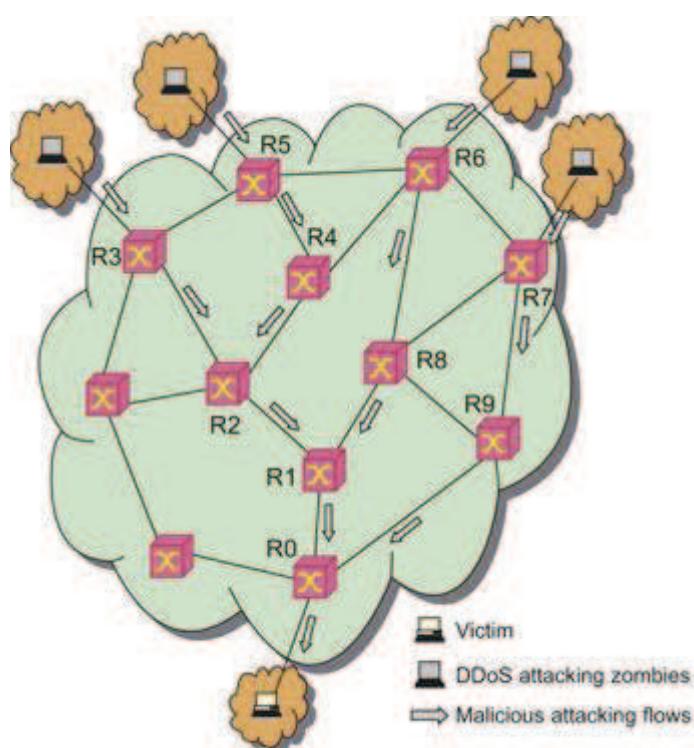
## 4.6.2 Distributed Intrusion/Anomaly Detection

Data security is the weakest link in all cloud models. We need new cloud security standards to apply common API tools to cope with the data lock-in problem and network attacks or abuses. The IaaS model represented by Amazon is most sensitive to external attacks. Role-based interface tools alleviate the complexity of the provisioning system. For example, IBM's Blue Cloud provisions through a role-based web portal. A SaaS bureau may order secretarial services from a common cloud platform. Many IT companies are now offering cloud services with no guaranteed security.

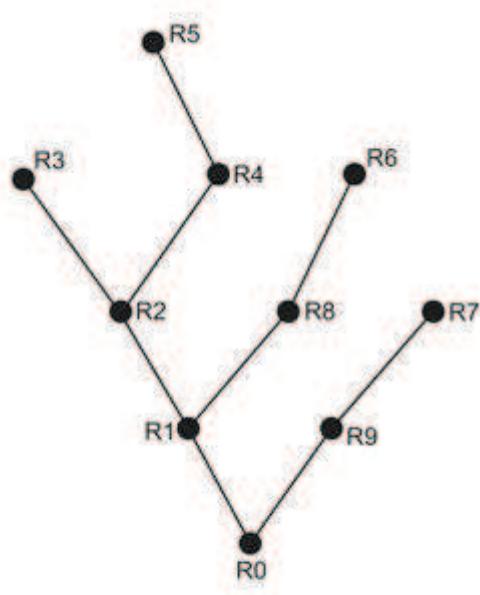
Security threats may be aimed at VMs, guest OSes, and software running on top of the cloud. IDSes attempt to stop these attacks before they take effect. Both signature matching and anomaly detection can be implemented on VMs dedicated to building IDSes. Signature-matching IDS technology is more mature, but require frequent updates of the signature databases. Network anomaly detection reveals abnormal traffic patterns, such as unauthorized episodes of TCP connection sequences, against normal traffic patterns. Distributed IDSes are needed to combat both types of intrusions.

### 4.6.2.1 Distributed Defense against DDoS Flooding Attacks

A DDoS defense system must be designed to cover multiple network domains spanned by a given cloud platform. These network domains cover the edge networks where cloud resources are connected. DDoS attacks come with widespread worms. The flooding traffic is large enough to crash the victim server by buffer overflow, disk exhaustion, or connection saturation. Figure 4.33(a) shows a flooding attack pattern. Here, the hidden attacker launched the attack from many zombies toward a victim server at the bottom router  $R_0$ .



(a) Traffic flow pattern of a DDoS attack



(b) The attack traffic flow tree over 10 routers

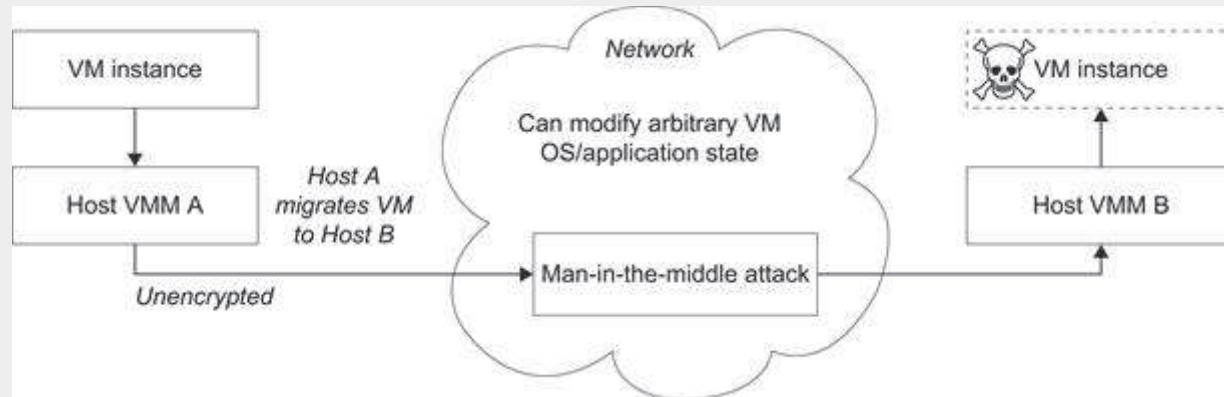
**FIGURE 4.33** DDoS attacks and defense by change-point detection at all routers on the flooding tree Courtesy of Chen, Hwang, and Ku [15]

The flooding traffic flows essentially with a tree pattern shown in Figure 4.33(b). Successive attack-transit routers along the tree reveal the abnormal surge in traffic. This DDoS defense system is based on change-point detection by all routers. Based on the anomaly pattern detected in covered network domains, the scheme detects a DDoS attack before the victim is overwhelmed. The detection scheme is suitable for protecting cloud core networks. The provider-level cooperation eliminates the need for intervention by edge networks.

### Example 4.8

#### Man-in-the-Middle Attacks

Figure 4.34 shows VM migration from host machine VMM A to host machine VMM B, via a security vulnerable network. In a man-in-the-middle attack, the attacker can view the VM contents being migrated, steal sensitive data, or even modify the VM-specific contents including the OS and application states. An attacker posing this attack can launch an active attack to insert a VM-based rootkit into the migrating VM, which can subvert the entire operation of the migration process without the knowledge of the guest OS and embedded application.



**FIGURE 4.34** A VM migrating from host A to host B through a vulnerable network threatened by a man-in-the-middle attack to modify the VM template and OS state.

### 4.6.3 Data and Software Protection Techniques

In this section, we will introduce a data coloring technique to preserve data integrity and

user privacy. Then we will discuss a watermarking scheme to protect software files from being widely distributed in a cloud environment.

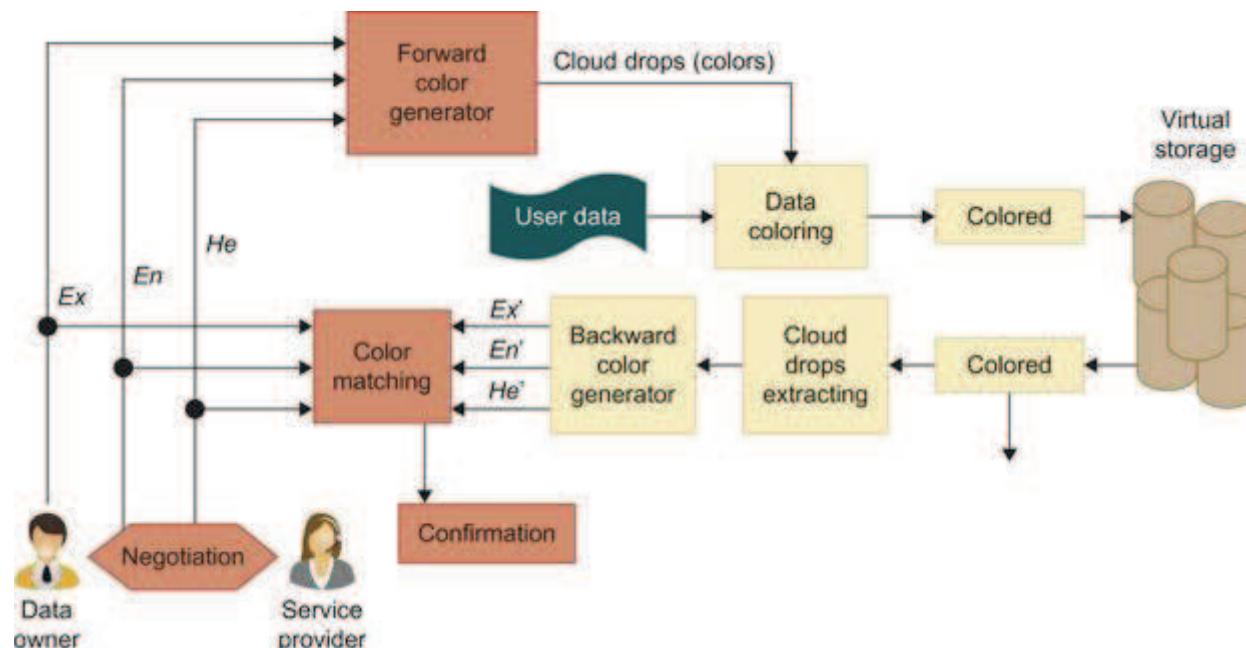
### 4.6.3.1 Data Integrity and Privacy Protection

Users desire a software environment that provides many useful tools to build cloud applications over large data sets. In addition to application software for MapReduce, BigTable, EC2, 3S, Hadoop, AWS, GAE, and WebSphere2, users need some security and privacy protection software for using the cloud. Such software should offer the following features:

- Special APIs for authenticating users and sending e-mail using commercial accounts
- Fine-grained access control to protect data integrity and deter intruders or hackers
- Shared data sets protected from malicious alteration, deletion, or copyright violation
- Ability to secure the ISP or cloud service provider from invading users' privacy
- Personal firewalls at user ends to keep shared data sets from Java, JavaScript, and ActiveX applets
- A privacy policy consistent with the cloud service provider's policy, to protect against identity theft, spyware, and web bugs
- VPN channels between resource sites to secure transmission of critical data objects

### 4.6.3.2 Data Coloring and Cloud Watermarking

With shared files and data sets, privacy, security, and copyright information could be compromised in a cloud computing environment. Users desire to work in a trusted software environment that provides useful tools to build cloud applications over protected data sets. In the past, watermarking was mainly used for digital copyright management. As shown in Figure 4.35, the system generates special colors for each data object. Data coloring means labeling each data object by a unique color. Differently colored data objects are thus distinguishable.



**FIGURE 4.35** Data coloring with cloud watermarking for trust management at various security clearance levels in data centers Courtesy of Hwang and Li [36]

The user identification is also colored to be matched with the data colors. This color matching process can be applied to implement different trust management events. Cloud storage provides a process for the generation, embedding, and extraction of the watermarks in colored objects. Interested readers may refer to the articles by Hwang and Li [36] for details on the data coloring and matching process. In general, data protection was done by encryption or decryption which is computationally expensive. The data coloring takes a minimal number of calculations to color or decolor the data objects. Cryptography and watermarking or coloring can be used jointly in a cloud environment.

### 4.6.3.3 Data Lock-in Problem and Proactive Solutions

Cloud computing moves both the computation and the data to the server clusters maintained by cloud service providers. Once the data is moved into the cloud, users cannot easily extract their data and programs from cloud servers to run on another platform. This leads to a data lock-in problem. This has hindered the use of cloud computing. Data lock-in is attributed to two causes: lack of interoperability, whereby each cloud vendor has its proprietary API that limits users to extract data once submitted; and lack of application compatibility, in that most computing clouds expect users to write new applications from scratch, when they switch cloud platforms.

One possible solution to data lock-in is the use of standardized cloud APIs. This requires building standardized virtual platforms that adhere to OVF, a platform-independent, efficient, extensible, and open format for VMs. This will enable efficient, secure software distribution, facilitating the mobility of VMs. Using OVF one can move data from one application to another. This will enhance QoS, and thus enable cross-cloud applications, allowing workload migration among data centers to user-specific storage. By deploying applications, users can access and intermix applications across different cloud services.

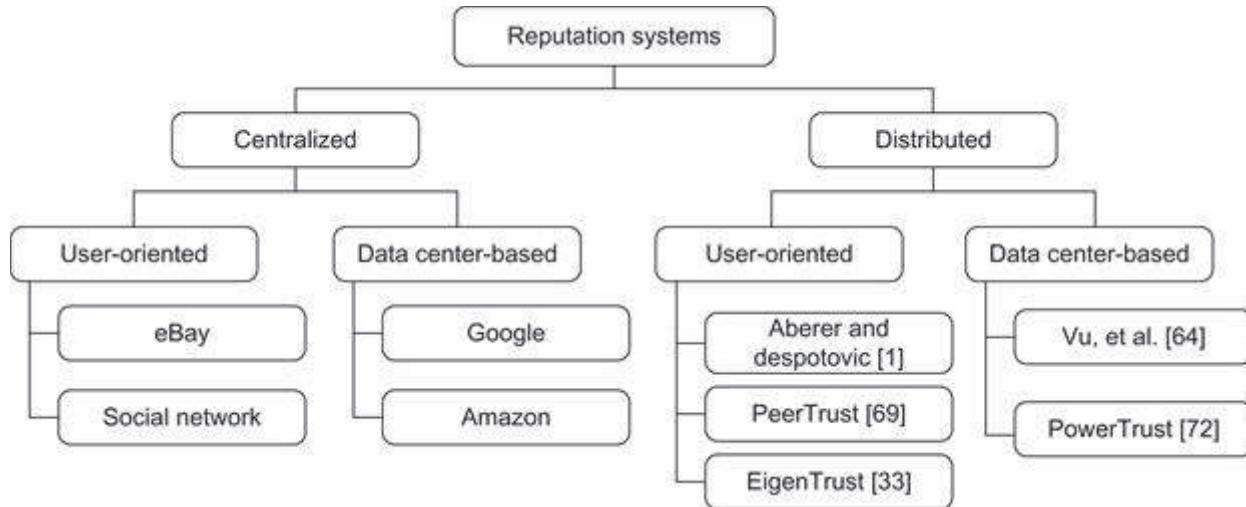
### 4.6.4 Reputation-Guided Protection of Data Centers

Trust is a personal opinion, which is very subjective and often biased. Trust can be transitive but not necessarily symmetric between two parties. Reputation is a public opinion, which is more objective and often relies on a large opinion aggregation process to evaluate. Reputation may change or decay over time. Recent reputation should be given more preference than past reputation. In this section, we review the reputation systems for protecting data centers or cloud user communities.

#### 4.6.4.1 Reputation System Design Options

Figure 4.36 provides an overview of reputation system design options. Public opinion on the character or standing (such as honest behavior or reliability) of an entity could be the reputation of a person, an agent, a product, or a service. It represents a collective

evaluation by a group of people/agents and resource owners. Many reputation systems have been proposed in the past mainly for P2P, multiagent, or e-commerce systems.



**FIGURE 4.36** Design options of reputation systems for social networks and cloud platforms.

To address reputation systems for cloud services, a systematic approach is based on the design criteria and administration of the reputation systems. Figure 4.36 shows a two-tier classification of existing reputation systems that have been proposed in recent years. Most of them were designed for P2P or social networks. These reputation systems can be converted for protecting cloud computing applications. In general, the reputation systems are classified as *centralized* or *distributed* depending on how they are implemented. In a centralized system, a single central authority is responsible for managing the reputation system, while the distributed model involves multiple control centers working collectively. Reputation-based trust management and techniques for securing P2P and social networks could be merged to defend data centers and cloud platforms against attacks from the open network.

A centralized reputation system is easier to implement, but demands more powerful and reliable server resources; a distributed reputation system is much more complex to build. Distributed systems are more scalable and reliable in terms of handling failures. At the second tier, reputation systems are further classified by the scope of reputation evaluation. *User-oriented* reputation systems focus on individual users or agents. Most P2P reputation systems belong to this category. In data centers, reputation is modeled for the resource site as a whole. This reputation applies to products or services offered by the cloud. Commercial reputation systems have been built by eBay, Google, and Amazon in connection with the services they provide. These are centralized reputation systems.

Distributed reputation systems are mostly developed by academic research communities. Aberer and Despotovic have proposed a model to manage trust in P2P systems. The Eigentrust reputation system was developed at Stanford University using a trust matrix approach. The PeerTrust system was developed at Georgia Institute of Technology for supporting e-commerce applications. The PowerTrust system was developed at the University of Southern California based on Power law characteristics of

Internet traffic for P2P applications. Vu, et al. proposed a QoS-based ranking system for P2P transactions.

#### **4.6.4.2 Reputation Systems for Clouds**

Redesigning the aforementioned reputation systems for protecting data centers offers new opportunities for expanded applications beyond P2P networks. Data consistency is checked across multiple databases. Copyright protection secures wide-area content distributions. To separate user data from specific SaaS programs, providers take the most responsibility in maintaining data integrity and consistency. Users can switch among different services using their own data. Only the users have the keys to access the requested data.

The data objects must be uniquely named to ensure global consistency. To ensure data consistency, unauthorized updates of data objects by other cloud users are prohibited. The reputation system can be implemented with a trust overlay network. A hierarchy of P2P reputation systems is suggested to protect cloud resources at the site level and data objects at the file level. This demands both coarse-grained and fine-grained access control of shared resources. These reputation systems keep track of security breaches at all levels.

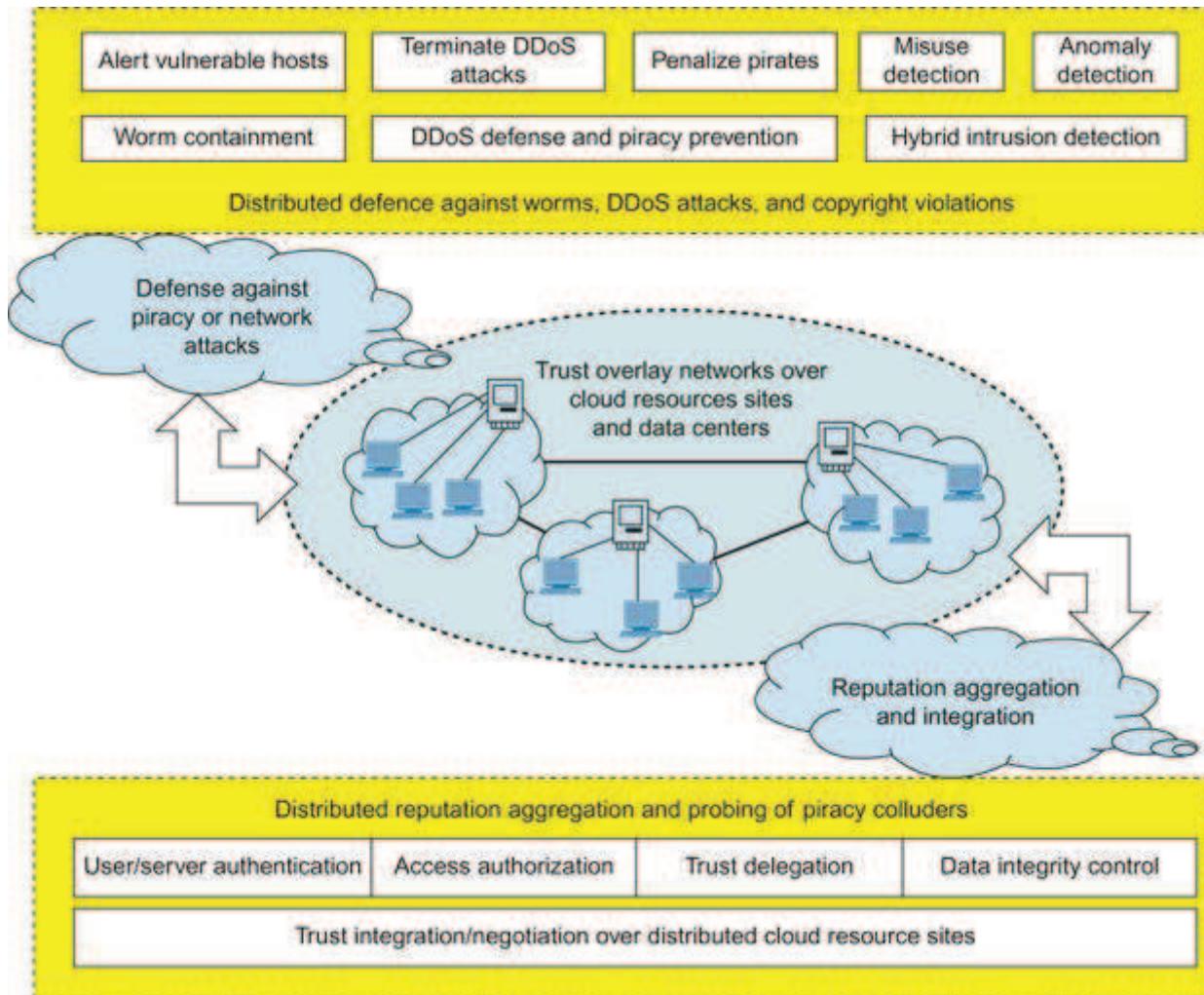
The reputation system must be designed to benefit both cloud users and data centers. Data objects used in cloud computing reside in multiple data centers over a SAN. In the past, most reputation systems were designed for P2P social networking or for online shopping services. These reputation systems can be converted to protect cloud platform resources or user applications in the cloud. A centralized reputation system is easier to implement, but demands more powerful and reliable server resources. Distributed reputation systems are more scalable and reliable in terms of handling failures. The five security mechanisms presented earlier can be greatly assisted by using a reputation system specifically designed for data centers.

However, it is possible to add social tools such as reputation systems to support safe cloning of VMs. Snapshot control is based on the defined RPO. Users demand new security mechanisms to protect the cloud. For example, one can apply secured information logging, migrate over secured virtual LANs, and apply ECC-based encryption for secure migration. Sandboxes provide a safe execution platform for running programs. Further, sandboxes can provide a tightly controlled set of resources for guest operating systems, which allows a security test bed to test the application code from third-party vendors.

#### **4.6.4.3 Trust Overlay Networks**

Reputation represents a collective evaluation by users and resource owners. Many reputation systems have been proposed in the past for P2P, multiagent, or e-commerce systems. To support trusted cloud services, Hwang and Li [36] have suggested building a *trust overlay network* to model trust relationships among data-center modules. This trust overlay could be structured with a *distributed hash table (DHT)* to achieve fast aggregation

of global reputations from a large number of local reputation scores. This trust overlay design was first introduced in [12]. Here, the designer needs to have two layers for fast reputation aggregation, updating, and dissemination to all users. Figure 4.37 shows construction of the two layers of the trust overlay network.



**FIGURE 4.37** DHT-based trust overlay networks built over cloud resources provisioned from multiple data centers for trust management and distributed security enforcement Courtesy of Hwang and Li [36]

At the bottom layer is the trust overlay for distributed trust negotiation and reputation aggregation over multiple resource sites. This layer handles user/server authentication, access authorization, trust delegation, and data integrity control. At the top layer is an overlay for fast virus/worm signature generation and dissemination and for piracy detection. This overlay facilitates worm containment and IDSees against viruses, worms, and DDoS attacks. The content poisoning technique [6] is reputation-based. This protection scheme can stop copyright violations in a cloud environment over multiple data centers.

The reputation system enables trusted interactions between cloud users and data-center owners. Privacy is enforced by matching colored user identifications with the colored data objects. The use of content poisoning was suggested to protect copyright of digital content [46]. The security-aware cloud architecture (see Figure 4.14) is specially

tailored to protect virtualized cloud infrastructure. The trust of provided cloud platforms comes from not only SLAs, but also from effective enforcement of security policies and deployment of countermeasures to defend against network attacks. By varying security control standards, one can cope with the dynamic variation of cloud operating conditions. The design is aimed at a trusted cloud environment to ensure high-quality services, including security.

The cloud security trend is to apply virtualization support for security enforcement in data centers. Both reputation systems and data watermarking mechanisms can protect data-center access at the coarse-grained level and to limit data access at the fine-grained file level. In the long run, a new *Security as a Service* is desired. This “SaaS” is crucial to the universal acceptance of web-scale cloud computing in personal, business, community, and government applications. Internet clouds are certainly in line with IT globalization and efficient computer outsourcing. However, interoperability among different clouds relies on a common operational standard by building a healthy cloud ecosystem.

## 4.7 Bibliographic Notes and Homework Problems

Tutorials and an introduction to cloud computing can be found in [18,32] and Wikipedia [79]. Data centers are introduced in Wikipedia [81]. Four recent books on cloud computing by Buyya, Broberg, and Goscinski [10], Linthicum [47], Rittinghouse and Ransome [62], and Velte, et al. [71] address cloud principles, paradigms, architecture, technologies, and implementations. Warehouse-scale data centers are covered by Barroso and Holzle [8]. Articles related to data-center architecture and implementations can be found in Al-Fares, et al. [2], Greenberg, et al. [29], Guo, et al. [30], Nelson, et al. [53], and Wu, et al. [82]. Green IT is studied in [22,28,60]. Articles arguing about cloud definitions can be found in [25,32,49,86]. Benefits and opportunities of cloud computing are found in [31,32,44,52,61]. Grid extensions for cloud computing can be traced to [66,83].

Pros and cons of cloud computing coexist in the service industry. Others think the cloud offers efficiency and flexibility in making desktop computing centralized. Cloud computing can consolidate the advantages from both centralized and distributed computing [56]. Barham, et al. [6] review Xen [84] and its use in clouds. VMware recovery schemes are given in [74]. Cloud computing infrastructures are reported by both academia [4,21,40,52,62,68] and IT industry or enterprises [3,5,9,26,38,43,65,70]. Cloud technology for HPC or HTC was studied in [33,56,63,87]. Virtualization support for cloud computing is studied in [14,21,41,42,67,76]. Live VM migration and disaster recovery have been studied in [17,54,57,75,78]. Cloud security is covered by Hwang and Li [36] and by Mather, et al. [51].

Business clouds are introduced by Chou in a 2010 book [16] with a lot of case studies of successes and failures. Reputation systems studied in [23,58,73,80,85,89] could be modified for protecting data centers and cloud resource sites. Data coloring and cloud watermarking are studied in [20,36,45,46,84]. Trust models are proposed in [1,23,34,36,50,67] for cloud, pervasive computing, and e-commerce applications. Security

and privacy issues are discussed in [13,15,19,35,36,37, 48,59,85,89]. A global cloud computing test bed, called *Open Circus*, was recently reported in [5]. A cloud simulator, CloudSim, is available from the University of Melbourne [64]. Buyya, et al. [11] and Stuer, et al. [69] have discussed issues concerning market-oriented cloud computing.

## Acknowledgements

This chapter was primarily written by Kai Hwang of University of Southern California. Raj Buyya of Merlbourne University and Kang Chen of Tsinghua University partially contributed to this chapter. Valuable suggestions were made by Geoffery Fox of Indiana University in revising this chapter.

## References

1. Aberer K, Despotovic Z. Managing trust in a peer-to-peer information system. In: *ACM CIKM International Conference on Information and Knowledge Management*. 2001.
2. Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity datacenter network architecture. In: *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. Seattle, WA. 17–22 August 2008.
3. Amazon EC2 and S3. Elastic Compute Cloud (EC2) and Simple Scalable Storage (S3). In: [http://spatten\\_presentation.s3.amazonaws.com/s3-on-rails.pdf](http://spatten_presentation.s3.amazonaws.com/s3-on-rails.pdf).
4. M. Armbrust, A. Fox, R. Griffith, et al., Above the Clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28, University of California at Berkley, 10 February 2009.
5. Arutyun I, et al. Open circus: a global cloud computing testbed. *IEEE Comput Mag*. 2010;35–43.
6. Barham P, et al. Xen and the art of virtualization. In: *Proceedings of the 19th ACM Symposium on Operating System Principles*. New York: ACM Press; 2003.
7. Barroso L, Dean J, Holzle U. Web search for a planet: the architecture of the Google cluster. *IEEE Micro* 2003. doi 10.1109/MM.2003.1196112.
8. Barroso L, Holzle U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines* Morgan Claypool Publishers 2009.
9. Boss G, Mlladi P, et al. Cloud computing: the bluecloud project. In: [www.ibm.com/developerworks/websphere/zones/hipods/](http://www.ibm.com/developerworks/websphere/zones/hipods/); 2007.
10. Buyya R, Broberg J, Goscinski A, eds. *Cloud Computing: Principles and Paradigms*. Wiley Press 2011.
11. Buyya R, Yeo CS, Venugopal S. Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In: *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC)*. Dalian, China. 25–27 September 2008.
12. Buyya R, Ranjan R, Calheiros RN. InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. In: *Proceedings of the*

13. Cai M, Hwang K, Pan J, Papadopoulos C. WormShield: fast worm signature generation with distributed fingerprint aggregation. In: *IEEE Transactions of Dependable and Secure Computing (TDSC)*. April/June 2007;88–104. Vol. 4, No. 2.
14. Chase, et al. Dynamic virtual clusters in a grid site manager. In: *IEEE 12th Symposium on High-Performance Distributed Computing (HPDC)*. 2003.
15. Chen Y, Hwang K, Ku WS. Collaborative detection of DDoS attacks over multiple network domains. In: *IEEE Transaction on Parallel and Distributed Systems*. December 2007;1649–1662. Vol. 18, No. 12.
16. Chou T. *Introduction to Cloud Computing: Business and Technology* Active Book Press 2010.
17. Clark C, Fraser K, Hansen J, et al. Live migration of virtual machines. In: *Proceedings of the Second Symposium on Networked Systems Design and Implementation*. Boston, MA. 2 May 2005;273–286.
18. Cloud Computing Tutorial. In: [www.thecloudtutorial.com](http://www.thecloudtutorial.com); January 2010.
19. Cloud Security Alliance. *Security guidance for critical areas of focus in cloud computing* April 2009.
20. Collberg C, Thomborson C. Watermarking, temper-proofing, and obfuscation tools for software protection. *IEEE Trans Software Eng*. 2002;28:735–746.
21. Costanzo A, Assuncao M, Buyya R. Harnessing cloud technologies for a virtualized distributed computing infrastructure. *IEEE Internet Comput*. 2009.
22. Dyer D. Current trends/Challenges in datacenter thermal management—A facilities perspective. In: *Presentation at ITERM*. San Diego. 1 June 2006.
23. Feng Q, Hwang K, Dai Y. Rainbow product ranking for upgrading e-commerce. *IEEE Internet Comput*. 2009.
24. Foster I. The grid: computing without bounds. *Sci Am*. 2003;288(4):78–85.
25. Foster I, Zhao Y, Raicu J, Lu S. Cloud computing and grid computing 360-degree compared. In: *Grid Computing Environments Workshop*. 12–16 November 2008.
26. Gannon D. The client+cloud: changing the paradigm for scientific research, Keynote address. In: *CloudCom 2010*. Indianapolis. 2 November 2010.
27. Google Inc. Efficient data center summit. In: [www.google.com/corporate/green/datacenters/summit.html](http://www.google.com/corporate/green/datacenters/summit.html); 2009.
28. Grid Green. Quantitative analysis of power distribution configurations for datacenters. In: [www.thegreengrid.org/gg\\_content/](http://www.thegreengrid.org/gg_content/).
29. Greenberg A, Hamilton J, Maltz D, Patel P. The cost of a cloud: research problems in datacenter networks. In: *ACM SIGCOMM Computer Communication Review*. January 2009; Vol. 39, No. 1.
30. Guo C, Lu G, et al. BCube: a high-performance server-centric network architecture for modular datacenters. In: *ACM SIGCOMM Computer Communication Review*. October 2009; Vol. 39, No. 44.
31. Hakan E. Cloud computing: does Nirvana hide behind the Nebula? *IEEE Softw*. 2009.

32. Hayes B. Cloud computing. *Commun ACM*. 2008;51:9–11.
33. Hoffa C, et al. On the use of cloud computing for scientific workflows. In: *IEEE Fourth International Conference on eScience*. December 2008.
34. He R, Hu J, Niu J, Yuan M. A novel cloud-based trust model for pervasive computing. In: *Fourth International Conference on Computer and Information Technology*. 14–16 September 2004;693–700.
35. Hwang K, Kulkarni S, Hu Y. Cloud security with virtualized defense and reputation-based trust management. In: *IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC 09)*. Chengdu, China. 12–14 December 2009.
36. Hwang K, Li D. Trusted cloud computing with secure resources and data coloring. *IEEE Internet Comput*. 2010.
37. Hwang K, Cai M, Chen Y, Qin M. Hybrid intrusion detection with weighted signature generation over anomalous internet episodes. In: *IEEE Transactions on Dependable and Secure Computing*. January–March 2007;41–55. Vol. 4, No. 1.
38. Jinesh V. Cloud Architectures, White paper, Amazon. <http://aws.amazon.com/about-aws/whats-new/2008/07/16/cloud-architectures-white-paper/>.
39. Kamvar D, Schlosser T, Garcia-Molina H. The EigenTrust algorithm for reputation management in P2P networks. In: *Proceedings of the 12th International Conference on the World Wide Web*. 2003.
40. Keahey K, Tsugawa M, Matsunaga A, Fortes J. Sky computing. *IEEE Internet Comput*. 2009.
41. Kivit, et al. KVM: the linux virtual machine monitor. In: *Proceedings of the Linux Symposium*. Ottawa, Canada. 2007;225.
42. KVM Project. *Kernel-based virtual machines*. [www.linux-kvm.org](http://www.linux-kvm.org); 2011; (accessed 02.11).
43. Lakshmanan G. *Cloud Computing: Relevance to Enterprise* Infosys Technologies, Inc. 2009.
44. Leavitt N, et al. Is cloud computing really ready for prime time? *IEEE Comput*. 2009;42(1):15–20.
45. Li D, Meng H, Shi X. Membership clouds and membership cloud generator. *J Comput Res Dev*. 1995;32(6):15–20.
46. Li D, Liu C, Gan W. A new cognitive model: cloud model. *Int J Intell Syst*. 2009.
47. Linthicum D. *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide* Addison Wesley Professional 2009.
48. Lou X, Hwang K. Collusive piracy prevention in P2P content delivery networks. *IEEE Trans Comput*. 2009.
49. Luis M, Vaquero, Rodero-Merino L, et al. A break in the clouds: towards a cloud definition. In: *ACM SIGCOMM Computer Communication Review Archive*. January 2009.
50. Manchala D. E-Commerce trust metrics and models. *IEEE Internet Comput*. 2000.
51. Mather T, et al. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance* O'Reilly Media, Inc. 2009.

52. Mei L, Chan W, Tse T. A tale of clouds: paradigm comparisons and some thoughts on research issues. In: *IEEE Asia-Pacific Services Computing Conference*. December 2008.
53. Nelson D, Ryan M, DeVito S, et al. The role of modularity in datacenter design, Sun BluePrints. In: [www.sun.com/storagetek/docs/EED.pdf](http://www.sun.com/storagetek/docs/EED.pdf).
54. Nelson M, Lim BH, Hutchins G. Fast transparent migration for virtual machines. In: *Proceedings of the USENIX 2005 Annual Technical Conference*. Anaheim, CA. 10–15 April 2005;391–394.
55. D. Nurmi, R. Wolski, et al., Eucalyptus: an Elastic utility computing architecture linking your programs to useful systems, in: UCSB Computer Science Technical Report No. 2008–10, August 2008.
56. Norman W, Paton M, de Aragao T, et al. Optimizing utility in cloud computing through autonomic workload execution. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*. 2009.
57. D.A. Patterson, et al., Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies, UC Berkeley CS Technical Report UCB//CSD-02-1175, 15 March 2002.
58. Pujol M, et al. Extracting reputation in multi-agent systems by means of social network topology. In: *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*. 2002.
59. Roschke S, Cheng F, Meinel C. Intrusion detection in the cloud. In: *IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC 09)*. 13 December 2009.
60. Raghavendra R, Ranganathan P, Talwar V, Wang Z, Zhu X. No ‘power’ struggles: coordinated multi-level power management for the datacenter. In: *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Seattle, WA. March 2008.
61. Reed D. Clouds, clusters and many core: the revolution ahead. In: *Proceedings of the 2008 IEEE International Conference on Cluster Computing*. 29 September – 1 October 2008.
62. Rittinghouse J, Ransome J. *Cloud Computing: Implementation, Management, and Security* CRC Publishers 2010.
63. Rochwerger B, Breitgand D, Levy E, et al. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Syst J*. 2008.
64. R.N. Calheiros, R. Ranjan, C.A.F. De Rose, R. Buyya, CloudSim: a novel framework for modeling and simulation of cloud computing infrastructures and services, Technical Report, GRIDS-TR-2009-1, University of Melbourne, Australia, 13 March 2009.
65. Salesforce.com <http://en.wikipedia.org/wiki/Salesforce.com/>; 2010.
66. Shen H, Hwang K. Locality-preserving clustering and discovery of resources in wide-area computational grids. *IEEE Trans Comput*. 2011; Accepted To Appear.
67. Song S, Hwang K, Zhou R, Kwok Y. Trusted P2P transactions with fuzzy reputation aggregation. In: *IEEE Internet Computing, Special Issue on Security for*

68. Sotomayor B, Montero R, Foster I. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Comput.* 2009.
69. Stuer G, Vanmechelen K, Broeckhovea J. A commodity market algorithm for pricing substitutable grid resources. *Future Gener Comput Syst.* 2007;23(5):688–701.
70. Vecchiola C, Chu X, Buyya R. Aneka: a software platform for .NET-based cloud computing. In: Gentzsch W, ed. *High Speed and Large Scale Scientific Computing*. Amsterdam, Netherlands: IOS Press; 2009;267–295.
71. Velte T, Velite A, Elsenpeter R. *Cloud Computing, A Practical Approach* McGraw-Hill Osborne Media 2010.
72. Venugopal S, Chu X, Buyya R. A negotiation mechanism for advance resource reservation using the alternate offers protocol. In: *Proceedings of the 16th International Workshop on Quality of Service (IWQoS 2008)*. Twente, The Netherlands. June 2008.
73. Vlitalo K, Kortesniemi Y. Privacy in distributed reputation management. In: *Workshop of the 1st Int'l Conference on Security and Privacy for Emerging Areas in Communication Networks*. September 2005.
74. VMware, Inc. *Disaster Recovery Solutions from VMware*, White paper. [www.vmware.com/](http://www.vmware.com/); 2007; (accessed 07).
75. VMware, Inc. *Migrating Virtual Machines with Zero Downtime*. [www.vmware.com/](http://www.vmware.com/); 2010; (accessed 07).
76. VMware, Inc. *vSphere*. [www.vmware.com/products/vsphere/](http://www.vmware.com/products/vsphere/); 2010; (accessed 02.10).
77. Vu L, Hauswirth M, Aberer K. QoS-based service selection and ranking with trust and reputation management. In: *Proceedings of the On The Move Conference (OTM '05)*, LNCS 3760. 2005.
78. Voosluys W, et al. Cost of VM live migration in clouds: a performance evaluation. In: *Proceedings of the First International Conference on Cloud Computing*. Netherlands: IOS Press; 2009;267–295.
79. Wang Y, Vassileva J. Toward trust and reputation based web service selection: a survey. *J Multi-agent Grid Syst (MAGS)* 2007.
80. Wikipedia. *Cloud computing*. [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing); 2010; (accessed 26.01.10).
81. Wikipedia. *Data center*. [http://en.wikipedia.org/wiki/Data\\_center](http://en.wikipedia.org/wiki/Data_center); 2010; (accessed 26.01.10).
82. Wu H, Lu G, Li D, Guo C, Zhang Y. MDCube: a high performance network structure for modular data center interconnection. *ACM CoNEXT '09, Rome* 1–4 December 2009.
83. Wu Y, Hwang K, Yuan Y, Zheng W. Adaptive workload prediction of grid performance in confidence windows. *IEEE Trans Parallel Distrib Syst.* 2010.
84. XEN Organization. [www.sen.org](http://www.sen.org); 2011; (accessed 20.02.11).
85. Xiong L, Liu L. PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans Knowl Data Eng.* 2004;843–857.
86. Yang J, Wang J, Wang C, Li. D. A novel scheme for watermarking natural

- language text. In: *Proceedings of the Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. 2007;481–484.
87. Youseff L, Butrico M, Maria D, Silva D. Toward a unified ontology of cloud computing. In: *Grid Computing Environments Workshop (GCE '08)*. November 2008;1–10.
  88. Zhang F, Cao J, Hwang K, Wu C. Ordinal optimized scheduling of scientific workflows in elastic compute Clouds. *IEEE Trans Comput*. 2011; submitted (under review).
  89. Zhou R, Hwang K. PowerTrust: a robust and scalable reputation system for trusted peer-to-peer computing. *IEEE Trans Parallel Distrib Syst*. 2007.

## Homework Problems

### Problem 4.1

Compile a table to compare public clouds and private clouds in each of the following four aspects. Also identify their differences, advantages, and shortcomings in terms of design technologies and application flexibility. Give several example platforms that you know of under each cloud class.

- a. Technology leveraging and IT resource ownership
- b. Provisioning methods of resources including data and VMs, and their management
- c. Workload distribution methods and loading policies
- d. Security precautions and data privacy enforcement

### Problem 4.2

Describe the following techniques or terminologies used in cloud computing and cloud services. Use a concrete example cloud or case study to explain the addressed technology.

- a. Virtualized data center
- b. Green information technology
- c. Multitenant technique

### Problem 4.3

You have read a number of success or failure stories on cloud computing applications by startup entrepreneurs or established IT companies. In Chou's introductory book on business computing [18], six categories of cloud services are identified, as listed here and shown below and also in Figure 4.23.

*Software Applications (SaaS)*: Concur, RightNOW, Teleo, Kenexa, Webex, Blackbaud, [Salesforce.com](#), Netsuite, Omniture, Kenexa, Vocus

*Platform Services (PaaS)*: [Force.com](#), App Engine, Postini, Facebook, MS Azure, NetSuite, IBM RC2, IBM BlueCloud, SGI Cyclone, eBay, Pitney Bowes

*Infrastructure Services (SaaS)*: Amazon AWS, OpSource Cloud, IBM Ensembles, Eli Lily, Rackspace cloud, Windows Azure, HP, Bank North, New York Times

*Collocation Services(LaaS):* Savvis, Internap, NTT Communications, Digital Realty Trust, 365 Main

*Network Cloud Services (NaaS):* Owest, AT&T, AboveNet

*Hardware/Virtualization Services (HaaS):* VMware, Intel, IBM, XenEnterprise

Pick one company from each of the six cloud service categories. Dig into the company services in detail, contact the company directly, or visit their web sites. The purpose is to report on their innovations in cloud technology, good business ideas implemented, software application development, business models they have developed, and success/failure lessons learned.

## Problem 4.4

Check the AWS cloud web site. Plan a real computing application using EC2, or S3, or SQS, separately. You must specify the resources requested and figure out the costs charged by Amazon. Carry out the EC2, S3, or SQS experiments on the AWS platform and report and analyze the performance results measured.

## Problem 4.5

Consider two cloud service systems: Google File System and Amazon S3. Explain how they achieve their design goals to secure data integrity and to maintain data consistency while facing the problems of hardware failure, especially concurrent hardware failures.

## Problem 4.6

Read the articles by Buyya and associates [10–14] on market-oriented cloud architecture and intercloud resource exchanges:

- a. Discuss the reasons for scaling applications across multiple clouds. Suggest ways to leverage the ideas in cloud mashup applications.
- b. Identify key architectural elements that need to be added to realize market-oriented clouds within commercial cloud systems supported by companies such as IBM and Microsoft.

## Problem 4.7

Suggest two hardware mechanisms and software schemes to secure the application cloud (SaaS), the infrastructure cloud (IaaS), and the platform cloud (PaaS). Discuss their specific requirements and difficulties and limitations that may be encountered.

## Problem 4.8

Identify infrastructure components of IaaS services offered by Amazon EC2 and Google, and use CloudSim from the University of Melbourne to model and simulate these infrastructures. Develop an algorithm for scheduling applications across an InterCloud

or federation of these simulated infrastructures. In this evaluation, carry out experiments by modeling applications that are short-lived such as web applications and long-lived such as high-performance computing.

## Problem 4.9

Explain the following terminologies associated with data-center design and managements:

- a. Differences in warehouse-scale data centers and modular data centers
- b. Study three data center architecture papers in [2,24,25,45,68] and report on their contributions to advance data-center performance and dependability
- c. Discuss the scalability of those data-center architectures you studied in Part (b)

## Problem 4.10

Explain the differences in the following two machine recovery schemes. Comment on their implementation requirements, advantages and shortcomings, and application potential.

- a. Recovery of a physical machine failure by another physical machine
- b. Recovery of a VM failure by another VM

## Problem 4.11

Elaborate on four major advantages of using virtualized resources in cloud computing applications. Your discussion should address resource management issues from the provider's perspective and the application flexibility, cost-effectiveness, and dependability concerns by cloud users.

## Problem 4.12

Answer the following questions on the InterGrid Gateway (IGG) experiments. Read the original paper [5] for details beyond Sections 6.3.3 and 6.3.4 on cloud creation and load peering experiments in a distributed virtualized cloud infrastructure built on top of the French Grid'5000 system.

- a. Study the details of the IGG software components and their usage to link resource sites
- b. Repeat the IGG experiments in a local small-scale network or grid environment by getting the IGG software from the University of Melbourne
- c. Use the CloudSim simulator from the University of Melbourne or write your own simulator to repeat the experiments in Part (b).

## Problem 4.13

Map the following 14 names or abbreviated terms on the left column with the best-match definitions or descriptions on the right column. Just enter the description label (a, b, c,

....) in the blank spaces in front of the terms being matched to. This is a 1-to-1 correspondence.

Term	Description to Be Mapped with the Terms on the Left Column
_____ GAE	a) The agreement signed between users and providers in cloud computing
_____ CRM	b) A public cloud that must run from Windows 7 based host
_____ AWS	c) A public cloud used mainly for PaaS applications
_____ SLA	d) A public compute cloud used in scalable business computing application
_____ Azure	e) A cloud platform built by SalesForce.com
_____ EC2	f) A commercial cloud OS for transforming a data center to a cloud platform
_____ S3	g) One of the most frequently used SaaS applications used in the business world
_____ Force.com	h) A cloud platform built mainly for IaaS applications
_____ vSphere/4	i) A storage cloud service for distributed storage applications
_____ EBS	j) An open-source hypervisor developed at Cambridge University
_____ SQL	k) The distributed file system used in Google search engines and in App Engine
_____ Chubby	l) An Amazon block lock interface for saving and restoring the VM instances
_____ XEN	m) An Azure service module for users to visit and use of relational database
_____ GFS	n) A distributed block lock service module in Google App Engine

## Problem 4.14

This is an extended research problem on an efficient data-center network by Al-Fare, et al. [18] that you studied in Section 4.2.2. The scalable data-center architecture was proposed to extend the fat-tree concept. You are asked to perform the following tasks:

- Study these two papers and justify the claimed network features on scalability, efficient routing, and compatibility with the Ethernet, IP, and TCP protocols
- Suggest a means to improve the network in fault tolerance, cross-section bandwidth, and implementation efficiency based on today's technology.

## Problem 4.15

You have studied the basic concepts of live migration of VMs and disaster recovery in Section 3.5.2 and Section 4.3.3. Read the related articles in [16,19,52,55,72] and answer the following questions based on your research findings:

- What virtualization support is needed to achieve fast cloning of VMs? Explain how VM cloning can enable fast recovery.
- What are the RPO and RTO in the design of a disaster recovery scheme? Explain the role of snapshots in disaster recovery schemes.

## Problem 4.16

Investigate the differences among encryption, watermarking, and coloring for protecting

data sets and software in cloud environments. Discuss their relative strengths and limitations.

## Problem 4.17

This is an extended research problem on efficient data-center networks. In the papers by Guo, et al. [25] and Wu, et al. [68], the MDCube networks module was proposed for building mega data centers as shown in [Figure 4.13](#) in [Section 4.2.4](#).

- a. Discuss the advantages of using this network to improve the intercontainer bandwidth, reduce the cost of the interconnection structure, and reduce the cabling complexity.
- b. Justify the claimed low diameter, high capacity, and fault tolerance of the MDCube network design for interconnecting data-center containers.

## Problem 4.18

Read the articles [15,17,21,35,37,46,49,57] on cloud security and data protection. Can you suggest any way to upgrade the cloud infrastructure for protection of data, privacy, and security in a cloud platform or data-center environment? Evaluate the trust models reported in [1,25,34,36,48,64] for securing cloud service applications.

## Problem 4.19

Draw a layered diagram to relate the construction of IaaS, PaaS, and SaaS clouds from bare machine hardware to the users applications. Briefly list the representative cloud service offerings at each cloud layer from the major cloud providers that you know of.

## Problem 4.20

Discuss the enabling technologies for building the cloud platforms from virtualized and automated data centers to provide IaaS, PaaS, or SaaS services. Identify hardware, software, and networking mechanisms or business models that enable multitenant services.

# CHAPTER 5

# Service-Oriented Architectures for Distributed Computing

## CHAPTER OUTLINE

### Summary

#### 5.1 Services and Service-Oriented Architecture

- 5.1.1 REST and Systems of Systems
- 5.1.2 Services and Web Services
- 5.1.3 Enterprise Multitier Architecture
- 5.1.4 Grid Services and OGSA
- 5.1.5 Other Service-Oriented Architectures and Systems

#### 5.2 Message-Oriented Middleware

- 5.2.1 Enterprise Bus
- 5.2.2 Publish-Subscribe Model and Notification
- 5.2.3 Queuing and Messaging Systems
- 5.2.4 Cloud or Grid Middleware Applications

#### 5.3 Portals and Science Gateways

- 5.3.1 Science Gateway Exemplars
- 5.3.2 HUBzero Platform for Scientific Collaboration
- 5.3.3 Open Gateway Computing Environments (OGCE)

#### 5.4 Discovery, Registries, Metadata, and Databases

- 5.4.1 UDDI and Service Registries
- 5.4.2 Databases and Publish-Subscribe
- 5.4.3 Metadata Catalogs
- 5.4.4 Semantic Web and Grid
- 5.4.5 Job Execution Environments and Monitoring

#### 5.5 Workflow in Service-Oriented Architectures

- 5.5.1 Basic Workflow Concepts
- 5.5.2 Workflow Standards
- 5.5.3 Workflow Architecture and Specification
- 5.5.4 Workflow Execution Engine
- 5.5.5 Scripting Workflow System Swift

## 5.6 Bibliographic Notes and Homework Problems

### Acknowledgements

### References

### Homework Problems

In this chapter, we cover two major service-oriented architecture styles, namely REST (*REpresentational State Transfer*) and WS (*Web Services*) and their extensions. We discuss message-oriented middleware and enterprise bus infrastructure with a publish-subscribe infrastructure. Two application interfaces (OGCE and HUBzero) are described using web service (portlet) and Web 2.0 (gadget) technologies. We handle data and metadata in distributed systems using service registries and semantic web/grid. Finally, we describe a general workflow approach illustrating it with the BPEL web service standard, Pegasus, Taverna, Kepler, Trident, and Swift.

service-oriented architecture (SOA), Web services, message-oriented middleware, science gateways, databases, workflows, Open Gateway Computing Environments (OGCE)

## Summary

In this chapter, we cover two major service-oriented architecture styles, namely REST (*REpresentational State Transfer*) and WS (*Web Services*) and their extensions. We discuss message-oriented middleware and enterprise bus infrastructure with a publish-subscribe infrastructure. Two application interfaces (OGCE and HUBzero) are described using web service (portlet) and Web 2.0 (gadget) technologies. We handle data and metadata in distributed systems using service registries and semantic web/grid. Finally, we describe a general workflow approach illustrating it with the BPEL web service standard, Pegasus, Taverna, Kepler, Trident, and Swift.

## 5.1 Services and Service-Oriented Architecture

Technology has advanced at breakneck speeds over the past decade, and many changes are still occurring. However, in this chaos, the value of building systems in terms of services has grown in acceptance and it has become a core idea of most distributed systems. Loose coupling and support of heterogeneous implementations makes services more attractive than distributed objects. Web services move beyond helping various types of applications to exchanging information. The technology also plays an increasingly important role in accessing, programming on, and integrating a set of new and existing applications.

We have introduced *service-oriented architecture* (SOA) in [Section 1.4.1](#). In general, SOA is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces. These applications are

often distributed over the networks. SOA also aims to make service interoperability extensible and effective. It prompts architecture styles such as loose coupling, published interfaces, and a standard communication model in order to support this goal. The *World Wide Web Consortium* (W3C) defines SOA as a form of distributed systems architecture characterized by the following properties [1]:

**Logical view:** The SOA is an abstracted, logical view of actual programs, databases, business processes, and so on, defined in terms of what it does, typically carrying out a business-level operation. The service is formally defined in terms of the messages exchanged between provider agents and requester agents.

**Message orientation:** The internal structure of providers and requesters include the implementation language, process structure, and even database structure. These features are deliberately abstracted away in the SOA: Using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application to adhere to the formal service definition.

**Description orientation:** A service is described by machine-executable metadata. The description supports the public nature of the SOA: Only those details that are exposed to the public and are important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.

- **Granularity** Services tend to use a small number of operations with relatively large and complex messages.
- **Network orientation** Services tend to be oriented toward use over a network, though this is not an absolute requirement.
- **Platform-neutral** Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

Unlike the component-based model, which is based on design and development of tightly coupled components for processes within an enterprise, using different protocols and technologies such as CORBA and DCOM, SOA focuses on loosely coupled software applications running across different administrative domains, based on common protocols and technologies, such as HTTP and XML. SOA is related to early efforts on the architecture style of large-scale distributed systems, particularly *Representational State Transfer* (REST). Nowadays, REST still provides an alternative to the complex standard-driven web services technology and is used in many Web 2.0 services. In the following subsections, we introduce REST and standard-based SOA in distributed systems.

## 5.1.1 REST and Systems of Systems

REST is a software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web. It has recently gained popularity among enterprises such as Google, Amazon, Yahoo!, and especially social networks such

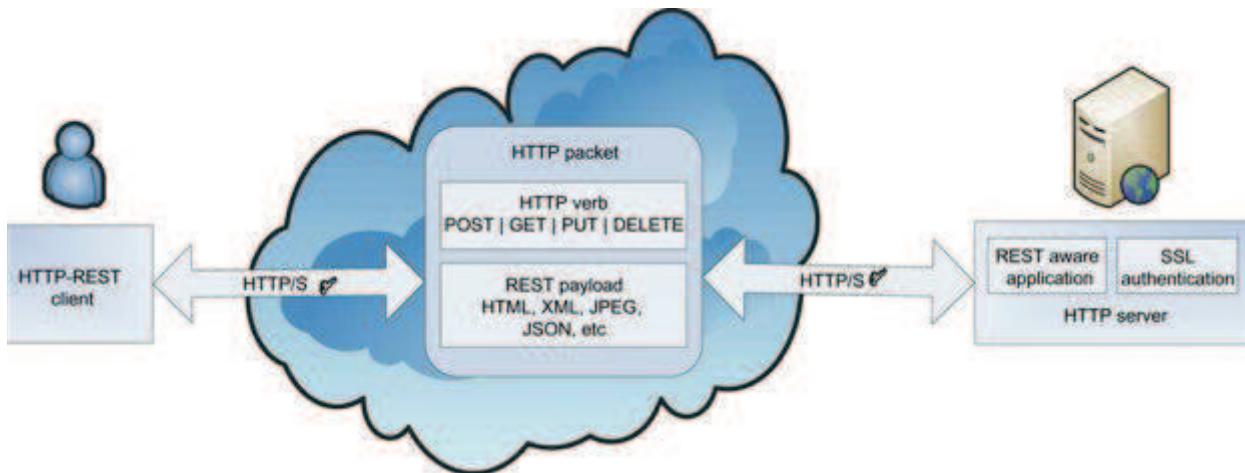
as Facebook and Twitter because of its simplicity, and its ease of being published and consumed by clients. REST, shown in [Figure 5.1](#), was introduced and explained by Roy Thomas Fielding, one of the principal authors of the HTTP specification, in his doctoral dissertation [\[2\]](#) in 2000 and was developed in parallel with the HTTP/1.1 protocol. The REST architectural style is based on four principles:

**Resource Identification through URIs:** The RESTful web service exposes a set of resources which identify targets of interaction with its clients. The key abstraction of information in REST is a resource. Any information that can be named can be a resource, such as a document or image or a temporal service. A resource is a conceptual mapping to a set of entities. Each particular resource is identified by a unique name, or more precisely, a *Uniform Resource Identifier (URI)* which is of type URL, providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery. The URIs can be bookmarked or exchanged via hyperlinks, providing more readability and the potential for advertisement.

**Uniform, Constrained Interface:** Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol. Resources are manipulated using a fixed set of four CRUD (create, read, update, delete) verbs or operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can then be destroyed by using DELETE. GET retrieves the current state of a resource. POST transfers a new state onto a resource.

**Self-Descriptive Message:** A REST message includes enough information to describe how to process the message. This enables intermediaries to do more with the message without parsing the message contents. In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.). REST provides multiple/alternate representations of each resource. Metadata about the resource is available and can be used for various purposes, such as cache control, transmission error detection, authentication or authorization, and access control.

**Stateless Interactions:** The REST interactions are “stateless” in the sense that the meaning of a message does not depend on the state of the conversation. Stateless communications improve visibility, since a monitoring system does not have to look beyond a single request data field in order to determine the full nature of the request reliability as it facilitates the task of recovering from partial failures, and increases scalability as discarding state between requests allows the server component to quickly free resources. However, stateless interactions may decrease network performance by increasing the repetitive data (per-interaction overhead). Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.



**FIGURE 5.1** A simple REST interaction between user and server in HTTP specification. Courtesy of Thomas Fielding [2]

Such lightweight infrastructure, where services can be built with minimal development tools, is inexpensive and easy to adopt. The effort required to build a client to interact with a RESTful service is rather small as developers can begin testing such services from an ordinary web browser, without having to develop custom client-side software. From an operational point of view, a stateless RESTful web service is scalable to serve a very large number of clients, as a result of REST support for caching, clustering, and load balancing.

RESTful web services can be considered an alternative to SOAP stack or “big web services,” described in the next section, because of their simplicity, lightweight nature, and integration with HTTP. With the help of URIs and hyperlinks, REST has shown that it is possible to discover web resources without an approach based on registration to a centralized repository. Recently, the web Application Description Language (WADL) [3] has been proposed as an XML vocabulary to describe RESTful web services, enabling them to be discovered and accessed immediately by potential clients. However, there are not a variety of toolkits for developing RESTful applications. Also, restrictions on GET length, which does not allow encoding of more than 4 KB of data in the resource URI, can create problems because the server would reject such malformed URIs, or may even be subject to crashes. REST is not a standard. It is a design and architectural style for large-scale distributed systems.

Table 5.1 lists the REST architectural elements. Several Java frameworks have emerged to help with building RESTful web services. Restlet [4], a lightweight framework, implements REST architectural elements such as resources, representation, connector, and media type for any kind of RESTful system, including web services. In the Restlet framework, both the client and the server are components. Components communicate with each other via connectors.

## Table 5.1

### REST Architectural Elements (Adapted from [2])

REST Elements	Elements	Example
Data elements	Resource	The intended conceptual target of a hypertext reference
	Resource identifier	URL
	Representation	HTML document, JPEG image, XML, etc.
	Representation metadata	Media type, last-modified time
	Resource metadata	Source link, alternates, vary
	Control data	If-modified-since, cache-control
	Client	libwww, libwww-perl
Connectors	Server	libwww, Apache API, NSAPI
	Cache	Browser cache, Akamai cache network
	Resolver	Bind (DNS lookup library)
	Tunnel	SSL after HTTP CONNECT
	Origin server	Apache httpd, Microsoft IIS
Components	Gateway	Squid, CGI, Reverse Proxy
	Proxy	CERN Proxy, Netscape Proxy, Gauntlet
	User agent	Netscape Navigator, Lynx, MOMspider

JSR-311 (JAX-RS) [5], a specification provided by Sun Microsystems, defines a set of Java APIs for the development of RESTful web services. The specification provides a set of annotations with associated classes and interfaces that can be used to expose Java objects as web resources. JSR-311 provides clear mappings between the URI and corresponding resources, and mappings between HTTP methods with the methods in Java objects, by using annotations. The API supports a wide range of HTTP entity content types including HTML, XML, JSON, GIF, JPG, and so on. Jersey [6] is a reference implementation of the JSR-311 specification for building RESTful web services. It also provides an API for developers to extend Jersey based on their needs.

## Example 5.1

### RESTful Web Service in Amazon S3 Interface

A good example of RESTful web service application in high-performance computing systems is the Amazon Simple Storage Service (S3) interface. Amazon S3 is data storage for Internet applications. It provides simple web services to store and retrieve data from anywhere at any time via the web. S3 keeps fundamental entities, “objects,” which are named pieces of data accompanied by some metadata to be stored in containers called “buckets,” each identified by a unique key. Buckets serve several purposes: They organize the Amazon S3 namespace at the highest level, identify the account responsible for storage and data transfer charges, play a role in access control, and serve as the unit of aggregation for usage reporting. Amazon S3 provides three types of resources: a list of user buckets, a particular bucket, and a particular S3 object, accessible through <https://s3.amazonaws.com/{name-of-bucket}/{name-of-object}>.

These resources are retrieved, created, or manipulated by basic HTTP standard operations: GET, HEAD, PUT, and DELETE. GET can be used to list buckets created by the user, objects kept inside a bucket, or an object's value and its related metadata. PUT can be used for creating a bucket or setting an object's value or metadata, DELETE for removing a particular bucket or object, and HEAD for getting a specific object's metadata. The Amazon S3 API supports the ability to find buckets, objects, and their related metadata; create new buckets; upload objects; and delete existing buckets and objects for the aforementioned operations. Table 5.2 shows some sample REST request-response message syntax for creating an S3 bucket.

**Table 5.2**

### Sample REST Request-Response for Creating an S3 Bucket

REST Request	REST Response
PUT/[bucket-name] HTTP/1.0 Date: Wed, 15 Mar 2011 14:45:15 GMT Authorization:AWS [aws-access-key-id]: [header-signature] Host: <a href="https://s3.amazonaws.com">s3.amazonaws.com</a>	HTTP/1.1 200 OK x-amz-id-2: VjzdTviQorQtSjcgLshzCZSzN+7CnewHA+6sNxR3VRcUPyO5fmSmo8bWnIS52qa x-amz-request-id: 91A8CC60F9FC49E7 Date: Wed, 15 Mar 2010 14:45:20 GMT Location: /[bucket-name] Content-Length: 0 Connection: keep-alive

Amazon S3 REST operations are HTTP requests to create, fetch, and delete buckets and objects. A typical REST operation consists of sending a single HTTP request to Amazon S3, followed by waiting for an HTTP response. Like any HTTP request, a request to Amazon S3 contains a request method, a URI, request headers which contain basic information about the request, and sometimes a query string and request body. The response contains a status code, response headers, and sometimes a response body.

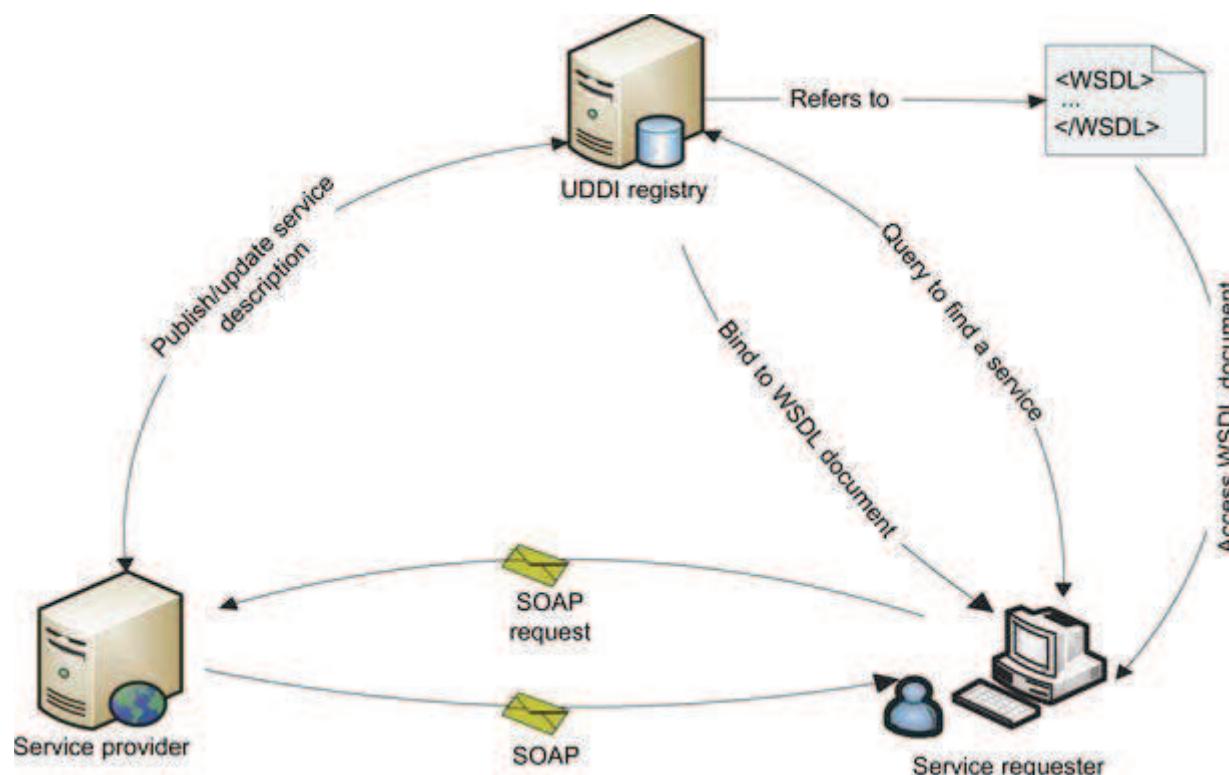
The request consists of a PUT command followed by the bucket name created on S3. The Amazon S3 REST API uses the standard HTTP header to pass authentication information. The authorization header consists of an AWS Access Key ID and AWS SecretAccess Key, issued by the developers when they register to S3 Web Services, followed by a signature. To authenticate, the *AWSAccessKeyId* element identifies the secret key to compute the signature upon request from the developer. If the request signature matches the signature included, the requester is authorized and subsequently, the request is processed.

The composition of RESTful web services has been the main focus of composite Web 2.0 applications, such as mashups, to be discussed in Sections 5.4 and 9.1.3. A mashup

application combines capabilities from existing web-based applications. A good example of a mashup is taking images from an online repository such as Flickr and overlaying them on Google Maps. Mashups differ from all-in-one software products in that instead of developing a new feature into an existing tool, they combine the existing tool with another tool that already has the desired feature. All tools work independently, but create a uniquely customized experience when used together in harmony.

## 5.1.2 Services and Web Services

In an SOA paradigm, software capabilities are delivered and consumed via loosely coupled, reusable, coarse-grained, discoverable, and self-contained services interacting via a message-based communication model. The web has become a medium for connecting remote clients with applications for years, and more recently, integrating applications across the Internet has gained in popularity. The term “web service” is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web. Once a web service is deployed, other applications and other web services can discover and invoke the deployed service (Figure 5.2).



**FIGURE 5.2** A simple web service interaction among provider, user, and the UDDI registry.

In fact, a web service is one of the most common instances of an SOA implementation. The W3C working group [1] defines a web service as a software system designed to support interoperable machine-to-machine interaction over a network. According to this definition, a web service has an interface described in a machine-executable format (specifically Web Services Description Language or WSDL). Other systems interact with

the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards. The technologies that make up the core of today's web services are as follows:

**Simple Object Access Protocol (SOAP)** SOAP provides a standard packaging structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP. By having such a standard message format, heterogeneous middleware systems can achieve interoperability. A SOAP message consists of a root element called *envelope*, which contains a *header*: a container that can be extended by intermediaries with additional application-level elements such as routing information, authentication, transaction management, message parsing instructions, and Quality of Service (QoS) configurations, as well as a *body* element that carries the payload of the message. The content of the payload will be marshaled by the sender's SOAP engine and unmarshaled at the receiver side, based on the XML schema that describes the structure of the SOAP message.

**Web Services Description Language (WSDL)** WSDL describes the interface, a set of operations supported by a web service in a standard format. It standardizes the representation of input and output parameters of its operations as well as the service's protocol binding, the way in which the messages will be transferred on the wire. Using WSDL enables disparate clients to automatically understand how to interact with a web service.

**Universal Description, Discovery, and Integration (UDDI)** UDDI provides a global registry for advertising and discovery of web services, by searching for names, identifiers, categories, or the specification implemented by the web service. UDDI is explained in detail in [Section 5.4](#).

SOAP is an extension, and an evolved version of XML-RPC, a simple and effective remote procedure call protocol which uses XML for encoding its calls and HTTP as a transport mechanism, introduced in 1999 [\[7\]](#). According to its conventions, a procedure executed on the server and the value it returns was a formatted in XML. However, XML-RPC was not fully aligned with the latest XML standardization. Moreover, it did not allow developers to extend the request or response format of an XML-RPC call. As the XML schema became a W3C recommendation in 2001, SOAP mainly describes the protocols between interacting parties and leaves the data format of exchanging messages to XML schema to handle.

The major difference between web service technology and other technologies such as J2EE, CORBA, and CGI scripting is its standardization, since it is based on standardized XML, providing a language-neutral representation of data. Most web services transmit messages over HTTP, making them available as Internet-scale applications. In addition, unlike CORBA and J2EE, using HTTP as the tunneling protocol by web services enables remote communication through firewalls and proxies.

SOAP-based web services are also referred to as "big web services" [\[7\]](#). As we saw earlier in this chapter, RESTful [\[8\]](#) services can also be considered a web service, in an HTTP context. SOAP-based web services interaction can be either synchronous or

asynchronous, making them suitable for both request-response and one-way exchange patterns, thus increasing web service availability in case of failure.

### 5.1.2.1 WS-I Protocol Stack

Unlike RESTful web services that do not cover QoS and contractual properties, several optional specifications have been proposed for SOAP-based web services to define nonfunctional requirements and to guarantee a certain level of quality in message communication as well as reliable, transactional policies, such as WS-Security [9], WS-Agreement [10], WS-ReliableMessaging [11], WS-Transaction [12], and WS-Coordination [13] as shown in Figure 5.3.



**FIGURE 5.3** WS-I protocol stack and its related specifications.

As mentioned, SOAP messages are encoded using XML, which requires that all self-described data be sent as ASCII strings. The description takes the form of start and end tags which often constitute half or more of the message's bytes. Transmitting data using XML leads to a considerable transmission overhead, increasing the amount of transferred data by a factor 4 to 10 [14]. Moreover, XML processing is compute and memory intensive and grows with both the total size of the data and the number of data fields, making web services inappropriate for use by limited-profile devices, such as handheld PDAs and mobile phones.

Web services provide on-the-fly software composition, described further in [Section 5.5](#), through the use of loosely coupled, reusable software components. By using Business Process Execution Language for Web Services (BPEL4WS), a standard executable language for specifying interactions between web services recommended by OASIS, web services can be composed together to make more complex web services and workflows. BPEL4WS is an XML-based language, built on top of web service specifications, which is

used to define and manage long-lived service orchestrations or processes.

In BPEL, a business process is a large-grained stateful service, which executes steps to complete a business goal. That goal can be the completion of a business transaction, or fulfillment of the job of a service. The steps in the BPEL process execute activities (represented by BPEL language elements) to accomplish work. Those activities are centered on invoking partner services to perform tasks (their job) and return results back to the process. BPEL enables organizations to automate their business processes by orchestrating services. Workflow in a grid context is defined [15] as “The automation of the processes, which involves the orchestration of a set of Grid services, agents and actors that must be combined together to solve a problem or to define a new service.”

The JBPM [16] Project, built for the JBoss [17] open source middleware platform, is an example of a workflow management and business process execution system. Another workflow system, Taverna [18], has been extensively used in life science applications. There are a variety of tools for developing and deploying web services in different languages, among them SOAP engines such as Apache Axis for Java, gSOAP [19] for C++, the Zolera Soap Infrastructure (ZSI) [20] for Python, and Axis2/Java and Axis2/C. These toolkits, consisting of a SOAP engine and WSDL tools for generating client stubs, considerably hide the complexity of web service application development and integration. As there is no standard SOAP mapping for any of the aforementioned languages, two different implementations of SOAP may produce different encodings for the same objects.

Since SOAP can combine the strengths of XML and HTTP, as a standard transmission protocol for data, it is an attractive technology for heterogeneous distributed computing environments, such as grids and clouds, to ensure interoperability. As we discussed in Section 7.4, Open Grid Services Architecture (OGSA) grid services are extensions of web services and in new grid middleware, such as Globus Toolkit 4 and its latest released version GT5, pure standard web services. Amazon S3 as a cloud-based persistent storage service is accessible through both a SOAP and a REST interface. However, REST is the preferred mechanism for communicating with S3 due to the difficulties of processing large binary objects in the SOAP API, and in particular, the limitation that SOAP puts on the object size to be managed and processed. Table 5.3 depicts a sample SOAP request-response to get a user object from S3.

**Table 5.3**

### Sample SOAP Request-Response for Creating an S3 Bucket

SOAP Request	SOAP Response
<pre>&lt;soap:Envelope   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"   soap:encodingStyle=   "http://www.w3.org/2001/12/soap-encoding"&gt;   &lt;soap:Body&gt;     &lt;CreateBucket xmlns="http://doc.s3.amazonaws.com/2010-03-15"&gt;       &lt;Bucket&gt;SampleBucket&lt;/Bucket&gt;</pre>	<pre>&lt;soap:Envelope   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"   soap:encodingStyle=   "http://www.w3.org/2001/12/soap-encoding"&gt;   &lt;soap:Body&gt;     &lt;CreateBucket xmlns="http://doc.s3.amazonaws.com/2010-03-15"&gt;       &lt;Bucket&gt;SampleBucket&lt;/Bucket&gt;</pre>

```

<AWSAccessKeyId>
1B9FVRAYCP1VJEXAMPLE=
</AWSAccessKeyId>
<Timestamp>2010-03-15T14:40:00.165Z
</Timestamp>
<Signature>luyz3d3P0aTou39dzbqaEXAMPLE
=</Signature>
</CreateBucket>
</soap:Body>
</soap:Envelope>

```

```

<AWSAccessKeyId>1B9FVRAYCP1VJEXAMPLE=
</AWSAccessKeyId>
<Timestamp>2010-03-15T14:40:00.165Z
</Timestamp>
<Signature>luyz3d3P0aTou39dzbqaEXAMPLE
=</Signature>
</CreateBucket>
</soap:Body>
</soap:Envelope>

```

A SOAP message consists of an envelope used by the applications to enclose information that need to be sent. An envelope contains a header and a body block. The *EncodingStyle* element refers to the URI address of an XML schema for encoding elements of the message. Each element of a SOAP message may have a different encoding, but unless specified, the encoding of the whole message is as defined in the XML schema of the root element. The header is an optional part of a SOAP message that may contain auxiliary information as mentioned earlier, which does not exist in this example.

The body of a SOAP request-response message contains the main information of the conversation, formatted in one or more XML blocks. In this example, the client is calling *CreateBucket* of the Amazon S3 web service interface. In case of an error in service invocation, a SOAP message including a *Fault* element in the body will be forwarded to the service client as a response, as an indicator of a protocol-level error.

### 5.1.2.2 WS-\* Core SOAP Header Standards

Table 5.4 summarizes some of the many (around 100) core SOAP header specifications. There are many categories and several overlapping standards in each category. Many are expanded in this chapter with XML, WSDL, SOAP, BPEL, WS-Security, UDDI, WSRF, and WSRP. The number and complexity of the WS-\* standards have contributed to the growing trend of using REST and not web services. It was a brilliant idea to achieve interoperability through self-describing messages, but experience showed that it was too hard to build the required tooling with the required performance and short implementation time.

**Table 5.4**

#### The 10 Areas Covered by the Core WS-\* Specifications

WS-* Specification Area	Examples
1. Core Service Model	XML, WSDL, SOAP
2. Service Internet	WS-Addressing, WS-MessageDelivery, Reliable WSRM, Efficient MOTM
3. Notification	WS-Notification, WS-Eventing (Publish-Subscribe)
4. Workflow and Transactions	BPEL, WS-Choreography, WS-Coordination
5. Security	WS-Security, WS-Trust, WS-Federation, SAML, WS-SecureConversation
6. Service Discovery	UDDI, WS-Discovery
7. System Metadata and State	WSRF, WS-MetadataExchange, WS-Context
8. Management	WSDM, WS-Management, WS-Transfer

9. Policy and Agreements	WS-Policy, WS-Agreement
10. Portals and User Interfaces	WSRP (Remote Portlets)

### Example 5.2

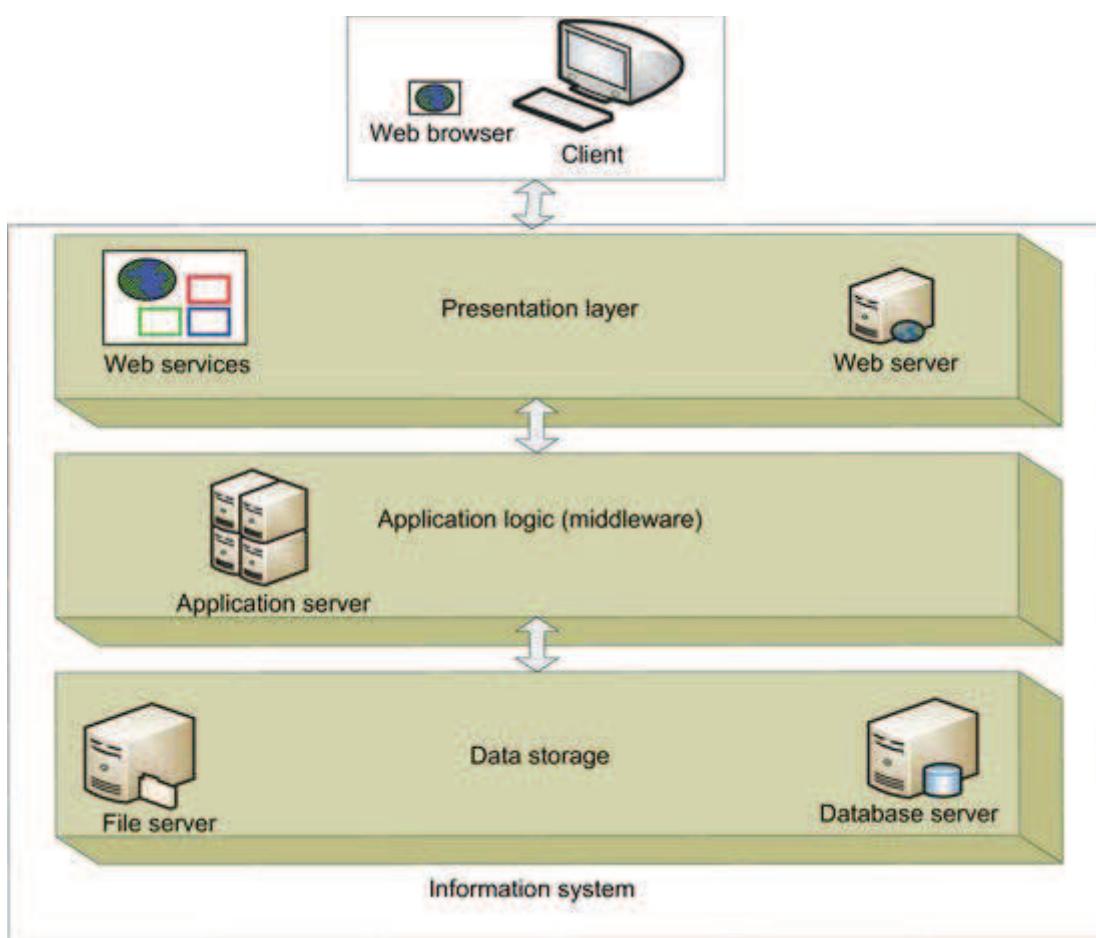
#### WS-RM or WS-Reliable Messaging

WS-RM is one of the best developed of the so-called WS-\* core web service specifications. WS-RM uses message instance counts to allow destination services to recognize message delivery faults (either missing or out-of-order messages). WS-RM somewhat duplicates the capabilities of TCP-IP in this regard, but operates at a different level—namely at the level of user messages and not TCP packets, and from source to destination independent of TCP routing in between. This idea was not fully developed (e.g., multicast messaging is not properly supported). Details can be found at <http://en.wikipedia.org/wiki/WS-ReliableMessaging>.

### 5.1.3 Enterprise Multitier Architecture

Enterprise applications often use multitier architecture to encapsulate and integrate various functionalities. Multitier architecture is a kind of client/server architecture in which the presentation, the application processing, and the data management are logically separate processes. The simplest known multilayer architecture is a two-tier or client/server system. This traditional two-tier, client/server model requires clustering and disaster recovery to ensure resiliency. While the use of fewer nodes in an enterprise simplifies manageability, change management is difficult as it requires servers to be taken offline for repair, upgrading, and new application deployments. Moreover, the deployment of new applications and enhancements is complex and time-consuming in fat-client environments, resulting in reduced availability. A three-tier information system consists of the following layers (Figure 5.4):

- **Presentation layer** Presents information to external entities and allows them to interact with the system by submitting operations and getting responses.
- **Business/application logic layer or middleware** Programs that implement the actual operations requested by the client through the presentation layer. The middle tier can also control user authentication and access to resources, as well as performing some of the query processing for the client, thus removing some of the load from the database servers.
- **Resource management layer** Also known as the data layer, deals with and implements the different data sources of an information system.



**FIGURE 5.4** Three-tier system architecture. Gustavo Alonso, et al, Web Services: Concepts, Architectures and Applications (Data-Centric Systems and Applications). Springer Verlag, 2010. With kind permission from Springer Science+Business Media B.V.

In fact, a three-tier system is an extension of two-tier architecture where the application logic is separated from the resource management layer [21]. By the late 1990s, as the Internet became an important part of many applications, the industry extended the three-tier model to an  $N$ -tier approach. SOAP-based and RESTful web services have become more integrated into applications. As a consequence, the data tier split into a data storage tier and a data access tier. In very sophisticated systems, an additional wrapper tier can be added to unify data access to both databases and web services. Web services can benefit from the separation of concerns inherent in multitier architecture in almost the same way as most dynamic web applications [22].

The business logic and data can be shared by both automated and GUI clients. The only differences are the nature of the client and the presentation layer of the middle tier. Moreover, separating business logic from data access enables database independence.  $N$ -tier architecture is characterized by the functional decomposition of applications, service components, and their distributed deployment. Such an architecture for both web services and dynamic web applications leads to reusability, simplicity, extensibility, and clear separation of component functionalities.

Web services can be seen as another tier on top of the middleware and application integration infrastructure [23], allowing systems to interact with a standard protocol across the Internet. Because each tier can be managed or scaled independently, flexibility is increased in the IT infrastructure that employs  $N$ -tier architecture. In the next section,

we will describe OGSA, as multitier, service-oriented architecture for middleware which describes the capabilities of a grid computing environment and embodies web services to make computing resources accessible in large-scale heterogeneous environments.

## 5.1.4 Grid Services and OGSA

The OGSA [24], developed within the OGSA Working Group of the Global Grid Forum (recently renamed to Open Grid Forum or OGF and being merged with the Enterprise Grid Alliance or EGA in June 2006), is a service-oriented architecture that aims to define a common, standard, and open architecture for grid-based applications. “Open” refers to both the process to develop standards and the standards themselves. In OGSA, everything from registries, to computational tasks, to data resources is considered a service. These extensible set of services are the building blocks of an OGSA-based grid. OGSA is intended to:

- Facilitate use and management of resources across distributed, heterogeneous environments
- Deliver seamless QoS
- Define open, published interfaces in order to provide interoperability of diverse resources
- Exploit industry-standard integration technologies
- Develop standards that achieve interoperability
- Integrate, virtualize, and manage services and resources in a distributed, heterogeneous environment
- Deliver functionality as loosely coupled, interacting services aligned with industry-accepted web service standards

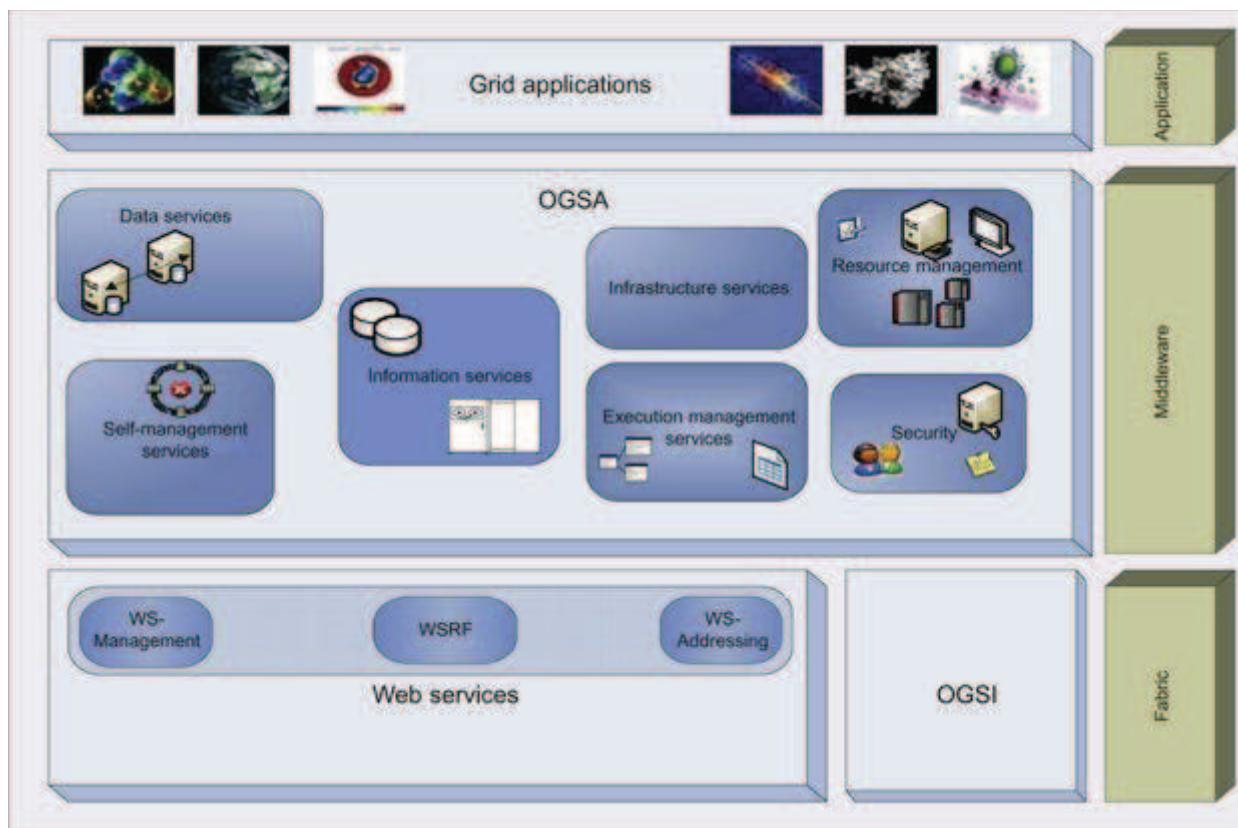
Based on OGSA, a grid is built from a small number of standards-based components, called grid services. [25] defines a grid service as “a web service that provides a set of well-defined interfaces, following specific conventions (expressed using WSDL).” OGSA gives a high-level architectural view of grid services and doesn’t go into much detail when describing grid services. It basically outlines what a grid service should have. A grid service implements one or more interfaces, where each interface defines a set of operations that are invoked by exchanging a defined sequence of messages, based on the *Open Grid Services Infrastructure* (OGSI) [26]. OGSI, also developed by the Global Grid Forum, gives a formal and technical specification of a grid service.

Grid service interfaces correspond to *portTypes* in WSDL. The set of *portTypes* supported by a grid service, along with some additional information relating to versioning, are specified in the grid service’s *serviceType*, a WSDL extensibility element defined by OGSA. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; whereas the conventions address naming and upgradeability. Grid service implementations can target native platform facilities for integration with, and of, existing IT infrastructures.

According to [25], OGSA services fall into seven broad areas, defined in terms of capabilities frequently required in a grid scenario. Figure 5.5 shows the OGSA

architecture. These services are summarized as follows:

- **Infrastructure Services** Refer to a set of common functionalities, such as naming, typically required by higher level services.
- **Execution Management Services** Concerned with issues such as starting and managing tasks, including placement, provisioning, and life-cycle management. Tasks may range from simple jobs to complex workflows or composite services.
- **Data Management Services** Provide functionality to move data to where it is needed, maintain replicated copies, run queries and updates, and transform data into new formats. These services must handle issues such as data consistency, persistency, and integrity. An OGSA data service is a web service that implements one or more of the base data interfaces to enable access to, and management of, data resources in a distributed environment. The three base interfaces, *Data Access*, *Data Factory*, and *Data Management*, define basic operations for representing, accessing, creating, and managing data.
- **Resource Management Services** Provide management capabilities for grid resources: management of the resources themselves, management of the resources as grid components, and management of the OGSA infrastructure. For example, resources can be monitored, reserved, deployed, and configured as needed to meet application QoS requirements. It also requires an information model (semantics) and data model (representation) of the grid resources and services.
- **Security Services** Facilitate the enforcement of security-related policies within a (virtual) organization, and supports safe resource sharing. Authentication, authorization, and integrity assurance are essential functionalities provided by these services.
- **Information Services** Provide efficient production of, and access to, information about the grid and its constituent resources. The term “information” refers to dynamic data or events used for status monitoring; relatively static data used for discovery; and any data that is logged. Troubleshooting is just one of the possible uses for information provided by these services.
- **Self-Management Services** Support service-level attainment for a set of services (or resources), with as much automation as possible, to reduce the costs and complexity of managing the system. These services are essential in addressing the increasing complexity of owning and operating an IT infrastructure.



**FIGURE 5.5** The OGSA architecture. Courtesy of Foster, et al. [24], <http://www.ogf.org/documents/GFD.80.pdf>.

OGSA has been adopted as reference grid architecture by a number of grid projects. The first prototype grid service implementation was demonstrated January 29, 2002, at a Globus Toolkit tutorial held at Argonne National Laboratory. Since then, the Globus Toolkit 3.0 and 3.2 have offered an OGSA implementation based on OGSI. Two key properties of a grid service are transience and statefulness. Creation and destruction of a transient grid service can be done dynamically. The creation and lifetime of OGSA grid services are handled following the “factory pattern,” to be explained in [Section 7.3.1](#). Web service technologies are designed to support loosely coupled, coarse-grained dynamic systems, and hence do not meet all grid requirements, such as keeping state information, and thus they are unable to fully address the wide range of distributed systems OGSA is designed to support.

OGSA applies a set of WSDL extensions to represent the identifiers necessary to implement a grid service instance across any system. These extensions were defined by OGSI. A key extension is the grid service reference: a network-wide pointer to a specific grid service instance, which makes that instance accessible to remote client applications. These extensions, including the *Grid Service Handle* (GSH) and *Grid Service Reference* (GSR), will be described in [Chapter 7](#). These extensions include stateful grid services and the shortcomings of OGSI with its dense and long specifications. Further problems concern incompatibility with some current web service tools and the fact that it takes a lot of concepts from object orientation.

Unlike the nature of web services, this has led to close cooperation between the grid and web service communities. As a result of these joint efforts, the *Web Services Resource Framework* (WSRF) [27], WS-Addressing [28], and WS-Notification (WSN) specifications

have been proposed to OASIS. Consequently, OGSI extensions to web services have been deprecated in favor of new web service standards, and in particular, WSRF. WSRF is a collection of five different specifications. Of course, they all relate to the management of WS-Resources. [Table 5.5](#) depicts WSRF-related interface operations.

**Table 5.5**  
**WSRF and Its Related Specifications**

Specification	Description		
WSRF-Related Specifications	WS-ResourceProperties		Standardizes the definition of the resource properties, its association with the WS interface, and the messages defining the query and update capability against resource properties
	WS-ResourceLifetime		Provides standard mechanisms to manage the life cycle of WS-resources (e.g., setting termination time)
	WS-ServiceGroup		Standard expression of aggregating web services and WS-Resources
	WS-Basefault		Provides a standard way of reporting faults
	WS-Notification	WS-Base Notification	Proposes a standard way of expressing the basic roles involved in web service publish and subscribe for notification message exchange
		WS-BrokeredNotification	Standardizes message exchanges involved in web service publish and subscribe of a message broker
		WS-Topics	Defines a mechanism to organize and categorize items of interest for a subscription known as "topics"
	WS-Addressing		Transport-neutral mechanisms to address web service and messages

Plain web services are usually stateless. This means the web service can't "remember" information, or keep state, from one invocation to another. However, since a web service is stateless, the following invocations have no idea of what was done in the previous invocations. Grid applications generally require web services to keep state information as they interact with the clients or other web services. The purpose of WSRF is to define a generic framework for modeling and accessing persistent resources using web services in order to facilitate the definition and implementation of a service and the integration and management of multiple services. Note that "stateless" services can, in fact, remember state if that is carried in messages they receive. These could contain a token remembered in a cookie on the client side and a database or cache accessed by the service. Again, the user accessing a stateless service can establish state for the session through the user login that references permanent information stored in a database.

The state information of a web service is kept in a separate entity called a resource. A service may have more than one (singleton) resource, distinguished by assigning a unique key to each resource. Resources can be either in memory or persistent, stored in secondary storage such as a file or database. The pairing of a web service with a resource

is called a WS-Resource. The preferred way of addressing a specific WS-Resource is to use the qualified *endpoint reference* (*EPR*) construct, proposed by the WS-Addressing specification. Resources store actual data items, referred to as resource properties. Resource properties are usually used to keep service data values, providing information on the current state of the service, or metadata about such values, or they may contain information required to manage the state, such as the time when the resource must be destroyed. Currently, the Globus Toolkit 4.0 provides a set of OGSA capabilities based on WSRF.

## 5.1.5 Other Service-Oriented Architectures and Systems

A survey of services and how they are used can be found in [29]. Here we give two examples: one of a system and one of a small grid.

### Example 5.3

#### U.S. DoD Net-Centric Services

The U.S. military has introduced a set of so-called Net-Centric services that are to be used in Department of Defense (DoD) software systems to be used on the GiG – Global Information Grid. As shown in Table 5.6, these make different choices of services from OGSA. This is not a radically different architecture, but rather a different layering. Messaging is present in Table 5.6 but viewed as part of WS-\* or a higher application layer in OGSA. (see more details in [http://www.ogf.org/gf/group\\_info/view.php?group=infod-wg](http://www.ogf.org/gf/group_info/view.php?group=infod-wg) for the INFOD standard) [29].

**Table 5.6**

#### Core Global Information Grid Net-Centric Services

Service or Feature	Examples
Enterprise services management	Life-cycle management
Security; information assurance (IA)	Confidentiality, integrity, availability, reliability
Messaging	Publish-subscribe important
Discovery	Data and services
Mediation	Agents, brokering, transformation, aggregation
Collaboration	Synchronous and asynchronous
User assistance	Optimize Global Information Grid user experience
Storage	Retention, organization, and disposition of all forms of data
Application	Provisioning, operations, and maintenance
Environmental control services	Policy

## **Example 5.4**

### **Services in CICC – The Chemical Informatics Grid**

The goal of this project is to support cluster analysis, data mining, and quantum simulation/first principles calculations on experimentally obtained data on small molecules with potential use in drug development. Small molecule data is gathered from NIH PubChem and DTP databases, with additional large molecule data available from service-wrapped databases such as the Varuna, Protein Data Bank, PDBBind, and MODB. NIH-funded High Throughput Screening centers are expected to deluge the PubChem database with assays of the next several years, making the automated organization and analysis of data essential.

Data analysis applications are interestingly combined with text analysis applications applied to journal and technical articles to make a comprehensive scientific environment. Workflow is a key part of this project as it encodes scientific use cases. Many CICC services are based on Cambridge University's WWMM project ([www.pmr.cam.ac.uk/wiki/Main\\_Page](http://www.pmr.cam.ac.uk/wiki/Main_Page)) led by Peter Rust. Table 5.7 shows the mix of system and application services used in CICC. This is a small project and larger grids would have many more services. Details can be found at the web site: <http://www.chembiogrid.org>.

**Table 5.7**

#### **Services and Standards Used in CICC**

Service Name	Description
Workflow/Monitoring/Management Services	Uses Taverna from the UK e-Science Program/OMII or mashups written in scripting languages.
Authentication/Authorization	Currently all services are openly available.
Registry and Discovery	Will inherit registry services from other grids.
Portal and Portlets	Use a JSR 168-based portal.
File Services	No specialized service. URLs are used for naming files and simple remote download. Services developed previously for other grids can be used for uploads.
NIH DTP Database Services	Access to the NIH Developmental Therapeutics Program (DTP)'s database of molecular screens against 60 cancer cell lines. This is a free service provided by the NIH and used by Chembiogrid.
PubMed Search Service	Searchable online database of medical journal articles. CICC develops harvesting services of the abstracts that can be combined with text analysis applications such as OSCAR3.
SPRESI Services	Clients/service proxies to the commercial SPRESI service ( <a href="http://www.spresi.com/">www.spresi.com/</a> ).

	This scientific database houses molecular and reaction data and references and patents.
Varuna Database Service	Molecular structure and more detailed information (such as force fields).
VOTables Data Tables Web Service	CICC-developed web service based on the National Virtual Observatory's VOTables XML format for tabular data.
Specific Applications: BCI, OpenEye, Varuna, AutoGEFF	CICC inherits job management services from other grids (including one based on Apache Ant) for managing the execution of both commercial and in-house developed high-performance computing applications.
Condor and BirdBath	Examine the use of Condor and its SOAP interface (BirdBath) as a super-scheduler for Varuna applications on the TeraGrid.
ToxTree Service	Wraps an algorithm for estimating toxic hazards in a particular compound. Useful in combination with other clustering programs in a workflow.
OSCAR3 Service	Based on OSCAR3 by the WWMM group, performs text analysis on journal articles and other documents to extract (in XML) the chemistry-specific information. SMILES assigned to well-known compounds. Works with traditional database and clustering algorithms.
CDK Services	CICC has developed a number of simple services based on the Chemistry Development Kit (CDK). These include similarity calculations, molecular descriptor calculations, fingerprint generators, 2D image generators, and 3D coordinate molecular generators.
OpenBabel Service	Converts between various chemical formats (such as between InChI and SMILES).
InChIGoogle	For a given InChI (a string specification of a molecular structure), performs a Google search to return a page-ranked list of matches.
Key Interfaces/Standards/Software Used	WSDL, SOAP (with Axis 1.x). CML, InChI, SMILES, Taverna SCUFI, JSR-168 JDBC Servlets, VOTables
Unused Interfaces/Software	WS-Security, JSDL, WSRF, BPEL, OGSA-DAI

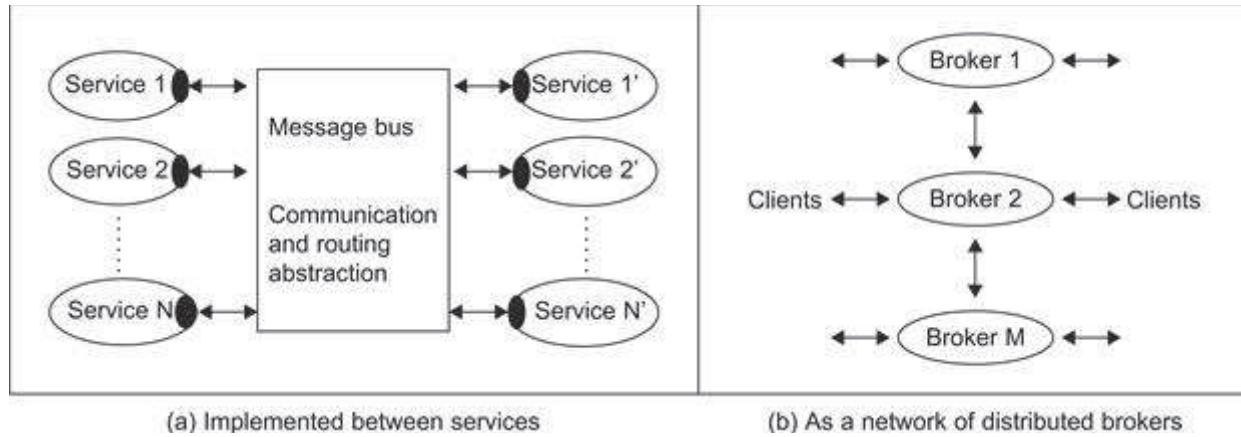
## 5.2 Message-Oriented Middleware

This section introduces message-oriented middleware for supporting distributed computing. The study included enterprise bus, publish-subscribe model, queuing, and messaging systems.

### 5.2.1 Enterprise Bus

In the previous section, we described services and service architectures. These services, by definition, interact with messages with a variety of different formats (APIs), wire protocols, and transport mechanisms. It is attractive to abstract the communication mechanism so that services can be defined that communicate independent of details of the implementation. For example, the author of a service should not need to worry that a special port is to be used to avoid firewall difficulties or that we need to use UDP and special fault tolerance approaches to achieve satisfactory latency on a long-distance communication. Further, one may wish to introduce a wrapper so that services expecting messages in different styles (say, SOAP, REST, or Java RMI) can communicate with each

other. The term “enterprise service bus” or ESB [30,31] refers to the case where the bus supports the convenient integration of many components, often in different styles. These remarks motivate the messaging black box abstraction shown in [Figure 5.6](#).



**FIGURE 5.6** Two message bus implementations between services or using a broker network.

One does not open a channel between source and destination, but rather injects a message into the bus with enough information to allow it to be delivered correctly. This injection is performed by code loaded into each service and represented by the filled ovals as client interfaces in [Figure 5.6\(a\)](#). The message bus is shown linking services in this figure, but it can work with any software or hardware entity sending and receiving messages. A simple example could be desktops or smart phones as the clients. Further, such buses can be implemented internally to an application, or in a distributed fashion. In the latter case, the message bus is typically implemented as a set of “brokers” shown in [Figure 5.6\(b\)](#).

The use of multiple brokers allows the bus to scale to many clients (services) and large message traffic. Note that the brokers of [Figure 5.6\(b\)](#) are “just” special servers/services that receive messages and perform needed transformations and routing on them and send out new messages. There is a special (simple) case of message buses where the brokers shown in [Figure 5.6\(b\)](#) are not separate servers but are included in the client software. Note that such buses support not just point-to-point messaging but broadcast or selective multicast to many recipient clients (services).

Often, one implements brokers as managers of queues, and software in this area often has MQ or “Message Queue” in its description. An early important example is MQSeries [32] from IBM which is now marketed as the more recent WebSphereMQ [32,33]. Later, when we study cloud platforms in [Chapter 8](#), we will find that both Azure and Amazon offer basic queuing software. A typical use of a message queue is to relate the master and workers in the “farm” model of parallel computing where the “master” defines separate work items that are placed in a queue which is accessed by multiple workers that select the next available item. This provides a simple dynamically load-balanced parallel execution model. If necessary, the multiple brokers of [Figure 5.6\(b\)](#) can be used to achieve scalability.

## 5.2.2 Publish-Subscribe Model and Notification

An important concept here is “publish-subscribe” [34] which describes a particular model for linking source and destination for a message bus. Here the producer of the message (publisher) labels the message in some fashion; often this is done by associating one or more topic names from a (controlled) vocabulary. Then the receivers of the message (subscriber) will specify the topics for which they wish to receive associated messages. Alternatively, one can use content-based delivery systems where the content is queried in some format such as SQL.

The use of topic or content-based message selection is termed message filtering. Note that in each of these cases, we find a many-to-many relationship between publishers and subscribers. Publish-subscribe messaging middleware allows straightforward implementation of notification or event-based programming models. The messages could, for example, be labeled by the desired notifying topic (e.g., an error or completion code) and contain content elaborating the notification [34].

## 5.2.3 Queuing and Messaging Systems

There are several useful standards in this field. The best known is the Java Message Service (JMS) [35] which specifies a set of interfaces outlining the communication semantics in pub/sub and queuing systems. *Advanced Message Queuing Protocol* (AMQP) [36] specifies the set of wire formats for communications; unlike APIs, wire formats are cross-platform. In the web service arena, there are competing standards, WS-Eventing and WS-Notification, but neither has developed a strong following. Table 5.8 compares a few common messaging and queuing systems. We selected two cloud systems: Amazon Simple Queue and Azure Queue.

---

**Table 5.8**

**Comparison of Selected Messaging and Queuing Systems**

---

System Features	Amazon Simple Queue [41]	Azure Queue [42]	ActiveMQ	MuleMQ	WebSphere MQ	Narada Brokering
AMQP compliant	No	No	No, uses OpenWire and Stomp	No	No	No
JMS compliant	No	No	Yes	Yes	Yes	Yes
Distributed broker	No	No	Yes	Yes	Yes	Yes
Delivery guarantees	Message retained in queue for four days	Message accessible for seven days	Based on journaling and JDBC drivers to databases	Disk store uses one file/channel, TTL purges messages	Exactly-once delivery supported	Guaranteed and exactly-once
Ordering guarantees	Best effort, once delivery, duplicate messages exist	No ordering, message returns more than once	Publisher order guarantee	Not clear	Publisher order guarantee	Publisher- or time-order by Network Time Protocol
Access model	SOAP, HTTP-based GET/POST	HTTP REST interfaces	Using JMS classes	JMS, Adm. API, and JNDI	Message Queue Interface, JMS	JMS, WS-Eventing
Max. message Buffering	8 KB	8 KB	N/A	N/A	N/A	N/A
Time decoupled delivery	N/A	Yes	Yes	Yes	Yes	Yes
Support for web services	Up to four days; supports timeouts	Up to seven days	Yes	Yes	Yes	Yes
Security scheme	Based on HMAC-SHA1 signature, Support for WS-Security 1.0	Access to queues by HMAC SHA256 signature	Authorization based on JAAS for authentication	Access control, authentication, SSL for communication	SSL, end-to-end application-level data security	SSL, end-to-end application-level data security, and ACLs
Transports	SOAP-based interactions	REST interfaces	REST	REST	REST, SOAP interactions	WS-Eventing
Subscription formats	HTTP/HTTPS, SSL	HTTP/HTTPS	TCP, UDP, SSL, HTTP/S, Multicast, in-VM, JXTA	Mule ESB supports TCP, UDP, RMI, SSL, SMTP, and FTP	TCP, UDP, Multicast, SSL, HTTP/S	TCP, Parallel TCP, UDP, Multicast, SSL, HTTP/S, IPsec
	Access is to individual queues	Access is to individual queues	JMS spec allows for SQL selectors; also access to individual queues	JMS spec allows for SQL selectors; also access to individual queues	JMS spec allows SQL selectors; access to individual queues	SQL selectors, regular expressions, <tag, value> pairs, XQuery and XPath

We also list MuleMQ [37], which is the messaging framework underlying the ESB [30,31] system Mule, developed in Java, of which there are 2,500 product deployments as of 2010. The focus of Mule is to simplify the integration of existing systems developed using JMS, Web Services, SOAP, JDBC, and traditional HTTP. Protocols supported within Mule include POP, IMAP, FTP, RMI, SOAP, SSL, and SMTP. ActiveMQ [38] is a popular Apache open source message broker while WebSphereMQ [33] is IBM's enterprise message bus offering. Finally, we list the open source NaradaBrokering [39] that is notable for its broad range of supported transports and was successfully used to support a software *Multipoint Control Unit (MCU)* for multipoint video conferencing and other collaboration capabilities.

Note that the four noncloud systems support JMS. Also, some key features of messaging systems are listed in the table but are not discussed in this brief section. These are security approach and guarantees and mechanisms for message delivery. Time-decoupled delivery refers to situations where the producer and consumer do not have to be present at the same time to exchange messages. Fault tolerance is also an important property: Some messaging systems can back up messages and provide definitive guarantees. This table is only illustrative and there are many other important messaging

systems. For example, RabbitMQ [40] is a new impressive system based on the AMQP standard.

## 5.2.4 Cloud or Grid Middleware Applications

Three examples are given here to illustrate the use of the NaradaBrokering middleware service with distributed computing. The first example is related to environmental protection. The second is for Internet conferencing and the third is for earthquake science applications.

### Example 5.5

#### Environmental Monitoring and Internet Conference Using NaradaBrokering

The GOAT project at Clemson University is part of the Program of Integrated Study for Coastal Environmental Sustainability (PISCES), which addresses environmental sustainability issues that can accompany coastal development. The current study incorporates groundwater monitoring, surface water quality and quantity monitoring, weather, and a variety of ecological measurements. The project utilizes the publish-subscribe messaging system, NaradaBrokering, to provide a flexible and reliable layer to move observation data from a wide variety of sensor sources to users that have diverse data management and processing requirements. NaradaBrokering can display environmental sensors.

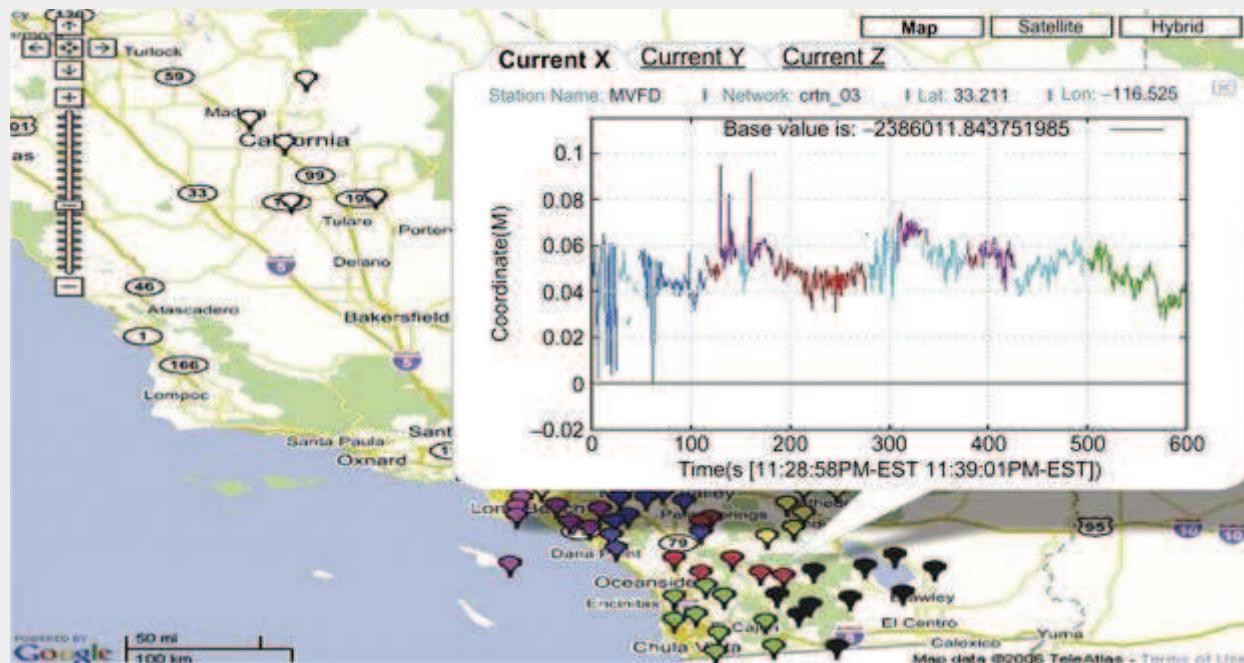
The commercial Internet Meeting software Anabas ([www.anabas.com](http://www.anabas.com)) incorporates support for sharing applications besides incorporating support for shared whiteboards, and chat tools. Anabas uses NaradaBrokering for its content dissemination and messaging requirements. On a daily basis, Anabas supports several online meetings in the United States, Hong Kong, and mainland China. Note NaradaBrokering supports audio-video conferencing (using UDP) as well as other collaborative applications using TCP. Dynamic screen display published to NaradaBrokering can be displayed on collaborating clients.

### Example 5.6

#### QuakeSim Project for Earthquake Science

The NASA-funded QuakeSim project uses NaradaBrokering to manage

workflows that connect distributed services, and to support GPS filters delivering real-time GPS data to both human and application consumers as shown in [Figure 5.7](http://quakesim.jpl.nasa.gov/) (<http://quakesim.jpl.nasa.gov/>). The GPS application is an important application similar to [Example 5.4](#) where publish-subscribe systems manage sensor networks. In fact, one can consider webcams as sensors (as they produce real-time streams) and so [Example 5.5](#) is also of this type. Clouds are an important implementation for this type of application as brokers can be added on demand to support myriad dynamic sensors from cell phones to military or civil sensors responding to an extreme event. The display of GPS sensors is managed by NaradaBrokering. The map displays the time series produced by GPS stations.



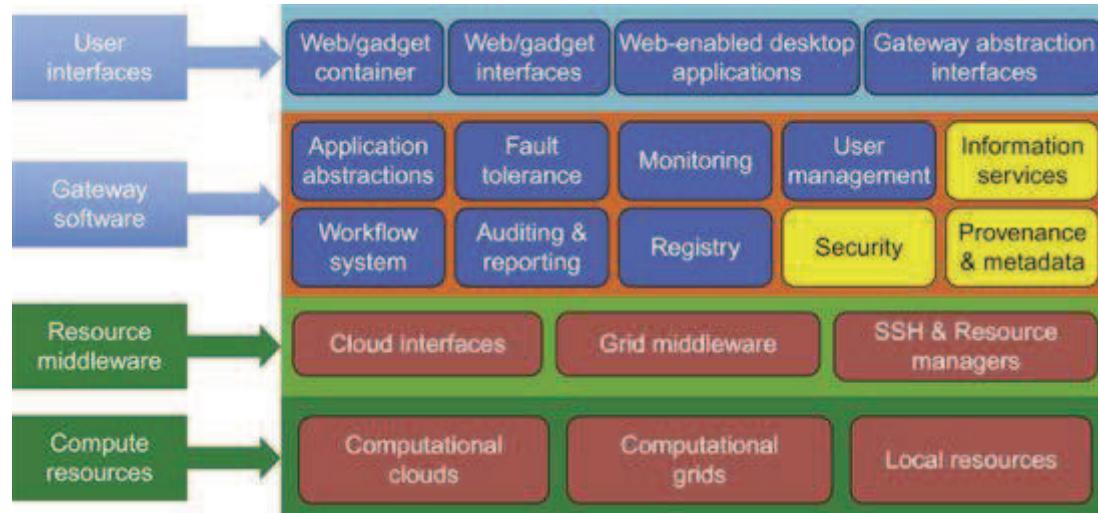
**FIGURE 5.7** Display of GPS sensors managed by NaradaBrokering in Southern California; the map displays the time series produced by one of the GPS stations. (<http://quakesim.jpl.nasa.gov/>).

## 5.3 Portals and Science Gateways

Science gateways [43,44] are tools that enable interactive, web-based science, education, and collaboration. Gateways provide user-centric environments for interacting with remote computational resources through user interfaces that are typically (but not exclusively) built with web technologies. Although they have superficial similarities to web sites and may leverage tools such as content management systems for their presentation layers, gateways are much more complicated entities. Science gateways are also termed portals. This section describes the general architecture of gateways, surveys several prominent examples, and discusses example gateway-building software.

We can categorize gateway software as “turnkey” solutions, exemplified by HUBzero; and “toolbox” solutions, exemplified by the Open Gateway Computing Environments project. Turnkey gateway software provides an end-to-end solution for building gateways, including hosting. Toolbox gateway software provides tools that solve specific problems and can be integrated into customized software stacks. The SimpleGrid project [45] is another toolkit example. HUBzero and OGCE software will be described in more detail shortly.

Figure 5.8 provides a high-level overview of the grouping of components that make up a gateway. The bottom tier is the resource layer, which may include campus computing clusters and storage systems, national-scale grid resources such as the TeraGrid [46] and the Open Science Grid [47], and computational cloud resources. The second tier consists of middleware for interacting with these resources. Common middleware examples include Globus [48] (providing job execution and remote file management), Condor [49] (providing job scheduling), and iRods [50] (providing data and metadata movement and management). Middleware systems typically expose secure, network-accessible APIs and developer libraries as well as providing numerous command-line tools.



**FIGURE 5.8** A gateway component software stack for scientific applications.

Tier 2 software and services run on the resources of Tier 1. The next two tiers are not tied to these resources and can run on separate host machines. Tier 3 is the gateway software layer and consists of service components that manage the execution of scientific applications and data on the Tier 1 resources through the Tier 2 interfaces. Execution management for both data and science applications can be decoupled into several components: the initial invocation, monitoring, fault tolerance, and task coupling as workflows. Security considerations [51,52] (such as authentication and authorization) permeate all tiers, but we locate the component in Tier 3 since the user’s identity is established here.

We also place user and group management (including social networking) in this tier, along with third-party information services [53]. Finally, the top tier (Tier 4) is the user presentation layer. Presentation layers may be built with a number of different tools,

including content management systems, Open Social gadgets, and desktop applications, as described in more detail shortly. Figure 5.8 is flexible enough to describe most gateways. We next make the description more concrete by reviewing two example science gateways: GridChem and UltraScan. Both gateways have been used for significant scientific research.

### 5.3.1 Science Gateway Exemplars

Three examples are given here to illustrate the gateway concept in scientific applications using computational grids or Internet clouds.

#### Example 5.7

##### Computational Chemistry Grid (CCG)

The CCG is also known as GridChem ([www.gridchem.org](http://www.gridchem.org)) [54,55], which is one of the most heavily used Science Gateways in TeraGrid. GridChem provides a grid-enabled desktop interface that enables users to set up, launch, and manage computational chemistry simulations on the TeraGrid. Applications supported by GridChem include Gaussian, CHARMM, and GAMESS.

These are computationally demanding parallel applications that require supercomputers, although supporting applications—input validation and visualization tools—can run on the user's desktop. GridChem pioneered gateway concepts such as “community users,” which allow users to access resources through a shared allocation, that are now commonplace in many Science Gateways. GridChem has delivered significant scientific accomplishments in the form of scientific publications that acknowledge and highlight the GridChem cyber-infrastructure’s utility for computational chemistry [56].

#### Example 5.8

##### UltraScan Biophysics Gateway

UltraScan [57,58] has developed a TeraGrid Science Gateway for the high-resolution analysis and modeling of hydrodynamic data from an analytical ultracentrifuge (AUC). This application is used for the solution-based study of biological macromolecules and synthetic polymers by biochemists, biophysicists, and material scientists for fitting experimental data to finite

element solutions of flow equations. More than 700 biochemists, biophysicists, biologists, and material scientists worldwide rely on UltraScan software for analysis of their experimental data. The software has assisted in understanding an array of disease processes, including cancer, neurodegenerative diseases, HIV/AIDS, diabetes, Huntington's disease, and aging studies. By conservative estimates, UltraScan has contributed to more than 250 peer-reviewed publications including 23 known publications from 2009 [59].

UltraScan makes its core experimental analysis software available as an online service to scientific users through its Science Gateway. This analysis software is computationally demanding and must be run on clusters and supercomputers. UltraScan job management services (Tier 3) hide complexities and provide fault tolerance for the experimental scientists using the portal. Although it uses TeraGrid for some of its computing power, UltraScan needs to span multiple resource providers: It also uses university clusters and would like to extend its resources to include international grids in Germany and Australia. The key to UltraScan's success and growth over the next three years is to provide the ability to manage jobs across multiple cyberinfrastructure resources that are not collectively part of a single, managed grid.

Gateways such as UltraScan provide an example where code optimization and efficient use of compute resources are done by experts in the field and shared with hundreds of end users. The gateways lower the entry barrier for analyzing the data on high-end resources. As an example, the UltraScan gateway provides an optimal solution for solving large non-negative least squares problems that arise in data analysis [60,61]. Solving these problems requires significant compute resources. The procedure improves compute resource utilization targeting the inverse problem involved in modeling analytical ultracentrifugation experimental data. Solving large non-negatively constrained least squares systems is frequently used in the physical sciences to estimate model parameters which best fit experimental data.

AUC is an important hydrodynamic experimental technique used in biophysics to characterize macromolecules and to determine parameters such as molecular weight and shape. Newer AUC instruments equipped with *multi-wavelength* (MWL) detectors have recently increased the data sizes by three orders of magnitude. Analyzing the MWL data requires significant compute resources. UltraScan gateway bridges these requirements for end users by providing both the procedures and the capabilities to execute them on supercomputing resources.

## Example 5.9

### The nanoHUB.org Gateway

The [nanoHUB.org](#) gateway is operated by the National Science Foundation (NSF)-funded Network for Computational Nanotechnology (NCN) to support the National Nanotechnology Initiative in the United States and worldwide and to accelerate the transformation of nanoscience into nanotechnology. Since 2002, the community using [nanoHUB.org](#) has grown from 1,000 users mostly at Purdue University to more than 290,000 visitors each year from 172 countries worldwide. From August 2009 to July 2010, some 8,600 users accessed more than 170 nanotechnology simulation tools and launched 340,000 simulation runs.

Today [nanoHUB.org](#) hosts more than 2,000 content items such as 170 online simulation tools and 43 complete courses, plus tutorials, research seminars, and teaching materials. All of [nanoHUB.org](#)'s services are free of charge to the user. All resources on [nanoHUB.org](#) are presented in scholarly form with title, authors, abstract, and archival citation information. To date, there are more than 560 citations in academic literature to [nanoHUB.org](#) and the tools, seminars, and other resources published there. Also to date, 379 courses at 131 institutions of higher education have used [nanoHUB.org](#) resources. The journal citations and the documented use in classrooms is evidence that [nanoHUB.org](#) resources aid both research and education.

### 5.3.2 HUBzero Platform for Scientific Collaboration

HUBzero is an open source software platform used to create web sites or “hubs” for scientific collaboration, research, and education.<sup>1</sup> It has a unique combination of capabilities that appeals to many people engaged in research and educational activities. Like [YouTube.com](#), HUBzero allows people to upload content and “publish” to a wide audience, but instead of being restricted to short video clips, it handles many different kinds of scientific content. In that respect, HUBzero resembles MIT’s OpenCourseWare [64], but it also integrates the content with collaboration capabilities. Like Google Groups, HUBzero lets people work together in a private space where they can share documents and send messages to one another. Like Askville on [Amazon.com](#), HUBzero lets people ask questions and post responses, but about scientific concepts instead of products.

Perhaps the most interesting feature of HUBzero is the way it handles simulation and modeling programs, or “tools.” Like [SourceForge.net](#), HUBzero allows researchers to work collaboratively on the source code of their simulation programs and share those programs with the community. But instead of sharing only by offering source code bundles to download, HUBzero also offers live *published* programs available for use

instantly and entirely within an ordinary web browser. The simulation engines run start to finish on computational resources selected for that hub.

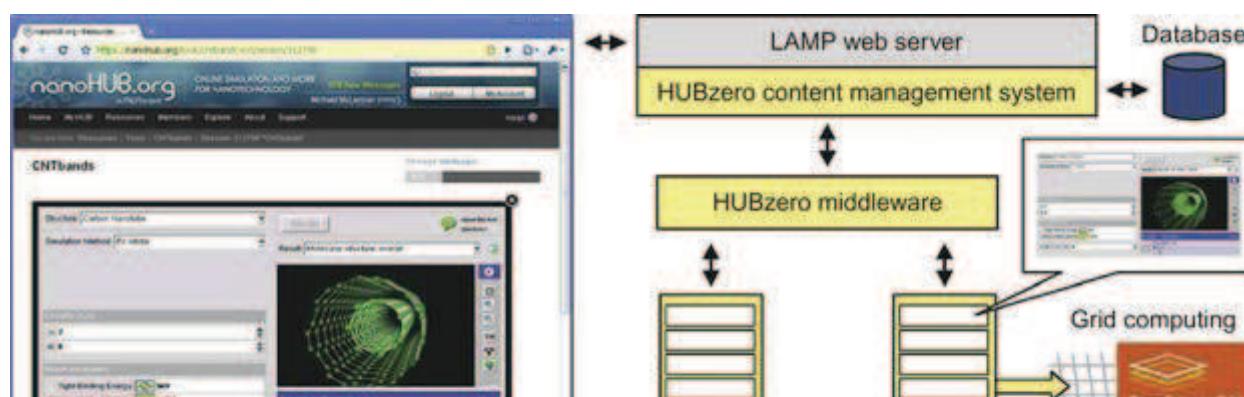
Computationally demanding runs can be dispatched to remote resources in a way that is completely transparent to users. Tools are driven by friendly GUIs that enable end-to-end operation of the simulation process encompassing setup, execution, and data visualization. Many GUIs are built using HUBzero's Rappture toolkit, which lets researchers compare simulation results from multiple runs and ask "What if?" questions. In effect, each hub powered by HUBzero is an "app store" for a scientific community connected to a cloud of resources for app execution, complete with a library of training materials and other collaboration features to support app use.

HUBzero was created by Purdue University and the NSF-funded NCN to power its web site at [nanoHUB.org](http://nanoHUB.org) [62,63]. Today, the same HUBzero software powers 30 similar gateways covering a wide variety of disciplines in engineering and science. Here are three examples:

- **GlobalHUB.org** 29,000 active users, online since December 2007 Leverages the group functionality within HUBzero to support engineering education on a global scale. Teams of students work together in groups on a variety of engineering projects.
- **cceHUB.org** 2,400 active users, online since June 2008 Approaches cancer care from an "engineering" perspective by gathering a database of blood samples from patients, extracting proteomic/metabolomic data, mining the data to find biomarker patterns, and modeling the efficacy of cancer treatments.
- **NEES.org** 15,000 users, online since August 2010 Home of the NSF Network for Earthquake Engineering Simulation, which catalogs experimental data from 14 institutions that are simulating earthquake conditions in the lab. The site also hosts modeling tools used to visualize and analyze the data [64a].

### 5.3.2.1 The HUBzero Architecture

On the surface, each HUBzero-powered gateway is a web site built with the widely used, open source "LAMP" architecture—a Linux operating system, an Apache web server, a MySQL database, and PHP web scripting. HUBzero adds a scientific content management system, the open source Rappture toolkit used to create GUIs for simulation programs, and unique middleware for hosting simulation tools and scientific data, as shown in Figure 5.9.





**FIGURE 5.9** The HUBzero architecture and its major functional components.

Each tool description page includes a “launch” button. When a user presses that button, the middleware allocates a session container on the tool execution host, starts the X11 windowing system within the container, starts the tool, and connects the session back to the user’s web browser via *Virtual Network Computing* (VNC) [65]. To the user, it appears that the tool is running in the browser, but it actually runs within the hub environment, where it has access to local computation and visualization clusters and remote computing resources such as TeraGrid, Open Science Grid, and Purdue’s DiaGrid [66]. Sessions can be shared among users collaborating offline or in real time, and they are persistent, so the user can close the browser window and revisit the same session later.

Unlike other portals and cyber environments, the tools in a hub are interactive and engaging. Users can zoom in on a graph, rotate a molecule, and probe the isosurfaces in a 3D volume [67]—all interactively without having to wait for a web page to refresh. Users can visualize results without reserving time on a supercomputer or waiting for a batch job to engage. Each hub can host an unlimited number of tools uploaded by its community members, and the tools are deployed without having to rewrite code for the web. The computational demand of these tools may vary from seconds on a single core to hours and days on multiple, possibly large numbers of cores [68,69].

Each session runs in a restricted virtual environment implemented using OpenVZ [70], which controls access to file systems, networking, and other system resources. Users can see their files and processes, but cannot see others on the system, cannot launch attacks on other hosts, and cannot exceed their file quota. If a tool already has a GUI that runs under Linux/X11, it can be deployed as is in a matter of hours. If not, tool developers can use HUBzero’s Rappture toolkit (<http://rappture.org>) to create a GUI with very little effort. Developers start by defining the inputs and outputs for their tool as a hierarchy of data objects written in an XML format. The data objects include simple elements, such as integers, Boolean values, numbers with units, and sets of mutually exclusive choices, as well as more complex objects, such as physical structures, finite element meshes, and molecules.

Rappture reads the XML description of a tool and generates a standard GUI for the tool automatically. The GUI prompts for values, launches simulation jobs, and loads results for visualization. The underlying simulation code uses an API to retrieve input values and store output results. Rappture includes libraries for C/C++, FORTRAN, MATLAB, Java, Python, Perl, Ruby, and Tcl/Tk, so the underlying simulator is not constrained to a particular language, but can be written in the developer’s language of choice. The tool shown in Figure 5.9, for example, is a Rappture GUI for a MATLAB program that simulates carbon nanotubes.

## 5.3.2 Operational Features

Having instant access to a large collection of simulation tools opens up new capabilities, but also introduces a new set of issues. Users want to know more about the tools and the underlying physics they encode. They may discover a bug and wish to report it or question whether results are correct. They want to exchange ideas about new models and new features within the platform. HUBzero has evolved beyond a simple repository to include many features supporting these sorts of collaborations.

### 5.3.2.3 Ratings and Citations

The hub philosophy is not to judge the quality of each resource before it is posted, but rather to post resources and help the community assess the quality. Registered users are allowed to post five-star ratings and reviews for each resource. Users can also post citations of each resource in academic literature. The ratings and citations for each resource are combined with web analytics (measuring popularity) to produce a single number on a scale of 0 to 10, called the ranking, to represent the quality of the resource.

### 5.3.2.4 Content Tagging

Items on a hub are categorized by a series of tags, in much the same way that photos are tagged on a photo sharing site. This makes it easy to browse categories of resources, or to find resources that intersect two or more categories.

### 5.3.2.5 User Support Area

From time to time, users will have problems with logins, have questions about tools, and may otherwise need assistance. The HUBzero software comes with a built-in user support area. Users can click on the Help or Support link near the top of any page and fill out a form to file a support ticket. Tickets can be handled by the staff managing the hub, or forwarded to the researchers supporting various simulation tools. Some questions are beyond the understanding of the hub staff, or even beyond the knowledge of a single researcher.

HUBzero includes a question-and-answer forum patterned after Askville on [Amazon.com](#) that can engage the whole community. Any registered user can post a question, and other users can provide answers. At some point, the best answer is chosen as the “final” answer by the person who asked the question, and participants earn points as a reward for their effort. Points can be used as bragging rights, or as currency to purchase t-shirts and other merchandise in a hub store. The list of past questions/answers forms a knowledge base upon which a community can draw for immediate help with a similar problem.

Other issues are beyond software problems and physics questions, but really entail requests for tool improvements and new features. Such requests are tracked on a “wish list” for each tool and also on a wish list for the entire hub. All tickets, questions, and wishes are managed by the HUBzero software. Various administrators, software developers, and community members are given access to these facilities to help address

the community support issues.

### 5.3.2.6 Wikis and Blogs

Each hub supports the creation of “topic” pages, which are wiki pages with a specific list of authors. Other users can add comments to a topic page and may suggest changes, which the original authors can choose to approve. Users can be added as coauthors for a page, so they can make further changes without approval. Ownership of a page can also be given to the entire community, so anyone can make changes without approval, in a wiki-like manner.

### 5.3.2.7 Usage Metrics

Each hub reports extensive metrics about how its resources are being used, just a few of which include the total number of users in a given period, the number of web hits, the number of simulation jobs launched, and CPU hours used. Metrics are reported down to the level of each individual resource, so everyone can see how many users have accessed a particular tool, or how many times a seminar has been viewed. Usage numbers are rolled up to provide an overview of usage for interesting categories, such as the total number of users that have accessed all resources published by a particular author. These usage metrics provide an incentive to use HUBzero-powered science gateways.

### 5.3.2.8 Future Directions

The development of HUBzero is driven largely by the projects using it. Although HUBzero started with a strong emphasis on simulation and modeling, it is evolving to include data management capabilities as well. Projects such as the cancer care engineering [cceHUB.org](#) and the earthquake engineering [NEES.org](#) are creating mechanisms for users to define, upload, publish, annotate, and analyze various types of structured data sets. Projects such as [GlobalHUB.org](#) are improving the group spaces where people exchange files and work together in a private context.

As hubs gather more tools, researchers see the need for stringing tools together in a workflow to tackle larger questions. A pill dissolution profile generated by one tool on [pharmaHUB.org](#), for example, could be fed into a model of a patient’s digestive track, and those results could be used to compute the amount of active ingredient in the patient’s bloodstream over time. That chain of tools may need to be run hundreds of times to perform an overall sensitivity analysis, to optimize a design goal, or to quantify uncertainty in the output. A hub should not only catalog tools, but connect them one to another, to grid computing resources, and ultimately, to the researcher with a question to answer.

## 5.3.3 Open Gateway Computing Environments (OGCE)

The OGCE project [71] is a provider of open source gateway software that is used in several collaborating gateways [72]. OGCE consists of components that can be either

used individually or integrated together to provide more comprehensive solutions for remote scientific application management. OGCE component tools include the following:

- **OGCE Gadget container** [73], a Google tool for integrating user interface components
- **XRegistry**, a registry service for storing information about other online services and workflows
- **XBaya** [74], a workflow composer and enactment engine
- **GFAC** [75], a factory service that can be used to wrap command-line-driven science applications and make them into robust, network-accessible services
- **OGCE Messaging Service** supports events and notifications across multiple cooperating services

OGCE's strategy is based on the toolkit model. This strategy has been shaped by the TeraGrid science gateway program and its wide variety of gateways. There are obviously many frameworks, programming languages, and tools for building web-based gateways and providing advanced capabilities. It has been our experience that many gateways benefit from lightly coupled tools that can be taken collectively or in part. In the toolkit model, gateways can take a specific tool or tools and integrate them into the gateway's existing infrastructure. For example, a gateway such as UltraScan benefits from a reliable job submission tool that hides differences between Globus GRAM versions; GFAC is a candidate tool. GridChem and ParamChem want to extend their job submission capabilities to include scientific workflows; XBaya and its supporting tools can be used.

The OGCE tools focus on scientific application and workflow management and defer issues such as data and metadata management to other projects. There is significant variation in the characteristics exhibited by the scientific applications, so intermediate, application-specific services between the user interfaces and the generic grid middleware are desirable. The gateway software layer (Tier 3 of [Figure 5.8](#)) must accommodate the complexities of a particular domain and provide a software infrastructure that bridges the gap between the user interface layer and the grid middleware.

Consequently, many science gateways use scientific web services along with workflow systems to build the gateway software. Application-specific web services provide the bridge between generic grid middleware and the specific needs of a gateway. Workflows go a step further by assembling multiple services and steps into scientific use cases. Workflows may be implicit in the design of the gateway or they may be explicitly exposed to users. Registries are services that can be used to look up other services and workflows. Finally, messaging systems are used for different distributed components to communicate. For example, users need a mechanism for monitoring a long-running workflow.

### 5.3.3.1 Workflows

The OGCE scientific workflow system [74,76,77] provides a programming model that allows the scientist to program experiments using application web services developed with the GFAC service that abstract the complexities of the underlying middleware. The workflow system enables scientists to author experiments that can be stored, replayed, and shared with others. The workflow suite is bundled with interfaces for composition,

execution, and monitoring of the workflows. Salient features include support for long-running applications and steering/dynamic user interactions.

OGCE's software stack is designed to be flexible in its coupling of various components to harness end-to-end, multiscale gateway infrastructures. Individual OGCE tools can be integrated into gateway deployments; likewise, other standard specification-based tools can be swapped within OGCE's software stack. As a specific example, the OGCE workflow system provides (with its XBaya frontend) a graphical interface to browse various application service registries (such as the OGCE's XRegistry).

From these registries, users can construct task graphs as workflows. The representation is captured as an abstract, high-level, workflow-neutral format, which can be translated into workflow execution specific syntax. Currently, integrations of BPEL [78], Jython, Taverna SCUFL [79], and Pegasus DAX [80] have all been demonstrated and exist at various levels of support. By default, the workflow enactment is facilitated by an open source BPEL implementation, the Apache *Orchestration and Director Engine* (ODE) [81], that OGCE developers have enhanced to support long-running scientific workflows on computational grids.

### 5.3.3.2 Scientific Application Management

Wrapping scientific applications as remotely accessible services is a common task for science gateways. Gateways that wish to offer many application services need a simple way to quickly wrap these applications. Task wrapping is only one step, however, in making an application available through a gateway. Grid middleware solutions (Tier 2 of Figure 5.8) abstract heterogeneous resources and offer a single unified job management interface to queuing systems. However, both the resources and grid middleware are still highly complex, so it is a difficult task to provide production-quality scientific application services that have the reliability and scalability (in number of users) needed by successful gateways. Gateways have often reinvented solutions for all of these problems.

The OGCE's GFAC tool is designed to wrap command-line-driven executables and make them available as external services. The primary goal of GFAC is to provide a general solution to the application-wrapping problem that can be reused by gateways as a plug-in service in their deployed infrastructure. GFAC generates web services that can be accessed through clients written in Java, Perl, PHP, Python, and other languages. GFAC-wrapped services can be run as stand-alone tools, or they can be registered with XRegistry for later incorporation into workflows. GFAC supports both persistent and dynamically created services.

Gateways adopting GFAC can outsource reliability and scalability issues and devote more resources to their domain-specific problems. The OGCE gateway suite has improved fault tolerance of computational jobs and data movement thanks to extensive efforts by the *Linked Environments for Atmospheric Discovery* (LEAD) Science Gateway [82]. The OGCE team has also leveraged large-scale coordinated gateway debugging efforts and applied the improvements to the advanced support requested gateways. Generic reliability and fault tolerance will not cover all problems: Codes will fail to run for reasons outside the gateway developer's control, so error detection, logging, and

resolution strategies must be specialized to the application. Providing extensibility for gateways to address their application-specific error conditions is an upcoming effort for GFAC development.

### 5.3.3.3 Gadget Container

Workflows, registries, and service wrappers have both client- and server-side pieces. The OGCE builds most of its default user interfaces as gadgets. Gadgets are client-side web components (Tier 4 of [Figure 5.8](#)) that rely on HTML, CSS, and JavaScript rather than on specific server-side development frameworks. Gadgets for science gateways need to communicate with server-side components (Tier 3 of [Figure 5.8](#)). This can be done using REST services. Greater interactivity, simpler development, and the freedom to use a wide range of server-side tools make gadgets an interesting component model for science gateways.

The OGCE Gadget Container is an Open Social-compatible tool for aggregating web gadgets. Gadgets are mostly self-contained web applications that comply with Google gadget or Open Social standards. The gadget container provides an operating context for a particular gadget, which may reside on a completely separate web server. The OGCE Gadget Container provides layout and skin management and user-level customization. The container supports an OpenID authentication option for registered users and OAuth authorization for gadgets. The container runs under HTTPS security and also supports secure (HTTPS) connections between the container and gadgets.

The gadget container is built on top of Apache Shindig, the reference implementation for the Open Social standard. This allows the container to create Open Social-compatible social networks. The container also supports Google FriendConnect, which provides a simplified programming interface for social networks.

### 5.3.3.4 Packaging

OGCE software is open source and available for download through SourceForge, and plans are underway to start an Apache Incubator project for GFAC, XBay, and supporting components. The preferred download mechanism is by SVN client checkouts and updates of tagged releases. The current OGCE release bridges several component projects. Each subproject can be built separately using Apache Maven; a master Maven POM is used to build all subprojects. This approach simplifies both development and deployment. Subprojects can be added when they mature, replaced with major upgrades, and discarded. Updates can be applied to specific components without rebuilding the entire software stack or developing a specific patching system. The OGCE software stack is designed to be portable and to compile on many platforms.

Building a science gateway requires to match end-user requirements, that is, the scientific use cases to be supported, to the capabilities of the gateway stack. Furthermore, developing the underlying workflows of the gateway is itself a lengthy scientific task, apart from the implementation issues. There are always components that can be used as is (security credential management and file browsers, for example), but OGCE software is

intended to be extended and modified by gateway developers.

This requires a close collaboration between science domain experts and cyberinfrastructure specialists. Long-term sustainability is an important challenge facing all gateways; particularly those that depend on external resource providers such as the TeraGrid and Open Science Grid (see [Figure 5.8](#) Tiers 1 and 2). These resources and their middleware evolve; gateways with Tier 3 and 4 components that are not actively maintained will decay. The challenge for many gateways is to maintain their middleware with reduced funding as the gateway matures from active development to stable usage.

## 5.4 Discovery, Registries, Metadata, and Databases

Distributed applications need to discover resources that suit their needs and manage them. In SOA, business services need to discover appropriate services to use and to integrate with. Registries are sophisticated naming and directory services that facilitate service resource discovery both at design and dynamically at runtime by classifying and categorizing services or metadata information about services. A registry requires a set of data structure specifications for the metadata to be stored in the registry, and a set of operations such as *Create, Read, Update, and Delete* (CRUD) for storing, deleting, and querying the data to store metadata for ownership, containment, and categorization of services. Registries usually contain three categories of information:

- **White pages** contain name and general contact information about an entity.
- **Yellow pages** contain classification information about the types and location of the services the entry offers.
- **Green pages** contain information about the details of how to invoke the offered services (technical data regarding the service).

Apart from registries, metadata, or information about data, can be used to facilitate discovery of resources and desired services. Metadata can be stored in relational or XML databases as metadata catalogs or added to web services to enhance service discovery capabilities. Integrating publish/subscribe patterns into databases can even add discovery capabilities to the static nature of databases, thus reducing the load on a single database caused by polling of enormous applications.

### 5.4.1 UDDI and Service Registries

UDDI specifications [83] define a way to describe, publish, and discover information about web services by creating a platform-independent, open framework. UDDI provides a name service and a directory service for looking up service descriptions by name or by a specific attribute. It was initiated in September 2000 as a joint collaboration on B2B integration between IBM, Microsoft, and Ariba. UDDI version 3.0 has been published as OASIS specifications and it has become an OASIS standard for public service registries.

The UDDI specification is focused on the definition of a collection of services supporting the description and discovery of: Businesses, organizations, and other web services providers; the web services they make available; and the technical interfaces

which may be used to access those services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a web service-based software environment for both publicly available services and services only exposed internally within an organization.

There are two primary types of registries. A public registry is a logically centralized distributed service that replicates data with other public registries on a regular basis. A private registry is only accessible within a single organization or is shared by a group of business partners for a special purpose. The latter is also called a semiprivate or shared registry. The UDDI Business Registry consists of replicated registries (initially hosted by IBM and Microsoft) called UDDI operators.

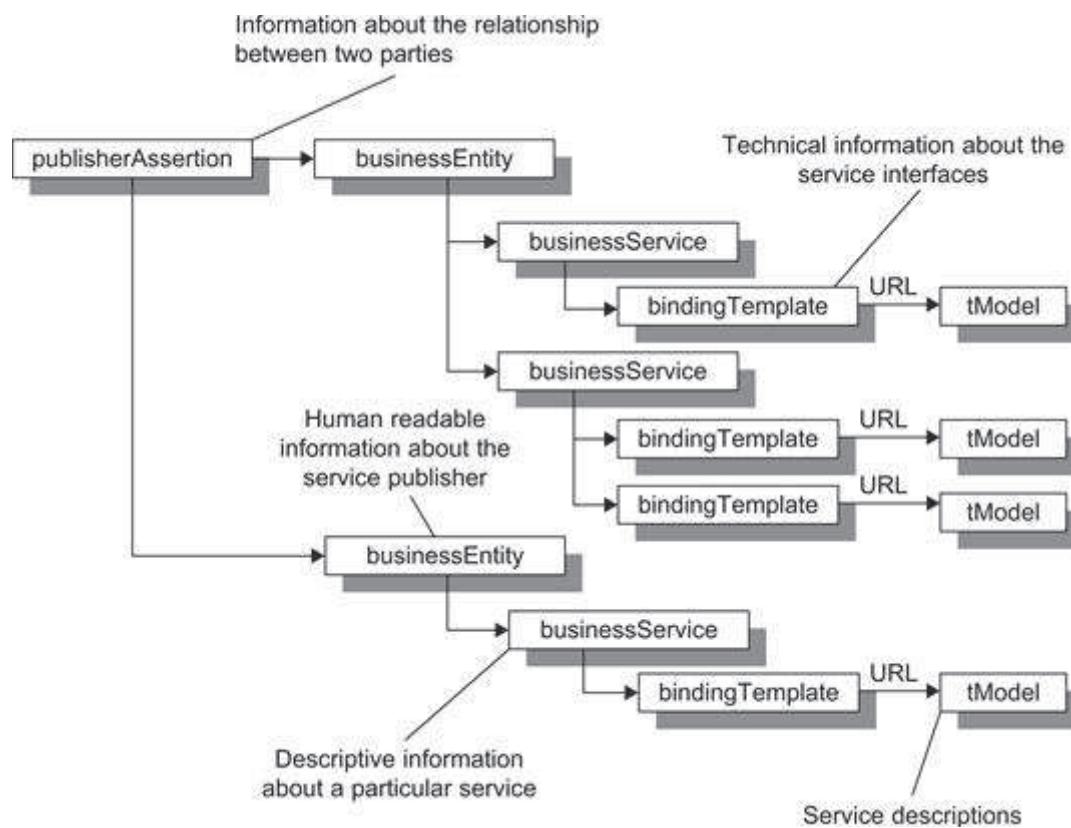
A UDDI registry is an instance of a web service, and its entries can be published and queried using a SOAP-based interface. UDDI defines data structures and APIs for programmatically publishing service descriptions and querying the registry. Data in a UDDI registry is organized as instance types:

- ***businessEntity*** Describes an organization or a business that provides the web services, including the company name, contact information, industry/product/geographic classification, and so on
- ***businessService*** Describes a collection of related instances of web services offered by an organization, such as the name of the service, a description, and so forth
- ***bindingTemplate*** Describes the technical information necessary to use a particular web service, such as the URL address to access the web service instance and references to its description
- ***tModel*** A generic container for specification of WSDL documents in general web services
- ***publisherAssertion*** Defines a relationship between two or more *businessEntity* elements
- ***subscription*** A standing request to keep track of changes to the entities in the subscription

The entities *businessEntity*, *businessService*, *bindingTemplate*, and *tModel* form the core data structures of UDDI, each of which can be uniquely identified and accessed by a URI, called the “UDDI key.” These entities and their relationships are depicted in [Figure 5.10](#). A UDDI registry can be used by service providers, service requestors, or other registries. For such interactions with the registry, UDDI provides a set of APIs. Two basic types of API operations are applied to UDDI components:

- **UDDI Inquiry API** In order to find the set of registry entries such as business, service, binding, or *tMode*, details matching a particular search criterion (*find\_*) or details of an entry corresponding to a given UDDI key (*get\_*) operation can be used.
- **UDDI Publishers API** This enables add, modify, and delete entries by providing *save\_* and *delete\_* operations. In addition to the aforementioned look-up APIs, UDDI also defines general-purpose operation types, such as the next 4 specialized APIs.
- **UDDI Security API** Allows users to get and discard authentication tokens (*getAuthToken*, *discardAuthToken*)

- **UDDI Custody and Ownership Transfer API** Enables registries to transfer the custody of information among themselves and to transfer ownership of these structures one another (*transfer\_entities*, *transfer\_custody*)
- **UDDI Subscription API** Enables monitoring of changes in a registry by subscribing to track new, modified, and deleted entries (*delete\_subscription*, *get\_subscriptionResults*, *get\_subscriptions*, *save\_subscriptions*)
- **UDDI Replication API** Supports replication of information between registries so that different registries can be kept synchronized



**FIGURE 5.10** UDDI entities and their relationship.

Although UDDI is an open standard, it has never gained much popularity among various enterprise and scientific communities, as there has been no global registry for registering enterprise, e-science, or grid services following the UDDI specification after the closure of the public nodes of the Universal Business Registry operated by IBM, Microsoft, and SAP in January 2006. However, several public registries have been launched for public use by various communities providing a categorized list of a variety of services and related APIs. One of them is [ProgrammableWeb.com](#) [84].

[ProgrammableWeb.com](#) is a registry of a variety of Web 2.0 applications, such as mashups and APIs organized by category, date, or popularity. It has similar goals to UDDI, but does not use the detailed UDDI specifications. Mashups are composite Web 2.0 applications which combine capabilities from existing web-based applications, typically RESTful web services. Mashups can be compared to workflows, as they both implement distributed programming at the service level. Content used in mashups is

typically sourced from a third party via a public interface or API. According to data released by the [ProgrammableWeb.com](#) register [84], most mashup and APIs are applied in mapping, search, travel, social, instant messaging, shopping, video streaming areas.

Other methods of sourcing content for mashups include web feeds (such as RSS) and JavaScript. web developers are able to programmatically search and retrieve APIs, mashups, member profiles, and other data from the [ProgrammableWeb.com](#) catalog, integrate on-demand registry and repository functionality into any service, and dynamically add new content as well as comment on existing entries, using the provided API, based on open standards including XML, RSS, OpenSearch, and *Atom Publishing Protocol* (APP) [85]. Among the most popular mashups frequently used on the web site are those provided for Google Maps, Flickr, Facebook, Twitter, and YouTube.

## 5.4.2 Databases and Publish-Subscribe

Publish-subscribe is a design pattern that enables asynchronous interaction among distributed applications and was discussed in [Section 5.2](#) for middleware. Many high-level applications regularly query the database in order to adapt their execution according to this information. Such periodic data polling is not only inefficient and unscalable, but also resource-demanding on both sides, especially when the interval between calls to the database is very small, or in cases when there is more than one consumer application, it may increase the amount of network traffic and CPU usage dramatically. The publish-subscribe mechanism, already largely adopted in the implementation of today's applications, solves this issue. In a publish-subscribe interaction, event subscribers register to particular event types and receive notifications from the event publishers when they generate such events.

There is a dynamic, many-to-many relationship between event publishers and event subscribers, as there can be any number of publishers/subscribers for any type of event which can vary at any time. Publish-subscription adds dynamicity to static the nature of databases. While the publish-subscribe pattern was first implemented in centralized client/server systems, current research focuses mainly on distributed versions. The key benefit of the distributed publish-subscribe mechanism is the natural decoupling of publishers and subscribers. Since the publishers are unconcerned with the potential consumers of their data, and the subscribers are unconcerned with the locations of the potential producers of interesting data, the client interface of the publish-subscribe system is simple and intuitive.

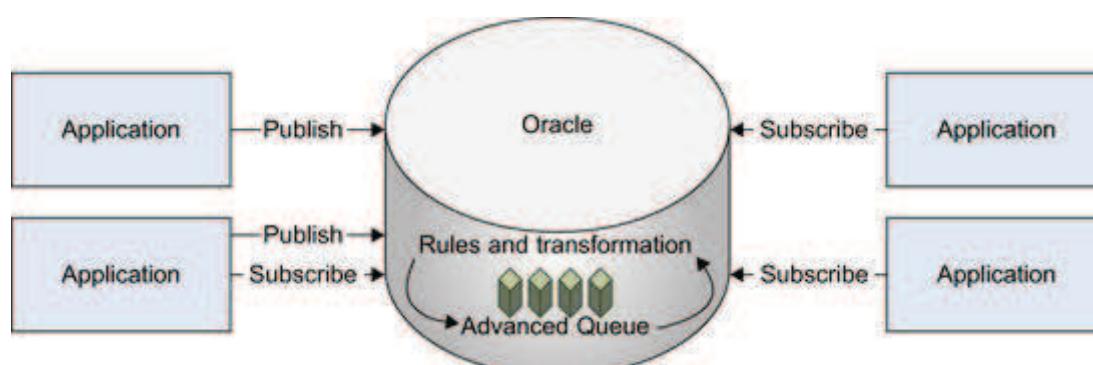
Publish-subscribe systems are classified as either topic-based or content-based. In topic-based systems, publishers generate events with respect to a topic or subject. Subscribers then specify their interest in a particular topic, and receive all events published on that topic. Defining events in terms of topic names only is inflexible and requires subscribers to filter events belonging to general topics. Content-based systems solve this problem by introducing a subscription scheme based on the contents of events. Content-based systems are preferable as they give users the ability to express their interest by specifying predicates over the values of a number of well-defined attributes.

The matching of publications (events) to subscriptions (interest) is done based on the content. Distributed solutions are mainly focused on topic-based publish-subscribe systems

Database systems provide many features that a messaging-based architecture can exploit, such as reliable storage, transactions, and triggers. On the other hand, integrated publish-subscribe capabilities in the database account for information-sharing systems that are simpler to deploy and maintain. However, since publish-subscribe and database technology have evolved independently, designing and implementing database-publish-subscribe-aware systems requires bringing together concepts and functionality from two separate worlds.

Jean Bacon, et al. [86] have extended the PostgreSQL open source database system to include publish-subscribe middleware functionality. It is based on the integration of active databases and the publish-subscribe communication model to form a global event-based system: Databases define and advertise change events, and clients subscribe to events of interest, and can refine their subscriptions through content-based filter expressions. This allows a database system in the local domain to function as an event broker (broker), reliably routing events among publishers, subscribers, and other brokers. This integration simplifies information management by grouping security, configuration (e.g., type schema), and recovery tasks for database and pub/sub operations under the same interface. Aktas has described use of publish-subscribe in grid information systems [87–89].

Message queuing and publish-subscribe are tightly intertwined: Oracle introduced a publish-subscribe solution for enterprise information delivery and messaging, shown in Figure 5.11, and based on Advanced Queuing—an implementation based on JMS and fully integrated in the Oracle database—to publish data changes and to automate business process workflows for distributed applications. Ontologies and other semantic mechanisms can be used to make events in the system more “context-aware.” Advanced Queuing was introduced in Oracle 8 and extended in Oracle 9i to support publish-subscription. In Oracle release 10.1, Advanced Queuing was integrated into Oracle Streams, called Oracle Streams Advanced Queuing.



**FIGURE 5.11** Oracle publish-subscribe model.

A combination of features are introduced to allow a publish-subscribe style of

messaging between applications. These features include rule-based subscribers, message propagation, the listen feature, and notification capabilities. Oracle Streams Advanced Queuing is built on top of Oracle Streams and leverages the functionality of Oracle Database so that messages can be stored persistently, propagated between queues on different computers and databases, and transmitted using Oracle Net Services and HTTP(S). As Oracle Streams Advanced Queuing is implemented in database tables, all operational benefits of *high availability* (HA), scalability, and reliability are also applicable to queue data. Standard database features such as recovery, restart, and security are supported by Oracle Streams Advanced Queuing. Database development and management tools such as Oracle Enterprise Manager can be applied to monitor queues. Like other database tables, queue tables can be imported and exported.

### 5.4.3 Metadata Catalogs

Metadata catalogs play a vital role in distributed heterogeneous environments such as grids by providing users and applications the means to discover and locate the desired data and services among lots of sites on such environments. Metadata is information about data. Metadata is important, since it adds context to the data, in order to identify, locate, and interpret it. Key metadata on the grid includes the name and location of the data resource, structure of the data held within the data resource, data item names and descriptions, and user information (name, address, and profiles and preferences), or basic listings and simple lookup of available services, relating function and location without significant rich context. Metadata catalogs are used by various groups and communities, ranging from high-energy physics to biomedical, earth observation, and geological science.

Because of the importance of metadata services for the use of large-scale local or wide area storage resources, many groups have made efforts to investigate and implement such services. Among the earliest metadata catalogs is the *Metadata Catalog Service* (MCAT) [90], which is a part of the *Storage Resource Broker* (SRB) [91], which has evolved to the iRODS system [50]. MCAT, developed by the San Diego Supercomputing Center, aims to provide an abstraction layer over heterogeneous storage devices and file systems either inside or across computing centers. MCAT stores the data hierarchically using a tree of collections and is both a file and metadata catalog. Later versions of MCAT support replication and federation of data resources.

The MCS developed by the Globus Alliance [92] provides hierarchical organization of metadata and flexible schemas, and hides the storage backend from the user. The Globus project also contains the *Replica Location Service* (RLS) [93], which uses index servers to provide a global list of files available on different replica catalogs. Several LHC experiments have implemented their own specific metadata catalogs using a standard relational database backend and providing an intermediate layer to access the catalog on a distributed environment.

AMGA (the ARDA *Metadata for Grid Applications*) [94] is the official metadata catalog of the gLite software stack of the EGEE project. It began as an exploratory project to study

the metadata requirements of the LHC experiments, and has since been deployed by several groups from different user communities, including high-energy physics (for LHCb bookkeeping), biomedicine, and earth observation. AMGA uses a hierarchical file system-like model to structure metadata, stored in a relational database. It stores entries representing the entities that are being described, such as files. The entries are grouped into collections, which can have a variant number of user-defined attributes, called the schema of the collection.

Attributes are represented as key-value pairs with type information, and each entry assigns an individual value to the attributes of its collection. A schema can be a representation of a directory, which can contain either entries or other schemas. As an advantage of this tree-like structure, users can define a hierarchical structure which can help to better organize metadata in subtrees that can be queried independently. The server supports several storage systems by using modules. AMGA can manage groups of users with different permissions on directories. In grid environment, file and metadata catalogs are used by users for discovering and locating data among the hundreds of grid sites.

#### 5.4.4 Semantic Web and Grid

The grid strives to share and access metadata in order to automate information discovery and integration of services and resources in a dynamic, large-scale distributed environment. Meanwhile, the Semantic web is all about automation discovery and integration: adding machine-processable semantics to data so that computers can understand such information and process it on behalf of end users, thus enabling more intelligent web searching and linkage based on attaching rich metadata to web pages. The Semantic web aims to provide an environment where software agents are able to dynamically discover, interrogate, and interoperate resources and perform sophisticated tasks on behalf of humans, which is not far from the ambition of grid computing.

To achieve this, work has been undertaken to assert the meaning of web resources in a common data model, the *Resource Description Framework (RDF)*, using agreed ontologies expressed in a common language, such as the OWL web ontology language, so we can share the metadata and add in background knowledge. From this basis we should be able to query, filter, integrate, and aggregate the metadata, and reason over it to infer more metadata by applying rules and policies.

RDF is the first language developed for the Semantic web, using XML to represent information (including metadata) about resources on the web. RDF uses web identifiers (URIs), and describes resources in terms of simple properties and property values. OWL is an expressive ontology language that extends RDF schema. OWL adds more vocabulary for describing properties and classes: among others, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes.

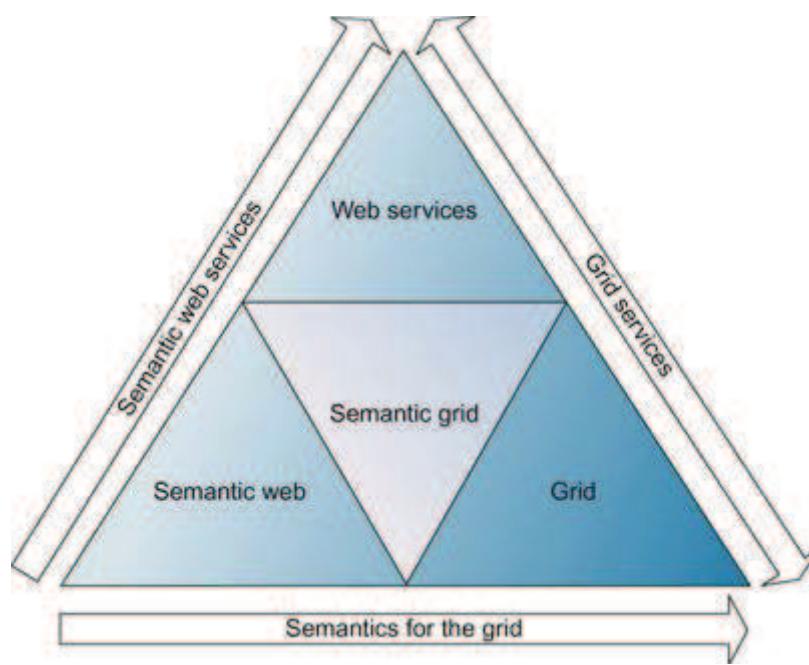
Semantic web services describe and annotate various aspects of a web service using explicit, machine-understandable semantics, facilitating the discovery, execution,

monitoring, and aggregation of resources and web services, which solves interoperability issues and helps bring resources together to create virtual organizations. The OWL-S ontology enables web services to be described semantically and their descriptions to be processed and understood by software agents. It provides a standard vocabulary that can be used together with the other aspects of the OWL description language to create service descriptions. The OWL-S ontology defines the top-level concept of a “Service” and three OWL-S subontologies:

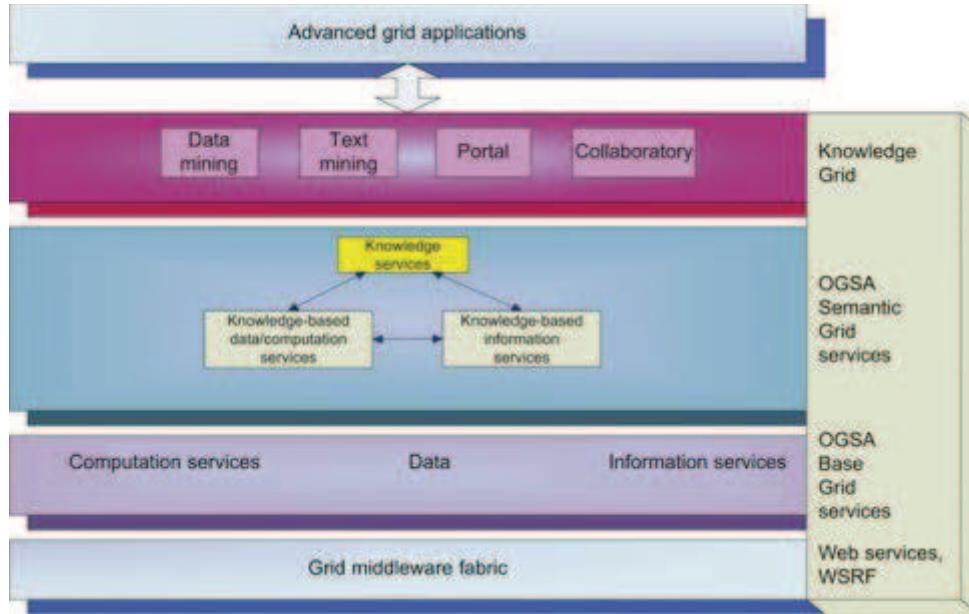
- **Service profile** Expresses what a service does in order to enable service advertisement and discovery.
- **Service model** Describes how the service works in order to enable service invocation, composition, monitoring, and recovery.
- **Service grounding** Specifies the details of how to access the service. A grounding can be thought of as a mapping from an abstract to a concrete specification, based on WSDL as a particular specification language.

The “semantic grid” or “grid with semantics” aims to incorporate the advantages of the grid, Semantic web, and web services. Built on the W3C Semantic Web Initiative, it is an extension of the current grid in which information and services are given well-defined meanings (ontologies, annotations, and negotiation processes as studied in the Semantic web and Software Agent paradigms), better enabling computers and people to work in cooperation. The semantic grid provides a general semantic-based, computational, and knowledge-based service for enabling the management and sharing of complex resources and reasoning mechanisms, thus systematically exposing semantically rich information associated with resources to build more intelligent grid services.

The notion of the semantic grid shown in [Figures 5.12](#) and [5.13](#) was first articulated in the context of e-science, which is all about scientific investigation among scientists of various communities such as physicists, biologist, and chemists and their resources performed through distributed global collaborations such as the grid to solve scientific problems by generating, analyzing, sharing, and discussing their insights, experiments, and results in an effective manner, and the computing infrastructure that enables this joint effort. Higher-level services use the information relating to the resources’ capability and the mechanisms for service fulfillment to automatically discover interoperable services and select the most appropriate service for the user with minimal human intervention.



**FIGURE 5.12** Semantic grid-related concepts and technologies. Courtesy of Goble and Roure, (ECAI-2004), Valencia, Spain, [95]



**FIGURE 5.13** Semantic grid architecture. Courtesy of Goble and Roure, (ECAI-2004), Valencia, Spain, [95]

Semantic OGSA (S-OGSA) [96] has been proposed as reference architecture for semantic-based grids. S-OGSA extends OGSA to support the explicit handling of semantics, and defines the associated knowledge services to support a set of service capabilities. This is done by introducing semantic provisioning services that give support to the provision of semantics, by allowing the creation, storage, update, removal, and access of different forms of knowledge and metadata. This set of services includes ontology management and reasoning services, metadata services, and annotation services.

S-OGSA has three main aspects: the model (the elements which it is composed of and their interrelationships), the capabilities (the services needed to deal with such

components), and the mechanisms (the elements that will enable delivery when deploying the architecture in an application, grounded to a grid platform). The myGrid [97] Project community as a pioneer of semantic grid efforts focusing on the challenges of e-science has developed a suite of tools and services to enable workflow-based composition of diverse biological data and computational resources. Within the project, Semantic web technologies have been applied to the problems of resource discovery and workflow results management in a grid environment.

## 5.4.5 Job Execution Environments and Monitoring

A distributed job execution environment often consists of two components: a job execution engine and a distributed data management system. The job execution engine mainly deals with job scheduling, resource allocation, and other issues such as fault tolerance. The data management system often provides an abstraction for jobs to access distributed data. In recent years, to tackle the increasing need for Internet scale information management and processing, many Internet service companies built their own distributed systems for their specific needs. Most of these systems provide distributed execution engines and support integrated applications. Google MapReduce [98] and Microsoft Dryad [99] are two examples of this type of system described in Chapter 6.

MapReduce was primarily designed to support Google applications that use and generate large data sets. It generalizes a map/reduce abstraction from these applications and provides a simple programming model for distributed execution of programs. Communications between programs are generalized as exchanges of key-value pairs. The storage of these key-value pairs is supported by Google File System (GFS) [100]. Parallelism is achieved through parallel scheduling of these distributable programs. Dryad has similar scope with MapReduce, but application dependencies can be arbitrarily constructed by explicitly specifying a DAG (directed acyclic graph). It supports data flowing along the links from one job's output to another job's input in a manner similar to a UNIX pipe.

The scheduling mechanism in MapReduce considers data locality by using the data location information provided by the GFS metadata server. Similar scheduling strategy is exploited in location-aware request distribution algorithms (LARD) for web server clusters [100], which schedule requests based on data locations as well as active connections to the servers that host the data. LARD allows dynamic creation of replicas.

Matchmaking built on the publish-subscribe model is a common approach for scheduling computationally intensive jobs. Grid computing middleware such as Condor [101] uses such a mechanism to distribute jobs. Condor matchmaking allows an agent that handles clients' job requests and resources to publish the job requirements and resource description in a form of semistructured data. The agents and resources subscribe to a matchmaker, which scans the published data and pairs jobs with resources according to the preferences and constraints specified in the data. Once a pair is formed, the matchmaker notifies the matching agent and resource. The agent and resource then

establish contact and execute a job.

Dryad has a similar scheduling mechanism. Each execution of a vertex in a DAG has an execution record that contains the state of the execution and the versions of the predecessor vertices providing its input data. A vertex is placed into a scheduling queue when its input data is ready. The constraints and preferences of running the vertex are attached to its execution record (e.g., the vertex may have a list of computers on which it would like to run, or prefers to be located in a computer where a data set is stored). The scheduler then does the matchmaking by allocating the vertex to the resource. The approach has a long history and can be traced back to the Linda programming model [102] and middleware inspired by Linda, such as JavaSpaces [103] where jobs are published in a shared space generalizing the messaging queue and consumed by resources subscribed to the space.

As jobs can be dispatched to different nodes, a job execution environment typically needs the support of a distributed data management system for a job to access remote data sets and, sometimes, exchange data with other jobs. As mentioned earlier, MapReduce is supported by GFS. Dryad also has a GFS-like distributed storage system, which can break large data files into small chunks. These chunks are then distributed and replicated across the disks of nodes in the system. The communication channels between jobs are often file-based. Through a distributed storage system, a file can be passed from one job to another transparently. Accessing and exchanging data with certain structures can be supported through abstractions built on top of the distributed storage system. Google BigTable [104] and Amazon Dynamo [105] are two examples.

The data abstraction in BigTable described in Section 6.3 is a multidimensional sorted map and in Dynamo is key-value pair. With these abstractions, a job can access multidimensional data or key-value pairs from any node in the system. As a consequence, a job execution environment is very powerful nowadays. It is capable of running not only computational jobs, but also a variety of data-intensive jobs. However, as maintaining data consistency comes with a high cost in a distributed environment, such a job execution environment has limitations for running applications with strong consistency requirements. At the time of this writing, BigTable and Dynamo only support relaxed consistency.

In a large-scale distributed system where applications are dynamically placed onto any computers to run, the capability of collecting information regarding the application and resource status becomes crucial for the system to achieve efficiency, detect failures or risks, and track system states. A contemporary distributed system often has a complex monitoring subsystem. Astrolabe [106], used in Amazon, is such an example. Astrolabe monitors the state of a collection of distributed resources and is capable of continuously computing summaries of the data in the system through aggregation. Its aggregation mechanism is driven by SQL queries (e.g., the query `SELECT MIN(load) AS load` returns the minimal load in the system).

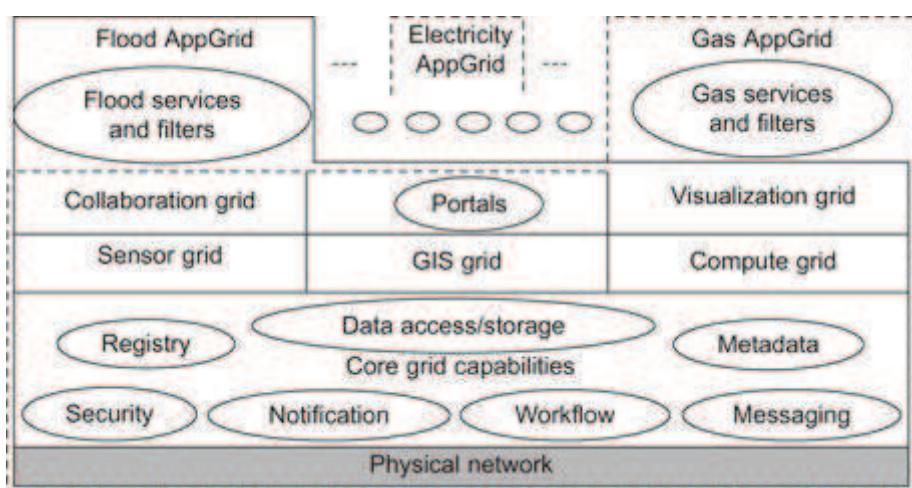
Aggregating such information dynamically from a large number of nodes is challenging. Traditional monitoring systems in clusters do not scale beyond a few dozen nodes [106]. Astrolabe adopts a decentralized approach to achieve scalability. It runs an

agent on each node. The agents talk with one another using a gossip protocol. Information managed by Astrolabe is organized hierarchically. Each agent maintains a data structure that contains a subset of the hierarchy. The local information is updated directly in an agent. An agent that needs the information of an internal node of the hierarchy can aggregate the information of the children in the hierarchy from agents that are responsible for them. Information of sibling nodes in the hierarchy can be obtained from relevant agents through the gossip protocol. The gossip protocol is simple: Periodically, each agent randomly selects some other agent and exchanges information with it. The information exchanged is the least common ancestor of the two agents.

Through this mechanism, Astrolabe is capable of taking snapshots of the distributed state. The state is aggregated dynamically without a central agent storing all the information. However, the aggregated information is replicated among all the agents involved in the process and it raises consistency issues. Two users retrieving the same data attribute may get different results. Astrolabe only supports a relaxed consistency called eventual consistency, that is, given an aggregate attribute  $X$  that depends on some other attribute  $Y$ , Astrolabe guarantees that an update made to  $Y$  is eventually reflected in  $X$ . Such a monitoring system is important to a job execution environment, particularly when the underlying system scales.

## 5.5 Workflow in Service-Oriented Architectures

In [Section 5.1](#), we described services as the basic unit for constructing distributed systems. However a “real system” consists of multiple interacting (generalized) services as shown in [Figure 5.14](#) and introduced in [Section 5.1.2.1](#). In particular, both a simple sensor (perhaps only an output data stream) and a complete grid (a collection of services with multiple input and output message ports) were shown in [Figure 1.22](#) as example collections of services. Thus, the prototypical complete system could be a “grid of services,” but we also talk about a “grid of grids” or even a “grid of clouds.” In [Figure 5.14](#), we show multiple application grids in different areas of “critical infrastructure.” Component grids (subgrids) are invoked for collaboration, visualization, sensor fusion, computing, and GIS applications. This builds on core grid capabilities in areas covered in [Sections 5.2, 5.3, and 5.4](#). By adding application-specific services, one builds a distributed system to support the study of flood, gas, or electrical infrastructure. Workflow is used to integrate component grids and services. Note that this discussion is phrased in terms of grids, but is equally applicable to clouds. We will discuss grid computing in more detail in [Chapter 7](#).



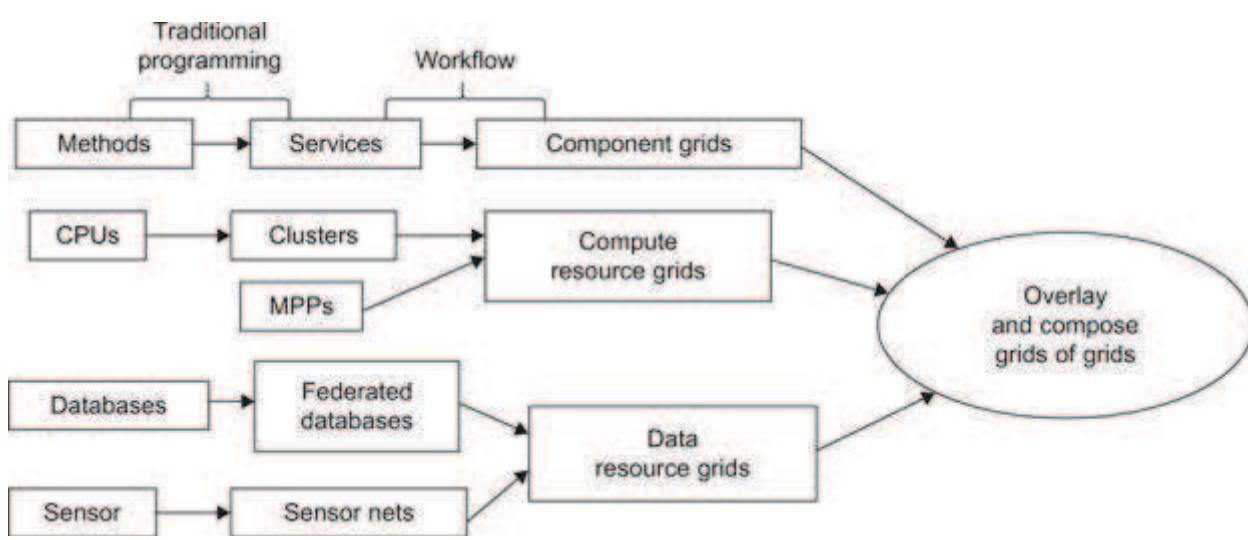
**FIGURE 5.14** The concept of the grid of grids of services.

## 5.5.1 Basic Workflow Concepts

In Section 5.2, we described approaches for managing the messages between services and component grids, but here we focus on workflow which is the approach to “programming the interaction between services.” As well as saying that workflow describes “programming the web or grid,” one can also use terms such as “software coordination,” “service orchestration,” “service or process coordination,” “service conversation,” “web or grid scripting,” “application integration,” or “software bus.”

It is an area of active research with different approaches emphasizing control flow, scheduling, and/or data flow. Some recent reviews of workflow are [15,107–110]. Note that workflow implies a two-level programming model for distributed systems. The basic services are programmed in traditional languages—C, C++, FORTRAN, Java, and Python—while workflow describes the coarser-grained programming of services interacting with one another. Each service is programmed in traditional languages while their interaction is described by workflow.

Note that the multilevel programming concept is familiar from elementary shell programming where pipes are often used to link executing programs. In fact, scripting (as in shell scripts) is one popular approach to workflow with distributed programming constructs replacing the familiar UNIX primitives. For example, TCP channels or publish-subscribe messaging replaces pipes. In Figure 5.15, we show that multilevel architectures are very common—not just in programming as in workflow, but also in computing, databases, and sensors.



**FIGURE 5.15** Hierarchical computing, data, and programming abstraction.

The concept of workflow was first developed by the Workflow Management Coalition [111] which has existed for almost 20 years and generated standard reference models, documents, and a substantial industry of tools and workflow management support products. However, this coalition is largely concerned with business process management, with steps in a workflow often involving human and not computer steps. For example, Allen defined business workflow as the automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules [112]. Thus, workflow in this chapter addresses very different issues from those described by the Workflow Management Coalition.

We can identify the development of workflow concepts and systems as a major achievement of the grid and distributed system community. Note that services are, of course, equally critical, but the essence of that idea came from business systems. Historically, workflow concepts evolved from distributed programming using systems such as Linda [113], HeNCE [114], AVS [115], and Khoros [116] as well as complex Shell (Perl) scripts. Petri Nets also can be considered one of the early ideas important for workflow. Although several good workflow systems have been built on Petri Nets [117–119], the most popular systems today are not based on Petri Nets. There are important commercial areas using environments very similar to scientific workflow. These include systems to analyze experimental (laboratory) data, termed Laboratory Information Systems or LIMS as exemplified by [120,121] or the general resource [122]. The chemical informatics and business intelligence areas have several workflow-like systems including InforSense [123] and Pipeline Pilot [124].

## 5.5.2 Workflow Standards

As with other web service-related concepts, there was substantial standards activity among OASIS, OMG, and W3C, often with overlapping goals, as summarized in Table 5.9. This work largely stemmed from the 2000 to 2005 when standards were viewed as essential to enable the web service dream that interoperability was achieved by complete

specification of service features. Recently, there has been a realization that this goal produced heavyweight architectures where the tooling could not keep up with the support of the many standards. Today we see greater emphasis on lightweight systems where interoperability is achieved by ad hoc transformations where necessary. Another problem of the standardization work was that it largely preceded the deployment of systems, and so one found premature standardization that missed key points. This background explains the many unfinished standard activities in [Table 5.9](#).

**Table 5.9**  
**Workflow Standards, Links, and Status**

Standard	Link	Status
<b>BPEL</b> Business Process Execution Language for Web Services (OASIS) V 2.0	<a href="http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html">http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html</a> ; <a href="http://en.wikipedia.org/wiki/BPEL">http://en.wikipedia.org/wiki/BPEL</a>	April 2007
<b>WS-CDL</b> Web Service Choreography Description Language (W3C)	<a href="http://www.w3.org/TR/ws-cdl-10/">http://www.w3.org/TR/ws-cdl-10/</a>	November 2005, not final
<b>WSCI</b> Web Service Choreography Interface V 1.0 (W3C)	<a href="http://www.w3.org/TR/wsci/">http://www.w3.org/TR/wsci/</a>	August 2002, note only
<b>WSCL</b> Web Services Conversation Language (W3C)	<a href="http://www.w3.org/TR/wscl10/">http://www.w3.org/TR/wscl10/</a>	March 2002, note only
<b>WSFL</b> Web Services Flow Language	<a href="http://www.ibm.com/developerworks/webservices/library/ws-wsfl2/">http://www.ibm.com/developerworks/webservices/library/ws-wsfl2/</a>	Replaced by BPEL
<b>XLANG</b> Web Services for Business Process Design (Microsoft)	<a href="http://xml.coverpages.org/_XLANG-C-200106.html">http://xml.coverpages.org/_XLANG-C-200106.html</a>	June 2001, replaced by BPEL
<b>WS-CAF</b> Web Services Composite Application Framework including <b>WS-CTX</b> , <b>WS-CF</b> , and <b>WS-TXM</b>	<a href="http://en.wikipedia.org/wiki/WS-CAF">http://en.wikipedia.org/wiki/WS-CAF</a>	Unfinished
<b>WS-CTX</b> Web Services Context (OASIS Web Services Composite Application Framework TC)	<a href="http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html">http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html</a>	April 2007
<b>WS-Coordination</b> Web Services Coordination (BEA, IBM, Microsoft at OASIS)	<a href="http://docs.oasis-open.org/ws-tx/wscoor/2006/06">http://docs.oasis-open.org/ws-tx/wscoor/2006/06</a>	February 2009
<b>WS-AtomicTransaction</b> Web Services Atomic Transaction (BEA, IBM, Microsoft at OASIS)	<a href="http://docs.oasis-open.org/ws-tx/wsat/2006/06">http://docs.oasis-open.org/ws-tx/wsat/2006/06</a>	February 2009
<b>WS-BusinessActivity</b> Framework (BEA, IBM, Microsoft at OASIS)	<a href="http://docs.oasis-open.org/ws-tx/wsba/2006/06">http://docs.oasis-open.org/ws-tx/wsba/2006/06</a>	February 2009
<b>BPMN</b> Business Process Modeling Notation (Object Management Group, OMG)	<a href="http://en.wikipedia.org/wiki/BPMN">http://en.wikipedia.org/wiki/BPMN</a> ; <a href="http://www.bpmn.org/">http://www.bpmn.org/</a>	Active
<b>BPSS</b> Business Process Specification Schema (OASIS)	<a href="http://www.ebxml.org/">http://www.ebxml.org/</a> ; <a href="http://www.ebxml.org/specs/ebBPSS.pdf">http://www.ebxml.org/specs/ebBPSS.pdf</a>	May 2001
<b>BTP</b> Business Transaction Protocol (OASIS)	<a href="http://www.oasis-open.org/committees/download.php/12449/business_transaction-btp-1.1-spec-cd-01.doc">http://www.oasis-open.org/committees/download.php/12449/business_transaction-btp-1.1-spec-cd-01.doc</a>	Unfinished

The successful activities have a business process flavor, and for scientific workflow, the most relevant standard is BPEL [125–128], which was based on the earlier proposals WSFL and XLANG. Note that XML is not well suited to specifying programming

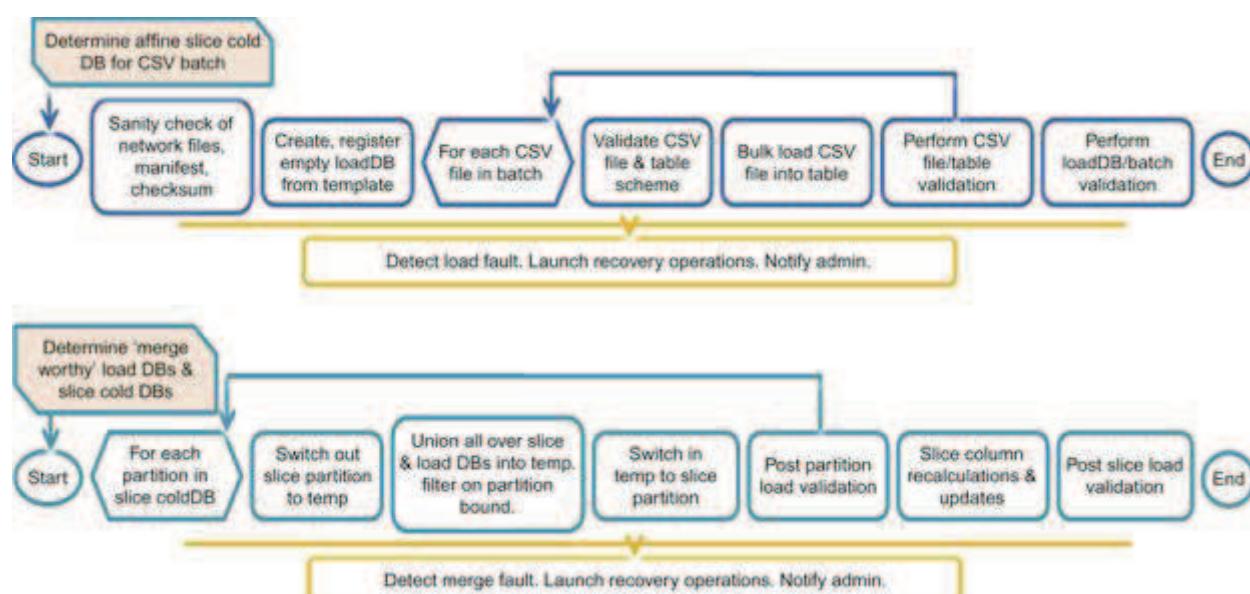
constructs; although XML can express data structures well, it is possible but not natural to express loops and conditionals that are essential to any language and the control of a workflow. It may turn out that expressing workflow in a modern scripting language is preferable to XML-based standards.

## 5.5.3 Workflow Architecture and Specification

Most workflow systems have two key components corresponding to the language and runtime components of any programming environment. We can call these components the workflow specification and workflow execution engine. They are linked by interfaces which could be specified by documents using, for example, the BPEL standard introduced earlier.

### 5.5.3.1 Workflow Specification

Scripting-based workflow systems can specify workflow in traditional language syntax similar to Python, JavaScript, or Perl. One can also directly specify the (XML) interface document driving the execution engine, although that is pretty low-level. However, most workflow systems use a graphical interface as illustrated in Figure 5.16. These capture scenarios illustrated by those of following examples.



**FIGURE 5.16** Two typical (Load and Merge) workflows from the Pan-STARRS astronomy data processing area.  
Courtesy of Barga, et al. [129]

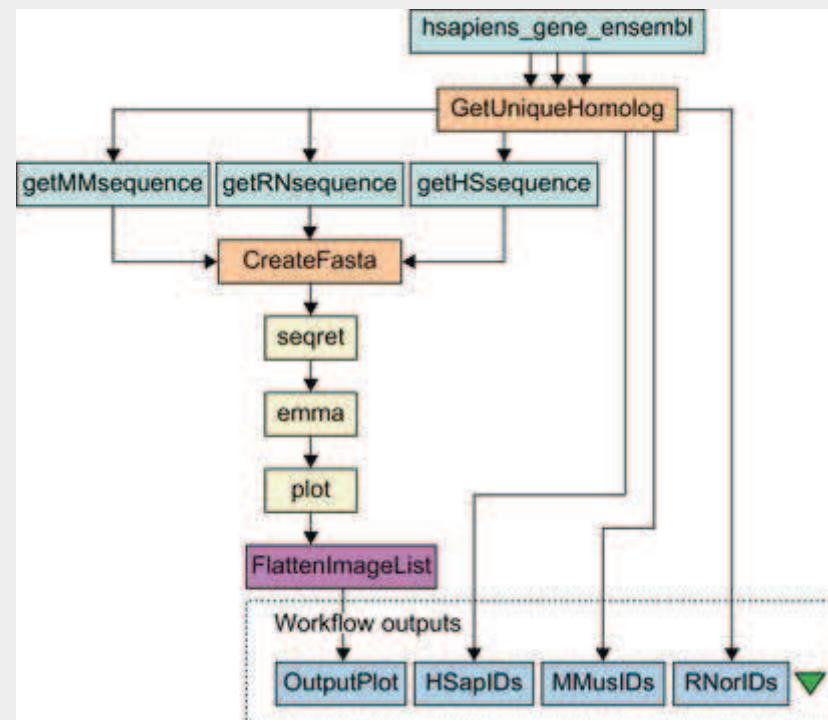
### Example 5.10

#### Pan-STARRS Workflow

Each step in Figure 5.16 is a “large” activity (i.e., a service) illustrating that

workflow is a programming system at a very different granularity from familiar languages such as C++. Note that scripting languages are often used for specifying coarse-grained operations, which “explains” why scripting is a popular approach to specifying workflows. However, a key feature of workflows involves a “few” steps, whereas a full program for a service is often thousands to perhaps millions of lines of code. This observation supports the use of a visual interface so that functional specifications are directly mapped into the programming model. Two visual interfaces are Microsoft’s Trident system [130,131] and Taverna, originally developed at the University of Manchester [18,132].

Figure 5.17 shows a typical workflow specified by the Taverna system [132]. There are menus specifying components (activities, services) that are chosen by the user and represented by “boxes” in the composition window. The user specifies the workflow logic through the linkage of the “boxes.” Each box has substantial metadata that can be specified and inspected by the user. The system allows specification of nontrivial control constructs such as looping over regions of workflow and conditional decisions. Note that each box can be a sequential or parallel component.



**FIGURE 5.17** The workflow in the Taverna system. Courtesy of C. Goble, 2008 [132]

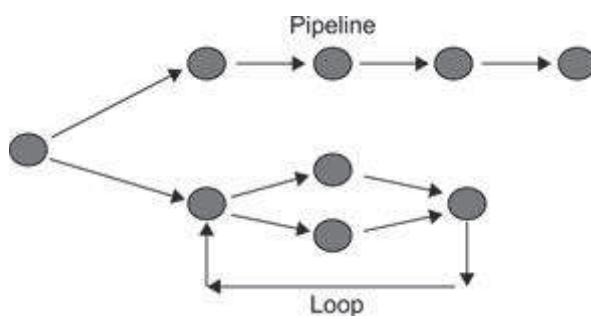
The need for the above operations is seen in the tornado tracking workflow [131] of LEAD II [133] which involves data processing services linked to traditional (MPI) parallel

weather simulations. The concurrency expressed in workflow is often termed “functional” parallelism distinguishing it from “data” parallelism. Typically functional parallelism (corresponding to different services in application) is absolute and is determined in size and nature by the problem. On the other hand, data parallelism is achieved by dividing a large data set into parts. The degree of data parallelism is determined by the number of available cores. Different parts correspond to using a parallel computing technology (MPI, threading, MapReduce) to implement the services.

## 5.5.4 Workflow Execution Engine

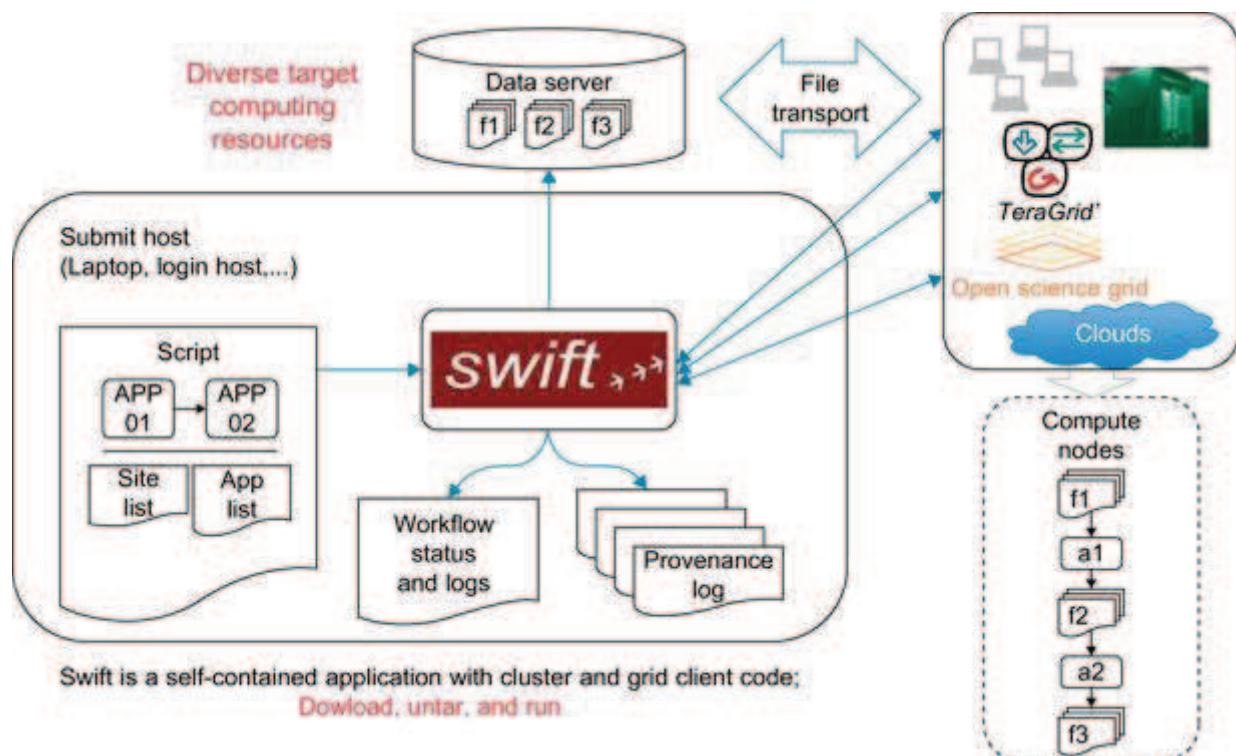
There are many different workflow systems. Workflow does not have strong performance constraints. We mentioned in the previous section that the typically large execution times of nodes made overhead less important than, say, for MPI. This same feature typically allows workflow to be executed in a distributed fashion—the network latency of long communication hops is often not important. The classification in [139] is of workflows used in domains from meteorology and ocean modeling, bioinformatics and biomedical workflows, astronomy and neutron science. These are examined according to their size, resource use, graph pattern, data pattern, and usage scenario. BPEL specifies the control and not the data flow of a workflow. Of course, the control structure implies the data flow structure for a given set of nodes.

Figure 5.18 shows a *pipeline* as seen for the astronomy workflows of Figure 5.16. A more general workflow structure is that of the DAG, which is a collection of vertices and directed edges, each edge connecting one vertex to another such that there are no cycles. That is, there is no way to start at some vertex  $V$  and follow a sequence of edges that eventually loops back to that vertex  $V$  again. In spite of sophisticated specialized workflow systems, scripting using traditional languages and toolkits is perhaps the dominant technique used to build workflows. Often this is done in an informal fashion using any environment with distributed computing (Internet) support; PHP may be the most popular environment for building mashups, but Python and JavaScript are also well used. Figure 5.18 shows a simple *cyclic graph*. Dagman [159] used in Condor [160] a sophisticated DAG processing engine. This leads to a class of workflow systems such as Pegasus [146] aimed at scheduling the nodes of DAG-based workflows. Karajan [161] and Ant [162] can also easily represent DAGs. Note that most sophisticated workflow systems support hierarchical specifications—namely the nodes of a workflow can be either services or collections of services (subworkflows). This is consistent with the Grid of Grids concept.



**FIGURE 5.18** A workflow graph that includes subgraphs illustrating pipelines and loops.

The issues that need to be addressed by a workflow engine are well understood in the active research community but are not well described in an integrated fashion [15,107–110]. Clearly, many and perhaps all of the execution issues discussed in this book for distributed systems, clouds, and grids are implicitly or explicitly relevant for workflow execution engines. Figure 5.19 specifies the interaction between the constituent services or activities. One important technology choice is the mechanism for transferring information between the nodes of the graph. The simplest choice is that each node reads from and writes to disk and this allows one to treat the execution of each node as an independent job invoked when all its needed input data is available on disk.



**FIGURE 5.19** Swift workflow system architecture. Courtesy of [www.ci.uchicago.edu/swift/](http://www.ci.uchicago.edu/swift/) [138]

This could seem inefficient, but we are not operating in a region for which MPI and parallel computing technologies are designed. There, low latency (microseconds) is often essential, but in a workflow, we have a different communication pattern: long-running jobs inputting and outputting substantial data sets. The cost of reading and writing is often quite acceptable and allows simpler fault tolerant implementations. Of course, one

can use the messaging systems described in [Section 5.2](#) to manage data transfer in a workflow, and in extreme cases, simple models where all communication is handled by a single central “control node.”

Obviously, this latter solution could lead to poor performance that does not properly scale as workflow size increases. Distributed object technology such as CORBA can also be used for communication, as it is in the Discover Middleware [\[136\]](#). In fact, there are often two communication systems in workflow environments corresponding to “control” and “data,” respectively. Obviously, the control communication would usually have small messages and very different requirements from the data network. In this regard, one should mention the “proxy model” which is often used in grid architectures and workflow.

Suppose one of the nodes in [Figure 5.18](#) corresponds to the execution of a large simulation job—say, the chemistry code Amber or Gaussian. Then one could consider the node as the Amber code with links corresponding directly to data used by this code. However, this is not normal. Rather, the node is a service that contains the metadata and mechanisms to invoke Amber on a (remote) machine and also to determine when it is complete. Then the information flowing between proxy nodes is all essentially control information. This proxy model can also be considered an agent framework [\[137\]](#).

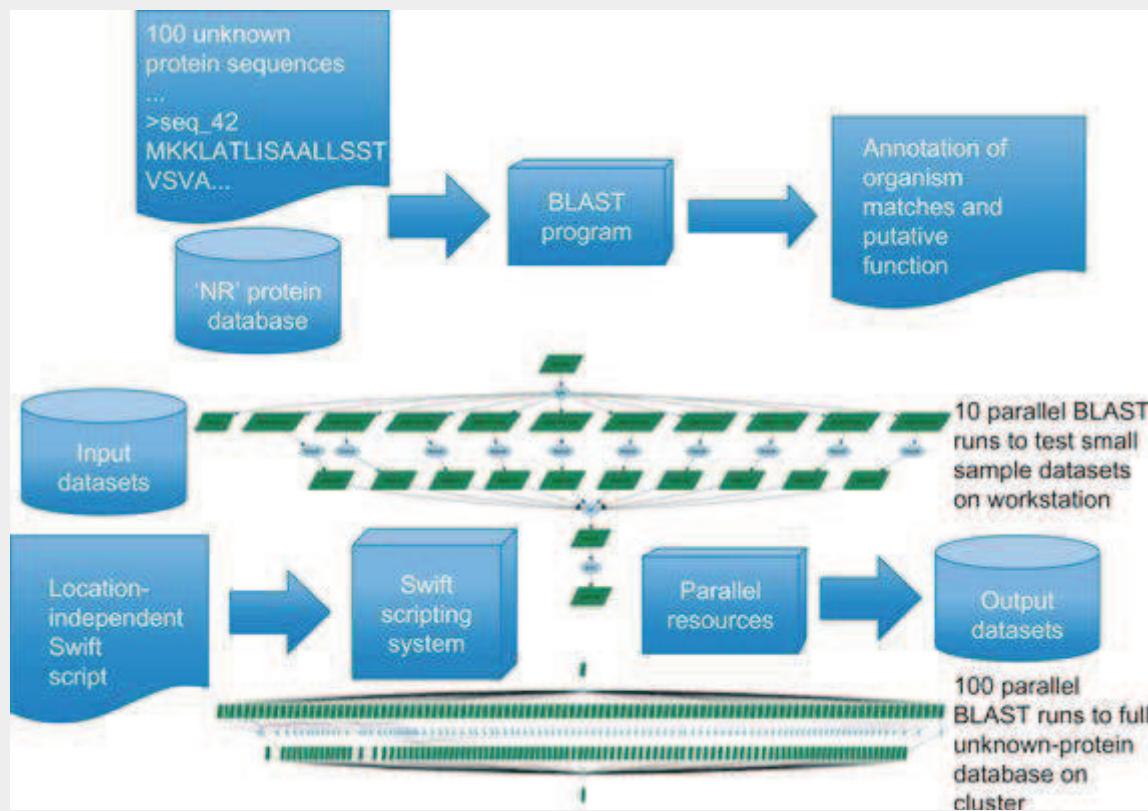
### 5.5.5 Scripting Workflow System Swift

Swift is a parallel scripting language in which application programs are represented as functions and variables can be mapped to files. Structure and array abstractions are used to manipulate sets of files in parallel. Swift has a functional data-flow-based execution model in which all statements are implicitly parallel. Parallel looping constructs explicitly specify large-scale parallel processing. [Figure 5.19](#) shows the architecture of the Swift workflow system.

#### Example 5.11

#### Bioinformatics Workflow in Swift

The Swift command, a Java application, runs on any user-accessible computer (e.g., a workstation or login server). It compiles and executes Swift scripts, coordinates remote data transfers, and executes applications on local and distributed parallel resources. The application of Swift to bioinformatics is illustrated in [Figure 5.20](#). This problem is perfect for MapReduce. Based on Swift’s implicit data flow model, the *foreach* loop starts automatically when the members of the array splitseqs become available.



**FIGURE 5.20** Application of the Swift to bioinformatics. Courtesy of Wikipedia, [http://en.wikipedia.org/wiki/Many-task\\_computing](http://en.wikipedia.org/wiki/Many-task_computing), [154]

The Swift system processes a set of protein sequences of unknown function through the BLAST search program, matching each sequence against proteins of known functions in the standard database "NR." Given, for example, 100 processors, the script can execute 100 invocations of BLAST in parallel to rapidly search the NR database. The purpose is to search for clues to the nature of the unknown proteins.

## 5.6 Bibliographic Notes and Homework Problems

We recommend the book [164] on key WS-\* web service technologies. The book [23] by Alonso provides detailed coverage of services and SOA, while Fox and Gannon review in detail services used in many grid applications [15,108]. [29] and [165] discuss issues differentiating web and grid services. The important RESTful approaches are described in [8] or the original thesis in [2]. Wikipedia is often a good source of information with strong coverage of messaging, as covered in Section 5.2.

Messaging is particularly well served with quality open source systems discussed in that section. There are several good collections of papers on portals and gateways [43,44]. The semantic web is well covered with [166]. The semantic grid is covered in [167]. iRods [50] is an important grid data system while the contribution of industry to this area is very important with many seminal papers from Google and Microsoft: for example, MapReduce [98], Dryad [99], the Google File System [100], and Fusion Tables [168].

Workflow is well covered in several reviews [15,107, 108,110].

Important workflow systems based on data flow technologies are the Kepler [140–142] and Triana [143–145] projects; Kepler is still actively being developed. Pegasus is an active system implementing the scheduling style of workflow [146]. Taverna [18,147] from the myGrid project [148] is very popular in the bioinformatics community and substantial effort has been made by the UK OMII effort [149] into making the system robust. An innovative extension of this project is the myExperiment scientific social networking site [150], which enables sharing of workflows. Other key aspects of workflow are security [163] and fault tolerance [110].

The HPSearch project [50,151,152] at Indiana University supported workflow efficiently from JavaScript focusing on control of the messaging system, NaradaBrokering [39], used to control streaming workflows (i.e., those where data is streamed between nodes of workflow and not written to disk). There are a few other workflow systems aimed at streaming data [153]. The new scripting environment, Swift [138] supports the Many Task programming model where innumerable jobs need to be executed [154].

There are very many approaches to workflow which are largely successful in prototype one-off situations. However, experience has found that most are not really robust enough for production use outside the development team. This observation motivated Microsoft to put its Trident workflow environment [129,130] into open source for science. Trident is built on the commercial quality Windows Workflow Foundation. Note that in most cases one would use Trident in a “proxy” fashion with workflow agents running on Windows controlling remote components that largely run on Linux machines [131,133]. Some workflow systems are built around the data flow concept with this being the original model [115,116,155] and the direct opposite of the proxy concept.

Today’s mashups are typically data flow systems with the interaction scripted in languages such as JavaScript or PHP. ProgrammableWeb.com has, for example, more than 2,000 APIs and 5,000 mashups recorded as of September 2010 [84]. Commercial mashup approaches include Yahoo! Pipes (<http://pipes.yahoo.com/pipes/>), Microsoft Popfly ([http://en.wikipedia.org/wiki/Microsoft\\_Popfly](http://en.wikipedia.org/wiki/Microsoft_Popfly)), and IBM’s sMash enhanced BPEL Web 2.0 [156] workflow environment (<http://yahoo.github.com/oozie/>) as it and workflow can execute similar problems. Other workflow approaches extend the “remote method invocation” model. This model underlies the *Common Component Architecture* (CCA) [157,158].

## Acknowledgements

This chapter was coauthored by Geoffrey Fox of Indiana University and Albert Zomaya of Sydney University, jointly. Their work was assisted by Ali Javadzadeh Boloori and Chen Wang of the Univ. of Sydney; by Shrideep Pallickara, Marlon Pierce, and Suresh Marru of Indiana Univ.; by Michael McLennan, George B. Adams III, and Gerhard Klimeck of Purdue Univ. and by Michael Wilde of the Univ. of Chicago. Kai Hwang of USC has edited the final manuscript of this chapter.

# References

1. Booth D, Haas H, McCabe F, et al. *Working Group Note 11: Web Services Architecture*. [www.w3.org/TR/2004/NOTE-ws-arch-20040211/](http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/); (accessed 18.10.10).
2. Fielding R. *Architectural Styles and the Design of Network-Based Software Architectures* University of California at Irvine 2000; p. 162  
<http://portal.acm.org/citation.cfm?id=932295>; 2000.
3. Hadley M. *Web Application Description Language (WADL)*, W3C Member Submission. [www.w3.org/Submission/wadl/](http://www.w3.org/Submission/wadl/); 2009; (accessed 18.10.10).
4. Restlet, the leading RESTful web framework for Java. [www.restlet.org/](http://www.restlet.org/); (accessed 18.10.10).
5. JSR 311 – JAX-RS: Java API for RESTful Web Services. <https://jsr311.dev.java.net/>; (accessed 18.10.10).
6. Jersey open source, production quality, JAX-RS (JSR 311) reference implementation for building RESTful web services. <https://jersey.dev.java.net/>; (accessed 18.10.10).
7. Winer D. *The XML-RPC specification*. [www.xmlrpc.com/](http://www.xmlrpc.com/); 1999; (accessed 18.10.10).
8. Richardson L, Ruby S. *RESTful Web Services* O'Reilly 2007.
9. Nadalin A, Kaler C, Monzillo R, Hallam-Baker P. *Web services security: SOAP message security 1.1 (WS-Security 2004)*, OASIS Standard Specification. <http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>; 2006; (accessed 18.10.10).
10. Andrieux A, Czajkowski K, Dan A, et al. *Web services agreement specification (WS-Agreement)*, OGF Documents, GFD.107. [www.ogf.org/documents/GFD.107.pdf](http://www.ogf.org/documents/GFD.107.pdf); 2007; (accessed 18.10.2010).
11. Davis D, Karmarkar A, Pilz G, Winkler S, Yalçinalp Ü. *Web services reliable messaging (WS-ReliableMessaging)*, OASIS Standard. <http://docs.oasis-open.org/ws-rx/wsrm/200702>; 2009; (accessed 18.10.2010).
12. Little M, Wilkinson A. *Web services atomic transaction (WS-AtomicTransaction)* Version 1.2, OASIS Standard. In: <http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os.pdf>; 2009.
13. Feingold M, Jeyaraman R. *Web services coordination (WS-Coordination)* Version 1.2, OASIS Standard. <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os.pdf>; 2009; (accessed 18.10.2010).
14. Chiu K, Govindaraju M, Bramley R. Investigating the limits of SOAP performance for scientific computing. In: *11th IEEE International Symposium on High Performance Distributed Computing*. 2002;246–254.
15. D. Gannon, G. Fox, Workflow in grid systems, Editorial of special issue of Concurrency & Computation: Practice & Experience, based on GGF10 Berlin meeting, Vol. 18, No. 10, 2006, pp. 1009–1019, doi:  
<http://dx.doi.org/10.1002/cpe.v18:10> and  
<http://grids.ucs.indiana.edu/ptliupages/publications/Workflow-overview.pdf>.
16. JBPM Flexible business process management (BPM) suite. [www.jboss.org/jbpm](http://www.jboss.org/jbpm); (accessed 18.10.10).

17. JBoss enterprise middleware. [www.jboss.org/](http://www.jboss.org/); (accessed 18.10.10).
18. Taverna workflow management system. [www.taverna.org.uk/](http://www.taverna.org.uk/); (accessed 18.10.10).
19. Englen RV, Gallivan K. The gSOAP toolkit for web services and peer-to-peer computing networks. In: *2nd IEEE/ACM International Symposium on Cluster Computing and The Grid (CCGRID '02)*. 2002.
20. Salz R. ZSI: The zolera soap infrastructure. <http://pywebsvcs.sourceforge.net/zsi.html>; 2005; (accessed 18.10.10).
21. Edwards J. *3-Tier server/client at work* first ed. John Wiley & Sons 1999.
22. Sun B. A multi-tier architecture for building RESTful web services, IBM Developer Works 2009. [www.ibm.com/developerworks/web/library/wa-aj-multitier/index.html](http://www.ibm.com/developerworks/web/library/wa-aj-multitier/index.html); 2009; (accessed 18.10.2010).
23. Alonso G, Casati F, Kuno H, Machiraju V. *Web Services: Concepts, Architectures and Applications (Data-Centric Systems and Applications)* Springer Verlag 2010.
24. Foster I, Kishimoto H, Savva A, et al. The open grid services architecture version 1.5, Open Grid Forum, GFD.80. In: [www.ogf.org/documents/GFD.80.pdf](http://www.ogf.org/documents/GFD.80.pdf); 2006.
25. Foster I, Tuecke S, Kesselman C. The philosophy of the grid. In: *1st International Symposium on Cluster Computing and the Grid (CCGRID0)*. IEEE Computer Society 2001.
26. Tuecke S, Czajkowski K, Foster I, et al. *Grid Services Infrastructure (OGSI) Version 1.0 Global Grid Forum Proposed Recommendation, GFD15*. [www.ggf.org/documents/GFD.15.pdf](http://www.ggf.org/documents/GFD.15.pdf); 2003; (accessed 18.10.10).
27. Graham S, Karmarkar A, Mischkinsky J, Robinson I, Sedukhin I. *Web Services Resource 1.2 (WS-Resource) WSRF, OASIS Standard*. [http://docs.oasis-open.org/wsrf/wsrf-ws\\_resource-1.2-spec-os.pdf](http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf); 2006; (accessed 18.10.10).
28. M. Gudgin, M. Hadley, T. Rogers, Web Services Addressing 1.0 – Core, W3C Recommendation, 9 May 2006.
29. Fox G, Gannon D. *A Survey of the Role and Use of Web Services and Service Oriented Architectures in Scientific/Technical Grids*. <http://grids.ucs.indiana.edu/ptliupages/publications/ReviewofServicesandWorkflow-IU-Aug2006B.pdf>; 2006; (accessed 16.10.10).
30. Supported version of Mule ESB. In: [www.mulesoft.com/](http://www.mulesoft.com/).
31. Open source version of Mule ESB. In: [www.mulesoft.org/](http://www.mulesoft.org/).
32. IBM's original network software MQSeries rebranded WebSphereMQ in 2002. In: [http://en.wikipedia.org/wiki/IBM\\_WebSphere\\_MQ](http://en.wikipedia.org/wiki/IBM_WebSphere_MQ).
33. WebSphereMQ IBM network software. In: [http://en.wikipedia.org/wiki/IBM\\_WebSphere\\_MQ](http://en.wikipedia.org/wiki/IBM_WebSphere_MQ).
34. Shen H. Content-based publish/subscribe systems. In: Shen X, ed. *Handbook of Peer-to-Peer Networking*. Springer Science+Business Media, LLC 2010;1333–1366.
35. Java Message Service JMS API. [www.oracle.com/technetwork/java/index-jsp-142945.html](http://www.oracle.com/technetwork/java/index-jsp-142945.html); and [http://en.wikipedia.org/wiki/Java\\_Message\\_Service](http://en.wikipedia.org/wiki/Java_Message_Service).
36. AQMP Open standard for messaging middleware. In: [www.amqp.org/confluence/display/AMQP/Advanced+Message+Queuing+Protocol](http://www.amqp.org/confluence/display/AMQP/Advanced+Message+Queuing+Protocol).
37. Mule MQ open source enterprise-class Java Message Service (JMS)

- implementation. In: [www.mulesoft.org/documentation/display/MQ/Home](http://www.mulesoft.org/documentation/display/MQ/Home).
38. Apache ActiveMQ open source messaging system. In: <http://activemq.apache.org/>.
39. NaradaBrokering open source content distribution infrastructure. In: [www.naradabrokering.org/](http://www.naradabrokering.org/).
40. RabbitMQ open source Enterprise Messaging System. In: [www.rabbitmq.com/](http://www.rabbitmq.com/).
41. Amazon Simple Queue Service (Amazon SQS). In: <http://aws.amazon.com/sqs/>.
42. Microsoft Azure Queues. In: <http://msdn.microsoft.com/en-us/windowsazure/ff635854.aspx>.
43. Wilkins-Diehr N. Special issue: Science gateways – common community interfaces to grid resources. *Concurr Comput Pract Exper.* 2007;19(6):743–749.
44. Wilkins-Diehr N, Gannon D, Klimeck G, Oster S, Pamidighantam S. TeraGrid science gateways and their impact on science. *Computer.* 2008;41.
45. Liu Y, Wang S, Wilkins-Diehr N. SimpleGrid 2.0: A learning and development toolkit for building highly usable TeraGrid science gateways. In: *SC-GCE.* 2009.
46. Reed DA. Grids, the TeraGrid and beyond. *IEEE Comput.* 2003;36(1):62–68.
47. Pordes R, Petravick D, Kramer B, et al. The open science grid. *J Phys Conf Ser.* 2007;78 012057.
48. Foster I. Globus toolkit version 4: software for service-oriented systems. *J Comput Sci Technol.* 2006;21(4):513–520.
49. Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. *Concurr Pract Exper.* 2005;17(2–4):323–356.
50. IRODS: Data Grids, Digital Libraries, Persistent Archives, and Real-time Data Systems. <https://www.irods.org>; (accessed 29.08.10).
51. Basney J, Humphrey M, Welch V. The MyProxy online credential repository. *Softw Pract Exp.* 2005;35(9):801–816.
52. Welch JV, Basney J, Marcusiu D, Wilkins-Diehr N. A AAAA model to support science gateways with community accounts. *Concurr Comput Pract Exper.* 2006;893–904.
53. Liming L, et al. TeraGrid's integrated information service. In: *5th Grid Computing Environments Workshop.* ACM 2009.
54. J. Kim, P.V. Sudhakar, Computational Chemistry Grid, a Production Cyber-environment through Distributed Computing: Recent Enhancements and Application for DFT Calculation of Amide I Spectra of Amyloid-Fibril. PRAGMA13, Urbana, IL, 2007.
55. Dooley R, Milfeld K, Guiang C, Pamidighantam S, Allen G. From proposal to production: lessons learned developing the computational chemistry grid cyberinfrastructure. *J Grid Comput.* 2006;4(2):195–208.
56. GridChem-Related Scientific Publications. In: <https://www.gridchem.org/papers>.
57. Demeler B. UltraScan: A comprehensive data analysis software package for analytical ultracentrifugation experiments. In: *Analytical Ultracentrifugation: Techniques and Methods.* 2005;210–229.
58. UltraScan Gateway. In: <http://uslims.uthscsa.edu/>.
59. UltraScan publications database. In: [www.ultrascan.uthscsa.edu/search-refs.html](http://www.ultrascan.uthscsa.edu/search-refs.html).

60. Brookes EH, Boppana RV, Demeler B. Biology – Computing large sparse multivariate optimization problems with an application in biophysics. In: *ACM/IEEE Conference on Supercomputing (SC2006)*. Tampa FL. 2006.
61. Brookes EH, Demeler B. Parsimonious regularization using genetic algorithms applied to the analysis of analytical ultracentrifugation experiments. In: *9th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. London. 2007;361–368.
62. Klimeck G, McLennan M, Brophy SP, et al. nanoHUB.org: Advancing education and research in nanotechnology. *Comput Sci Eng*. 2008;10(5):17–23.
63. Strachan A, Klimeck G, Lundstrom MS. Cyber-enabled simulations in nanoscale science and engineering. *Comput Sci Eng*. 2010;12:12–17.
64. Abelson H. The creation of OpenCourseWare at MIT. *J Sci Educ Technol*. 2007;17(2):164–174.
65. Hacker TJ, Eigenmann R, Bagchi S, et al. *The NEEShub cyberinfrastructure for earthquake engineering, computing, in Science and Engineering*. 2011;13(4):67–78. doi 10.1109/MCSE.2011.70.
66. Richardson T, Stafford-Fraser Q, Wood KR, Hopper A. Virtual network computing. *IEEE Internet Comput*. 1998;2(1):33–38.
67. Smith PM, Hacker TJ, Song CX. Implementing an industrial-strength academic cyberinfrastructure at Purdue University. *IEEE International Parallel and Distributed Processing Symposium (IPDPS)* 2008.
68. Qiao W, McLennan M, Kennell R, Ebert D, Klimeck G. Hub-based simulation and graphics hardware accelerated visualization for nanotechnology applications. *IEEE Trans Vis Comput Graph*. 2006;12:1061–1068.
69. Haley BP, Klimeck G, Luisier M, et al. Computational nanoelectronics research and education at nanoHUB.org. *J Comput Electron*. 2009;8:124–131.
70. *OpenVZ web site*. <http://openvz.org>; (accessed 17.08.10).
71. Alameda J, Christie M, Fox G, et al. The open grid computing environments collaboration: portlets and services for science gateways. *Concurr Comput Pract Exper*. 2007;19(6):921–942.
72. *Open Grid Computing Environments web site*. [www.collab-ogce.org](http://www.collab-ogce.org); (accessed 18.10.10).
73. Guo Z, Singh R, Pierce ME. Building the PolarGrid portal using Web 2.0 and OpenSocial. In: SC-GCE. 2009.
74. Gunarathne T, Herath C, Chinthaka E, Marru S. Experience with adapting a WS-BPEL runtime for eScience workflows. In: *5th Grid Computing Environments Workshop*. ACM 2009.
75. Marru S, Perera S, Feller M, Martin S. Reliable and Scalable Job Submission: LEAD Science Gateways Testing and Experiences with WS GRAM on TeraGrid Resources. In: *TeraGrid Conference*. 2008.
76. Perera S, Marru S, Gunarathne T, Gannon D, Plale B. Application of management frameworks to manage workflow-based systems: A case study on a large scale e-

- science project. In: *IEEE International Conference on Web Services*. 2009.
77. Perera S, Marru S, Herath C. Workflow Infrastructure for Multi-scale Science Gateways. In: *TeraGrid Conference*. 2008.
78. T. Andrews, F. Curbera, H. Dholakia, et al., Business process execution language for web services, version 1.1, 2003.
79. Oinn T, Addis M, Ferris J, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 2004.
80. Deelman E, Blythe J, Gil Y, et al. Pegasus: Mapping scientific workflows onto the grid. In: *Grid Computing*. Springer 2004.
81. Apache ODE (Orchestration Director Engine) open source BPEL execution engine. In: <http://ode.apache.org/>.
82. Christie M, Marru S. The LEAD Portal: a TeraGrid gateway and application service architecture. *Concurr Comput Pract Exper*. 2007;19(6):767–781.
83. UDDI Version 3 Specification, OASIS Standard, OASIS UDDI Specifications TC – Committee Specifications. [www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3](http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3); 2005; (accessed 18.10.10).
84. Programmable Web site for contributed service APIs and mashups. [www.programmableweb.com/](http://www.programmableweb.com/); (accessed 18.10.10).
85. Gregorio J, de hOra B. RFC 5023 – The Atom Publishing Protocol. IETF Request for Comments 2007; <http://tools.ietf.org/html/rfc5023>; 2007; (accessed 18.10.10).
86. Vargas L, Bacon J. Integrating Databases with Publish/Subscribe. *25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '05)* 2005.
87. Aktas M. Thesis Information Federation in Grid Information Services Indiana University. In: <http://grids.ucs.indiana.edu/ptliupages/publications/MehmetAktasThesis.pdf>; 2007.
88. Aktas MS, Fox GC, Pierce M. A federated approach to information management in grids. *J Web Serv Res*. 2010;7(1):65–98 In: <http://grids.ucs.indiana.edu/ptliupages/publications/JWSR-PaperRevisedSubmission529-Proofread.pdf>; 2010.
89. Aktas MS, Pierce M. High-performance hybrid information service architecture. *Concurr Comput Pract Exper*. 2010;22(15):2095–2123.
90. Deelman E, Singh G, Atkinson MP, et al. Grid based metadata services. In: *16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*. Santorini, Greece. 2004.
91. Baru CK. The SDSC Storage Resource Broker. In: Press I, ed. *CASCON '98 Conference*. Toronto. 1998.
92. Singh G, Bharathi S, Chervenak A, et al. A Metadata Catalog Service for Data Intensive Applications. In: *2003 ACM/IEEE Conference on Supercomputing, Conference on High Performance Networking and Computing*. ACM Press 15–21 November 2003;17.
93. Chervenak L, et al. Performance and Scalability of a Replica Location Service. In: *13th IEEE International Symposium on High Performance Distributed Computing*

- (HPDC 13). Washington, DC: IEEE Computer Society; 2004;182–191.
94. Koblitz B, Santos N, Pose V. The AMGA metadata service. *J Grid Comput.* 2007;6(1):61–76.
95. Goble CA, De Roure D. The Semantic Grid: Myth busting and bridge building. In: *16th European Conference on Artificial Intelligence (ECAI-2004)*. Valencia, Spain. 2004.
96. Corcho O, et al. An Overview of S-OGSA: A Reference Semantic Grid Architecture. In: *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*. 2006;102–115.
97. MyGrid. In: [www.mygrid.org.uk/](http://www.mygrid.org.uk/).
98. Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. In: *Sixth Symposium on Operating Systems Design and Implementation*. 2004;137–150.
99. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In: *ACM SIGOPS Operating Systems Review*. Lisbon, Portugal: ACM Press; 2007.
100. Ghemawat S. The Google File System. In: *19th ACM Symposium on Operating System Principles*. 2003;20–43.
101. Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the Condor experience. *Concurr Comput Pract Exper.* 2005;17(2–4):323–356.
102. Gelernter D. Generative communication in Linda. In: *ACM Transactions on Programming Languages and Systems*. 1985;80–112.
103. Freeman E, Hupfer S, Arnold K. *JavaSpaces: Principles, Patterns, and Practice*. Addison-Wesley 1999.
104. F. Chang, et al., BigTable: A Distributed Storage System for Structured Data, in: OSDI 2006, Seattle, pp. 205–218.
105. G. De Candia, et al., Dynamo: Amazon's highly available key-value store, in: SOSP, Stevenson, WA, pp. 205–219.
106. Van Renesse R, Birman K, Vogels W. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. In: *ACM Transactions on Computer Systems*. 2003;164–206.
107. Yu J, Buyya R. A taxonomy of workflow management systems for grid computing. In: *Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory*. Australia: University of Melbourne; 2005.
108. Taylor IJ, Deelman E, Gannon DB, Shields M. *Workflows for e-Science: Scientific Workflows for Grids*. Springer 2006.
109. Zhao Z, Belloum A, Bubak M. Editorial: Special section on workflow systems and applications in e-Science. *Future Generation Comp Syst.* 2009;25(5):525–527 In: <http://dx.doi.org/10.1016/j.future.2008.10.011>; 2009.
110. Deelman E, Gannon D, Shields M, Taylor I. *Workflows and e-Science: an overview of workflow system features and capabilities*. *Future Generation Comp Syst.* 2009;25(5):528–540 doi: <http://dx.doi.org/10.1016/j.future.2008.06.012>; 2009.
111. Workflow Management Consortium. In: [www.wfmc.org/](http://www.wfmc.org/).
112. R. Allen, Workflow: An Introduction, Workflow Handbook. Workflow

- Management Coalition, 2001.
113. Carriero N, Gelernter D. Linda in context. *Commun ACM*. 1989;32(4):444–458.
114. Beguelin A, Dongarra J, Geist GA. HeNCE: A User's Guide, Version 2.0. In: [www.netlib.org/hence/hence-2.0-doc-html/hence-2.0-doc.html](http://www.netlib.org/hence/hence-2.0-doc-html/hence-2.0-doc.html).
115. Upson C, Faulhaber Jr T, Laidlaw DH, et al. The application visualization system: a computational environment for scientific visualization. *IEEE Comput Graph Appl*. 1989;30–42.
116. Rasure J, Kubica S. The Khoros application development environment. In: *Khoros Research Inc*. New Mexico: Albuquerque; 1992.
117. Hoheisel A. User tools and languages for graph-based Grid workflows. *Concurr Comput Pract Exper*. 2006;18(10):1101–1113 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
118. Guan Z, Hernandez F, Bangalore P, et al. Grid-Flow: A Grid-enabled scientific workflow system with a Petri-net-based interface. *Concurr Comput Pract Exper*. 2006;18(10):1115–1140 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
119. Kosiedowski M, Kurowski K, Mazurek C, Nabrzyski J, Pukacki J. Workflow applications in GridLab and PROGRESS projects. *Concurr Comput Pract Exper*. 2006;18(10):1141–1154 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
120. LabVIEW Laboratory Virtual Instrumentation Engineering Workbench. In: <http://en.wikipedia.org/wiki/LabVIEW>.
121. LabSoft LIMS laboratory information management system. In: [www.labsoftlims.com/](http://www.labsoftlims.com/).
122. LIMSource Internet LIMS resource. In: <http://limsource.com/home.html>.
123. InforSense Business Intelligence platform. In: [http://www.inforsense.com/products/core\\_technology/inforsense\\_platform/index.html](http://www.inforsense.com/products/core_technology/inforsense_platform/index.html).
124. Pipeline Pilot scientific informatics platform from Accelrys. In: <http://accelrys.com/products/pipeline-pilot/>.
125. OASIS Web Services Business Process Execution Language Version 2.0 BPEL. In: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.
126. Curbera F, Khalaf R, Nagy WA, Weerawarana S. Implementing BPEL4WS: The architecture of a BPEL4WS implementation. *Concurr Comput Pract Exper*. 2006;18(10):1219–1228 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
127. ActiveBPEL Open Source workflow engine. In: [www.activebpel.org/](http://www.activebpel.org/).
128. Leyman F. Choreography for the Grid: Towards fitting BPEL to the resource framework. *Concurr Comput Pract Exper*. 2006;18(10):1201–1217 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
129. Barga R, Guo D, Jackson J, Araujo N. Trident: a scientific workflow workbench. In: *Tutorial eScience Conference*. Indianapolis. 2008.
130. Microsoft. *Project Trident: A Scientific Workflow Workbench*. <http://tridentworkflow.codeplex.com/>; and <http://research.microsoft.com/en-us/collaboration/tools/trident.aspx>.
131. The forecast before the storm [iSGTW International Science Grid This Week]. In: [www.isgtw.org/?pid=1002719](http://www.isgtw.org/?pid=1002719); 2010.

132. Goble C. Curating services and workflows: the good, the bad and the ugly, a personal story in the small. In: *European Conference on Research and Advanced Technology for Digital Libraries*. 2008.
133. Linked Environments for Atmospheric Discovery II (LEAD II). In: <http://pti.iu.edu/d2i/leadII-home>.
134. XBaya workflow composition tool. In: [www.collab-ogce.org/ogce/index.php/XBaya](http://www.collab-ogce.org/ogce/index.php/XBaya).
135. XBaya integration with OGCE Open Grid Computing Environments Portal. In: [www.collab-ogce.org/ogce/index.php/XBaya](http://www.collab-ogce.org/ogce/index.php/XBaya).
136. Bhat V, Parashar M. Discover middleware substrate for integrating services on the grid. In: *Proceedings of the 10th International Conference on High Performance Computing (HiPC 2003)*. Lecture Notes in Computer Science. Hyderabad, India: Springer-Verlag; 2003.
137. Zhao Z, Belloum A, Laat CD, Adriaans P, Hertzberger B. Distributed execution of aggregated multidomain workflows using an agent framework. In: *IEEE Congress on Services (Services 2007)*. 2007.
138. Open source scripting workflow supporting the many task execution paradigm. In: [www.ci.uchicago.edu/swift/](http://www.ci.uchicago.edu/swift/).
139. Ramakrishnan L, Plale B. A multi-dimensional classification model for scientific workflow characteristics. In: *1st International Workshop on Workflow Approaches to New Data-Centric Science*. Indianapolis. 2010.
140. Kepler Open Source Scientific Workflow System. In: <http://kepler-project.org>.
141. Ludäscher B, Altintas I, Berkley C, et al. Scientific workflow management and the Kepler system. *Concurr Comput Pract Exper*. 2006;18(10):1039–1065 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
142. McPhillips T, Bowers S, Zinn D, Ludäscher B. Scientific workflow design for mere mortals. *Future Generation Comp Syst*. 2009;25(5):541–551 In: <http://dx.doi.org/10.1016/j.future.2008.06.013>; 2009.
143. Triana. *Triana Open Source Problem Solving Environment*. [www.trianacode.org/index.html](http://www.trianacode.org/index.html); (accessed 18.10.10).
144. Taylor I, Shields M, Wang I, Harrison A. In: Taylor I, ed. *Workflows for e-Science*. Springer 2007;320–339.
145. Churches D, Gombas G, Harrison A, et al. Programming scientific and distributed workflow with Triana services. *Concurr Comput Pract Exper*. 2006;18(10):1021–1037 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
146. Pegasus Workflow Management System. In: <http://pegasus.isi.edu/>.
147. Oinn T, Greenwood M, Addis M, et al. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurr Comput Pract Exper*. 2006;18(10):1067–1100 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
148. myGrid multi-institutional, multi-disciplinary research group focusing on the challenges of eScience. In: [www.mygrid.org.uk/](http://www.mygrid.org.uk/).
149. OMII UK Software Solutions for e-Research. In: [www.omii.ac.uk/index.jhtml](http://www.omii.ac.uk/index.jhtml).
150. Collaborative workflow social networking site. In: [www.myexperiment.org/](http://www.myexperiment.org/).
151. Gadgil H, Fox G, Pallickara S, Pierce M. Managing grid messaging middleware. In:

152. *HPSearch*, Scripting environment for managing streaming workflow and their messaging based communication. [www.hpssearch.org/](http://www.hpssearch.org/); 2005; (accessed 18.10.10).
153. Herath C, Plale B. Streamflow. In: *10th IEEE/ACM International Conference on Cluster*. Cloud and Grid Computing 2010.
154. Many Task Computing Paradigm. In: [http://en.wikipedia.org/wiki/Many-task\\_computing](http://en.wikipedia.org/wiki/Many-task_computing).
155. Bhatia D, Burzevski V, Camuseva M, et al. WebFlow: A visual programming paradigm for web/Java based coarse grain distributed computing. *Concurr Comput Pract Exper.* 1997;9(6):555–577.
156. IBM WebSphere sMash Web 2.0 Workflow system. *IBM DeveloperWorks*. [www.ibm.com/developerworks/websphere/zones/smash/](http://www.ibm.com/developerworks/websphere/zones/smash/); (accessed 19.10.10).
157. Common Component Architecture CCA Forum. In: [www.cca-forum.org/](http://www.cca-forum.org/).
158. Gannon D, Krishnan S, Fang L, et al. On building parallel & grid applications: component technology and distributed services. *Cluster Comput.* 2005;8(4):271–277 In: <http://dx.doi.org/10.1007/s10586-005-4094-2>; 2005.
159. Deelman E, Kosar T, Kesselman C, Livny M. What makes workflows work in an opportunistic environment? *Concurr Comput Pract Exper.* 2006;18(10):10 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
160. Condor home page. In: [www.cs.wisc.edu/condor/](http://www.cs.wisc.edu/condor/).
161. Karajan parallel scripting language. In: <http://wiki.cogkit.org/index.php/Karajan>.
162. GridAnt extension of the Apache Ant build tool residing in the Globus COG kit. In: [www.gridworkflow.org/snips/gridworkflow/space/GridAnt](http://www.gridworkflow.org/snips/gridworkflow/space/GridAnt).
163. Chivers H, McDermid J. Refactoring service-based systems: How to avoid trusting a workflow service. *Concurr Comput Pract Exper.* 2006;18(10):1255–1275 In: <http://dx.doi.org/10.1002/cpe.v18:10>; 2006.
164. S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D.F. Ferguson. Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall, 2005.
165. Atkinson M, DeRoure D, Dunlop A, et al. Web Service Grids: An evolutionary approach. *Concurr Comput Pract Exper.* 2005;17:377–389 In: <http://dx.doi.org/10.1002/cpe.936>; 2005.
166. Segaran T, Evans C, Taylor J. *Programming the Semantic Web* O'Reilly 2009.
167. Fox G. Data and metadata on the semantic grid. *Comput Sci Eng.* 2003;5(5):76–78.
168. Gonzalez H, Halevy A, Jensen CS, et al. Google fusion tables: Data management, integration and collaboration in the cloud International Conference on Management of Data. In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. Indianapolis: ACM; 2010;175–180.

## Homework Problems

### Problem 5.1

Discuss the strengths and drawbacks of WS-\* and RESTful web services. Compare their architectural principles. Which one is the preferred mechanism for communicating with Amazon S3? Why?

## Problem 5.2

Discuss advantages and shortcomings of stateless web services. How can we keep the state in REST and standard web services?

## Problem 5.3

This assignment requires you to combine the queuing and publish-subscribe paradigms within a single application. Organizations A and B are two businesses that use queuing for B2B transactions. Every transaction is stored (prior to forwarding) and has a 128-bit UUID identifier associated with it. Within the organization, messages are delivered using publish-subscribe. Create five subscribers (sales, marketing, audit, packaging, and finance) within each organization; these subscribers do not have to log the messages again since there is a copy of that message already available. If the assignment is done in Java it is prescribed that the Java Message Service be used.

## Problem 5.4

Develop an application that uses publish-subscribe to communicate between entities developed in different languages. Producers should be written in C++, and consumers should be in Java. Distributed components communicate with one another using the AMQP wire format.

## Problem 5.5

Map the terms and abbreviations on the left to the best suited descriptions on the right.

1. _____ S-OGSA	(a) A set of specifications defining a way to describe, publish, and discover information about web services
2. _____ REST	(b) The first language developed for the Semantic web, using XML to represent information about resources on the web
3. _____ UDDI	(c) An XML vocabulary to describe RESTful web services
4. _____ MCS	(d) A family of OASIS-published specifications which provide a set of operations that web services may implement to become stateful
5. _____ WSDL	(e) A catalog service developed by the Globus Alliance providing hierarchical organization of the metadata
6. _____ RDF	(f) Composite Web 2.0 applications which combine capabilities from existing web-based applications
7. _____ WADL	(g) A software architecture for distributed hypermedia systems based on HTTP
8. _____ WSRF	(h) An expressive ontology language that extends RDF schema
9. _____ Mashups	(i) A reference architecture for semantic-based grids by extending Open Grid Services Architecture
10. _____ OWL-S	(j) An XML-based language to describe the set of operations and messages supported by a web service

## **Problem 5.6**

Combine the two assignments 5.3 and 5.4 above, where organization A is developed in Java, while organization B is developed in C++. Use AMQP format in these communications.

## **Problem 5.7**

Describe possible uses of enterprise buses in grid architectures.

## **Problem 5.8**

Run NaradaBrokering or an equivalent system selected from Table 5.8 in an available cloud. Use it to connect two Android smartphones exchanging messages and pictures snapped by the phone.

## **Problem 5.9**

Install the OGCE toolkit using instructions at the web site: [www.collabogce.org/ogce/index.php/Portal\\_download](http://www.collabogce.org/ogce/index.php/Portal_download). Then select one or more from the range of projects given at another web site: [www.collabogce.org/ogce/index.php/Tutorials#Screen\\_Capture\\_Demos](http://www.collabogce.org/ogce/index.php/Tutorials#Screen_Capture_Demos).

## **Problem 5.10**

Compare Google App Engine, Drupal, HUBzero, and OGCE approaches to science gateways. Discuss in the context of portals you are familiar with.

## **Problem 5.11**

What are the three main categories of information that a service registry contains? Which entities map these categories of information in UDDI?

## **Problem 5.12**

Outline the main components of a real-life parallel or distributed job execution environment. You can choose any example platforms such as clusters in Chapter 2, clouds in Chapter 4, grids in Chapter 7, and P2P networks in Chapter 8.

## **Problem 5.13**

Describe the role of a distributed file system in a job execution environment such as MapReduce in a large-scale cloud system.

## Problem 5.14

Resource Description Framework (RDF) and the OWL web ontology language are two technologies for the Semantic web. Describe the relationship between them.

## Problem 5.15

Why are grids popular in academic applications while clouds dominate commercial use? Use case studies to compare their strengths and weaknesses.

## Problem 5.16

Study more details from [138,154]. Implement the Swift application shown in Figure 5.20 on an available cloud platform.

## Problem 5.17

Use Swift to implement the word count problem described in MapReduce (Details of MapReduce are given in Section 6.2.2).

## Problem 5.18

Use Taverna to build a workflow linking a module to extract comments on grids and clouds (or your favorite topic) from Twitter, Flickr, or Facebook. The social networking APIs can be found at [www.programmableweb.com](http://www.programmableweb.com).

<sup>1</sup>HUBzero is a trademark of Purdue University.

# CHAPTER 6

# Cloud Programming and Software Environments

## CHAPTER OUTLINE

### Summary

#### 6.1 Features of Cloud and Grid Platforms

- 6.1.1 Cloud Capabilities and Platform Features
- 6.1.2 Traditional Features Common to Grids and Clouds
- 6.1.3 Data Features and Databases
- 6.1.4 Programming and Runtime Support

#### 6.2 Parallel and Distributed Programming Paradigms

- 6.2.1 Parallel Computing and Programming Paradigms
- 6.2.2 MapReduce, Twister, and Iterative MapReduce
- 6.2.3 Hadoop Library from Apache
- 6.2.4 Dryad and DryadLINQ from Microsoft
- 6.2.5 Sawzall and Pig Latin High-Level Languages
- 6.2.6 Mapping Applications to Parallel and Distributed Systems

#### 6.3 Programming Support of Google App Engine

- 6.3.1 Programming the Google App Engine
- 6.3.2 Google File System (GFS)
- 6.3.3 BigTable, Google's NOSQL System
- 6.3.4 Chubby, Google's Distributed Lock Service

#### 6.4 Programming on Amazon AWS and Microsoft Azure

- 6.4.1 Programming on Amazon EC2
- 6.4.2 Amazon Simple Storage Service (S3)
- 6.4.3 Amazon Elastic Block Store (EBS) and SimpleDB
- 6.4.4 Microsoft Azure Programming Support

#### 6.5 Emerging Cloud Software Environments

- 6.5.1 Open Source Eucalyptus and Nimbus
- 6.5.2 OpenNebula, Sector/Sphere, and OpenStack
- 6.5.3 Manjrasoft Aneka Cloud and Appliances

## 6.6 Bibliographic Notes and Homework Problems

### Acknowledgement

### References

### Homework Problems

This chapter is devoted to programming real cloud platforms. MapReduce, BigTable, Twister, Dryad, DryadLINQ, Hadoop, Sawzall, and Pig Latin are introduced and assessed. We use concrete service examples to explain the implementation and application requirements in the cloud. We review core service models and access technologies. Cloud services provided by Google App Engine (GAE), Amazon Web Service (AWS), and Microsoft Windows Azure are illustrated by example applications. In particular, we illustrate how to program the GAE, AWS EC2, S3, and EBS. We review the open source Eucalyptus, Nimbus, and OpenNebula and the startup Manjrasoft Aneka system for cloud computing.

clouds, grids, parallel and distributed programming, Google App Engine (GAE), Amazon Web Services (AWS), Microsoft Azure, Google File System (GFS)

## Summary

This chapter is devoted to programming real cloud platforms. MapReduce, BigTable, Twister, Dryad, DryadLINQ, Hadoop, Sawzall, and Pig Latin are introduced and assessed. We use concrete examples to explain the implementation and application requirements in the cloud. We review core service models and access technologies. Cloud services provided by Google App Engine (GAE), Amazon Web Service (AWS), and Microsoft Windows Azure are illustrated by example applications. In particular, we illustrate how to programming the GAE, AWS EC2, S3, and EBS. We review the open-source Eucalyptus, Nimbus, and OpenNebula and the startup Manjrasoft Aneka system for cloud computing.

## 6.1 Features of Cloud and Grid Platforms

In this section, we summarize important features in real cloud and grid platforms. In four tables, we cover the capabilities, traditional features, data features, and features for programmers and runtime systems to use. The entries in these tables are source references for anyone who wants to program the cloud efficiently. To get the most from this chapter, readers should refresh their familiarity with and understanding of the languages and software tools for service-oriented architecture (SOA) and web services introduced in [Chapter 5](#).

### 6.1.1 Cloud Capabilities and Platform Features

Commercial clouds need broad capabilities, as summarized in [Table 6.1](#). These capabilities offer cost-effective utility computing with the elasticity to scale up and down in power. However, as well as this key distinguishing feature, commercial clouds offer a growing number of additional capabilities commonly termed “Platform as a Service” (PaaS). For Azure, current platform features include Azure Table, queues, blobs, Database SQL, and web and Worker roles. Amazon is often viewed as offering “just” Infrastructure as a Service (IaaS), but it continues to add platform features including SimpleDB (similar to Azure Table), queues, notification, monitoring, content delivery network, relational database, and MapReduce (Hadoop). Google does not currently offer a broad-based cloud service, but the Google App Engine (GAE) offers a powerful web application development environment.

**Table 6.1**  
**Important Cloud Platform Capabilities**

Capability	Description
Physical or virtual computing platform	The cloud environment consists of some physical or virtual platforms. Virtual platforms have unique capabilities to provide isolated environments for different applications and users.
Massive data storage service, distributed file system	With large data sets, cloud data storage services provide large disk capacity and the service interfaces that allow users to put and get data. The distributed file system offers massive data storage service. It can provide similar interfaces as local file systems.
Massive database storage service	Some distributed file systems are sufficient to provide the underlying storage service application developers need to save data in a more semantic way. Just like DBMS in the traditional software stack, massive database storage services are needed in the cloud.
Massive data processing method and programming model	Cloud infrastructure provides thousands of computing nodes for even a very simple application. Programmers need to be able to harness the power of these machines without considering tedious infrastructure management issues such as handling network failure or scaling the running code to use all the computing facilities provided by the platforms.
Workflow and data query language support	The programming model offers abstraction of the cloud infrastructure. Similar to the SQL language used for database systems, in cloud computing, providers have built some workflow language as well as data query language to support better application logic.
Programming interface and service deployment	Web interfaces or special APIs are required for cloud applications: J2EE, PHP, ASP, or Rails. Cloud applications can use Ajax technologies to improve the user experience while using web browsers to access the functions provided. Each cloud provider opens its programming interface for accessing the data stored in massive storage.
Runtime support	Runtime support is transparent to users and their applications. Support includes distributed monitoring services, a distributed task scheduler, as well as distributed locking and other services. They are critical in running cloud applications.
Support services	Important support services include data and computing services. For example, clouds offer rich data services and interesting data parallel execution models like MapReduce.

[Table 6.2](#) lists some low-level infrastructure features. [Table 6.3](#) lists traditional programming environments for parallel and distributed systems that need to be supported in Cloud environments. They can be supplied as part of system (Cloud

Platform) or user environment. [Table 6.4](#) presents features emphasized by clouds and by some grids. Note that some of the features in [Table 6.4](#) have only recently been offered in a major way. In particular, these features are not offered on academic cloud infrastructures such as Eucalyptus, Nimbus, OpenNebula, or Sector/Sphere (although Sector is a data parallel file system or DPFS classified in [Table 6.4](#)). We will cover these emerging cloud programming environments in [Section 6.5](#).

---

## Table 6.2

### Infrastructure Cloud Features

---

**Accounting:** Includes economies; clearly an active area for commercial clouds

**Appliances:** Preconfigured virtual machine (VM) image supporting multifaceted tasks such as message-passing interface (MPI) clusters

**Authentication and authorization:** Could need single sign-on to multiple systems

**Data transport:** Transports data between job components both between and within grids and clouds; exploits custom storage patterns as in BitTorrent

**Operating systems:** Apple, Android, Linux, Windows

**Program library:** Stores images and other program material

**Registry:** Information resource for system (system version of metadata management)

**Security:** Security features other than basic authentication and authorization; includes higher level concepts such as trust

**Scheduling:** Basic staple of Condor, Platform, Oracle Grid Engine, etc.; clouds have this implicitly as is especially clear with Azure Worker Role

**Gang scheduling:** Assigns multiple (data-parallel) tasks in a scalable fashion; note that this is provided automatically by MapReduce

**Software as a Service (SaaS):** Shared between clouds and grids, and can be supported without special attention; Note use of services and corresponding service oriented architectures are very successful and are used in clouds very similarly to previous distributed systems.

**Virtualization:** Basic feature of clouds supporting “elastic” feature highlighted by Berkeley as characteristic of what defines a (public) cloud; includes virtual networking as in ViNe from University of Florida

**Table 6.3****Traditional Features in Cluster, Grid, and Parallel Computing Environments**

**Cluster management:** ROCKS and packages offering a range of tools to make it easy to bring up clusters

**Data management:** Included metadata support such as RDF triple stores (Semantic web success and can be built on MapReduce as in SHARD); SQL and NOSQL included in

**Grid programming environment:** Varies from link-together services as in Open Grid Services Architecture (OGSA) to GridRPC (Ninf, GridSolve) and SAGA

**OpenMP/threading:** Can include parallel compilers such as Cilk; roughly shared memory technologies. Even transactional memory and fine-grained data flow come here

**Portals:** Can be called (science) gateways and see an interesting change in technology from portlets to HUBzero and now in the cloud: Azure Web Roles and GAE

**Scalable parallel computing environments:** MPI and associated higher level concepts including ill-fated HP FORTRAN, PGAS (not successful but not disgraced), HPCS languages (X-10, Fortress, Chapel), patterns (including Berkeley dwarves), and functional languages such as F# for distributed memory

**Virtual organizations:** From specialized grid solutions to popular Web 2.0 capabilities such as Facebook

**Workflow:** Supports workflows that link job components either within or between grids and clouds; relate to LIMS Laboratory Information Management Systems.

**Table 6.4****Platform Features Supported by Clouds and (Sometimes) Grids**

**Blob:** Basic storage concept typified by Azure Blob and Amazon S3

**DPFS:** Support of file systems such as Google (MapReduce), HDFS (Hadoop), and Cosmos (Dryad) with compute-data affinity optimized for data processing

**Fault tolerance:** As reviewed in [1] this was largely ignored in grids, but is a major feature of clouds

**MapReduce:** Support MapReduce programming model including Hadoop on Linux, Dryad on Windows HPCS, and Twister on Windows and Linux. Include new associated languages such as Sawzall, Pregel, Pig Latin, and LINQ

**Monitoring:** Many grid solutions such as Inca. Can be based on publish-subscribe

**Notification:** Basic function of publish-subscribe systems

**Programming model:** Cloud programming models are built with other platform features and are related to familiar web and grid models

**Queues:** Queuing system possibly based on publish-subscribe

**Scalable synchronization:** Apache Zookeeper or Google Chubby. Supports distributed locks and used by BigTable. Not clear if (effectively) used in Azure Table or Amazon SimpleDB

**SQL:** Relational database

**Table:** Support of table data structures modeled on Apache Hbase or Amazon SimpleDB/Azure Table. Part of NOSQL movement

**Web role:** Used in Azure to describe important link to user and can be supported otherwise with a portal framework. This is the main purpose of GAE

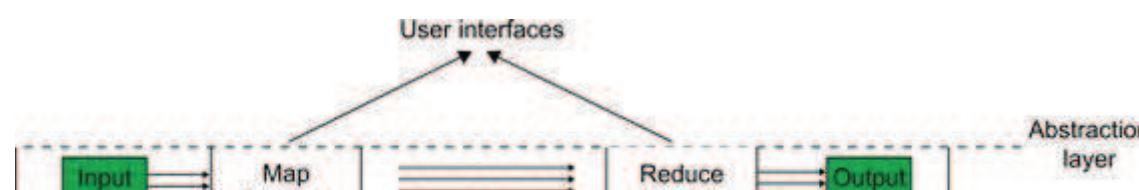
**Worker role:** Implicitly used in both Amazon and grids but was first introduced as a high-level construct by Azure

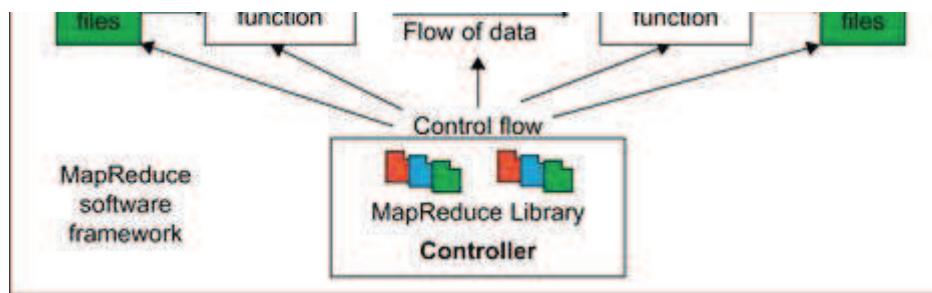
## 6.1.2 Traditional Features Common to Grids and Clouds

In this section, we concentrate on features related to workflow, data transport, security, and availability concerns that are common to today's computing grids and clouds.

### 6.1.2.1 Workflow

As introduced in [Section 5.5](#), workflow has spawned many projects in the United States and Europe. Pegasus, Taverna, and Kepler are popular, but no choice has gained wide acceptance. There are commercial systems such as Pipeline Pilot, AVS (dated), and the LIMS environments. A recent entry is Trident [2] from Microsoft Research which is built on top of Windows Workflow Foundation. If Trident runs on Azure or just any old Windows machine, it will run workflow proxy services on external (Linux) environments. Workflow links multiple cloud and noncloud services in real applications on demand.





**FIGURE 6.1** MapReduce framework: Input data flows through the Map and Reduce functions to generate the output result under the control flow using MapReduce software library. Special user interfaces are used to access the Map and Reduce resources.

### 6.1.2.2 Data Transport

The cost (in time and money) of data transport in (and to a lesser extent, out of) commercial clouds is often discussed as a difficulty in using clouds. If commercial clouds become an important component of the national cyberinfrastructure we can expect that high-bandwidth links will be made available between clouds and TeraGrid. The special structure of cloud data with blocks (in Azure blobs) and tables could allow high-performance parallel algorithms, but initially, simple HTTP mechanisms are used to transport data [3–5] on academic systems/TeraGrid and commercial clouds.

### 6.1.2.3 Security, Privacy, and Availability

The following techniques are related to security, privacy, and availability requirements for developing a healthy and dependable cloud programming environment. We summarize these techniques here. Some of these issues have been discussed in Section 4.4.6 with plausible solutions.

- Use virtual clustering to achieve dynamic resource provisioning with minimum overhead cost.
- Use stable and persistent data storage with fast queries for information retrieval.
- Use special APIs for authenticating users and sending e-mail using commercial accounts.
- Cloud resources are accessed with security protocols such as HTTPS and SSL.
- Fine-grained access control is desired to protect data integrity and deter intruders or hackers.
- Shared data sets are protected from malicious alteration, deletion, or copyright violations.
- Features are included for availability enhancement and disaster recovery with life migration of VMs.
- Use a reputation system to protect data centers. This system only authorizes trusted clients and stops pirates.

## 6.1.3 Data Features and Databases

In the following paragraphs, we review interesting programming features related to the

program library, blobs, drives, DPFS, tables, and various types of databases including SQL, NOSQL, and nonrelational databases and special queuing services.

### **6.1.3.1 Program Library**

Many efforts have been made to design a VM image library to manage images used in academic and commercial clouds. The basic cloud environments described in this chapter also include many management features allowing convenient deployment and configuring of images (i.e., they support IaaS).

### **6.1.3.2 Blobs and Drives**

The basic storage concept in clouds is blobs for Azure and S3 for Amazon. These can be organized (approximately, as in directories) by containers in Azure. In addition to a service interface for blobs and S3, one can attach “directly” to compute instances as Azure drives and the Elastic Block Store for Amazon. This concept is similar to shared file systems such as Lustre used in TeraGrid. The cloud storage is intrinsically fault-tolerant while that on TeraGrid needs backup storage. However, the architecture ideas are similar between clouds and TeraGrid, and the Simple Cloud File Storage API [6] could become important here.

### **6.1.3.3 DPFS**

This covers the support of file systems such as Google File System (MapReduce), HDFS (Hadoop), and Cosmos (Dryad) with compute-data affinity optimized for data processing. It could be possible to link DPFS to basic blob and drive-based architecture, but it's simpler to use DPFS as an application-centric storage model with compute-data affinity and blobs and drives as the repository-centric view.

In general, data transport will be needed to link these two data views. It seems important to consider this carefully, as DPFS file systems are precisely designed for efficient execution of data-intensive applications. However, the importance of DPFS for linkage with Amazon and Azure is not clear, as these clouds do not currently offer fine-grained support for compute-data affinity. We note here that Azure Affinity Groups are one interesting capability [7]. We expect that initially blobs, drives, tables, and queues will be the areas where academic systems will most usefully provide a platform similar to Azure (and Amazon). Note the HDFS (Apache) and Sector (UIC) projects in this area.

### **6.1.3.4 SQL and Relational Databases**

Both Amazon and Azure clouds offer relational databases and it is straightforward for academic systems to offer a similar capability unless there are issues of huge scale where, in fact, approaches based on tables and/or MapReduce might be more appropriate [8]. As one early user, we are developing on FutureGrid a new private cloud computing model for the Observational Medical Outcomes Partnership (OMOP) for patient-related medical data which uses Oracle and SAS where FutureGrid is adding Hadoop for scaling to many different analysis methods.

Note that databases can be used to illustrate two approaches to deploying capabilities. Traditionally, one would add database software to that found on computer disks. This software is executed, providing your database instance. However, on Azure and Amazon, the database is installed on a separate VM independent from your job (worker roles in Azure). This implements “SQL as a Service.” It may have some performance issues from the messaging interface, but the “aaS” deployment clearly simplifies one’s system. For  $N$  platform features, one only needs  $N$  services, whereas number of possible images with alternative approaches is a prohibitive  $2^N$ .

### **6.1.3.5 Table and NOSQL Nonrelational Databases**

A substantial number of important developments have occurred regarding simplified database structures—termed “NOSQL” [9,10]—typically emphasizing distribution and scalability. These are present in the three major clouds: BigTable [11] in Google, SimpleDB [12] in Amazon, and Azure Table [13] for Azure. Tables are clearly important in science as illustrated by the VOTable standard in astronomy [14] and the popularity of Excel. However, there does not appear to be substantial experience in using tables outside clouds.

There are, of course, many important uses of nonrelational databases, especially in terms of triple stores for metadata storage and access. Recently, there has been interest in building scalable RDF triple stores based on MapReduce and tables or the Hadoop File System [8,15], with early success reported on very large stores. The current cloud tables fall into two groups: Azure Table and Amazon SimpleDB are quite similar [16] and support lightweight storage for “document stores,” while BigTable aims to manage large distributed data sets without size limitations.

All these tables are schema-free (each record can have different properties), although BigTable has a schema for column (property) families. It seems likely that tables will grow in importance for scientific computing, and academic systems could support this using two Apache projects: Hbase [17] for BigTable and CouchDB [18] for a document store. Another possibility is the open source SimpleDB implementation M/DB [19]. The new Simple Cloud APIs [6] for file storage, document storage services, and simple queues could help in providing a common environment between academic and commercial clouds.

### **6.1.3.6 Queuing Services**

Both Amazon and Azure offer similar scalable, robust queuing services that are used to communicate between the components of an application. The messages are short (less than 8 KB) and have a Representational State Transfer (REST) service interface with “deliver at least once” semantics. They are controlled by timeouts for posting the length of time allowed for a client to process. One can build a similar approach (on the typically smaller and less challenging academic environments), basing it on publish-subscribe systems such as ActiveMQ [20] or NaradaBrokering [21,22] with which we have substantial experience.

## 6.1.4 Programming and Runtime Support

Programming and runtime support are desired to facilitate parallel programming and provide runtime support of important functions in today's grids and clouds. Various MapReduce systems are reviewed in this section.

### 6.1.4.1 Worker and Web Roles

The roles introduced by Azure provide nontrivial functionality, while preserving the better affinity support that is possible in a nonvirtualized environment. Worker roles are basic schedulable processes and are automatically launched. Note that explicit scheduling is unnecessary in clouds for individual worker roles and for the "gang scheduling" supported transparently in MapReduce. Queues are a critical concept here, as they provide a natural way to manage task assignment in a fault-tolerant, distributed fashion. Web roles provide an interesting approach to portals. GAE is largely aimed at web applications, whereas science gateways are successful in TeraGrid.

### 6.1.4.2 MapReduce

There has been substantial interest in "data parallel" languages largely aimed at loosely coupled computations which execute over different data samples. The language and runtime generate and provide efficient execution of "many task" problems that are well known as successful grid applications. However, MapReduce, summarized in [Table 6.5](#), has several advantages over traditional implementations for many task problems, as it supports dynamic execution, strong fault tolerance, and an easy-to-use high-level interface. The major open source/commercial MapReduce implementations are Hadoop [23] and Dryad [24–27] with execution possible with or without VMs.

---

**Table 6.5**

**Comparison of MapReduce Type Systems**

---

	<b>Google MapReduce [28]</b>	<b>Apache Hadoop [23]</b>	<b>Microsoft Dryad [26]</b>	<b>Twister [29]</b>	<b>Azure Twister [30]</b>
Programming Model	MapReduce	MapReduce	DAG execution, extensible to MapReduce and other patterns	Iterative MapReduce	Currently just MapReduce; will extend to Iterative MapReduce
Data Handling	GFS (Google File System)	HDFS (Hadoop Distributed File System)	Shared directories and local disks	Local disks and data management tools	Azure blob storage
Scheduling	Data locality	Data locality; rack-aware, dynamic task scheduling using global queue	Data locality; network topology optimized at runtime; static task partitions	Data locality; static task partitions	Dynamic task scheduling through global queue
Failure Handling	Reexecution of failed tasks; duplicated execution of slow tasks	Reexecution of failed tasks; duplicated execution of slow tasks	Reexecution of failed tasks; duplicated execution of slow tasks	Reexecution of iterations	Reexecution of failed tasks; duplicated execution of slow tasks
HLL Support	Sawzall [31]	Pig Latin [32,33]	DryadLINQ [27]	Pregel [34] has related features	N/A
Environment	Linux cluster	Linux clusters, Amazon Elastic MapReduce on EC2	Windows HPCS cluster	Linux cluster, EC2	Windows Azure, Azure Local Development Fabric
Intermediate Data Transfer	File	File, HTTP	File, TCP pipes, shared-memory FIFOs	Publish-subscribe messaging	Files, TCP

Hadoop is currently offered by Amazon, and we expect Dryad to be available on Azure. A prototype Azure MapReduce was built at Indiana University, which we will discuss shortly. On FutureGrid, we already intend to support Hadoop, Dryad, and other MapReduce approaches, including Twister [29] support for iterative computations seen in many data-mining and linear algebra applications. Note that this approach has some similarities with Cloudera [35] which offers a variety of Hadoop distributions including Amazon and Linux. MapReduce is closer to broad deployment than other cloud platform features, as there is quite a bit of experience with Hadoop and Dryad outside clouds.

#### 6.1.4.3 Cloud Programming Models

In many ways, most of the previous sections describe programming model features, but these are “macroscopic” constructs and do not address, for example, the coding (language and libraries). Both the GAE and Manjrasoft Aneka environments represent programming models; both are applied to clouds, but are really not specific to this architecture. Iterative MapReduce is an interesting programming model that offers

portability between cloud, HPC and cluster environments.

#### 6.1.4.4 SaaS

Services are used in a similar fashion in commercial clouds and most modern distributed systems. We expect users to package their programs wherever possible, so no special support is needed to enable SaaS. We already discussed in [Section 6.1.3](#) why “Systems software as a service” was an interesting idea in the context of a database service. We desire a SaaS environment that provides many useful tools to develop cloud applications over large data sets. In addition to the technical features, such as MapReduce, BigTable, EC2, S3, Hadoop, AWS, GAE, and WebSphere2, we need protection features that may help us to achieve scalability, security, privacy, and availability.

## 6.2 Parallel and Distributed Programming Paradigms

We define a parallel and distributed program as a parallel program running on a set of computing engines or a distributed computing system. The term carries the notion of two fundamental terms in computer science: distributed computing system and parallel computing. A distributed computing system is a set of computational engines connected by a network to achieve a common goal of running a job or an application. A computer cluster or network of workstations is an example of a distributed computing system. Parallel computing is the simultaneous use of more than one computational engine (not necessarily connected via a network) to run a job or an application. For instance, parallel computing may use either a distributed or a nondistributed computing system such as a multiprocessor platform.

Running a parallel program on a distributed computing system (parallel and distributed programming) has several advantages for both users and distributed computing systems. From the users' perspective, it decreases application response time; from the distributed computing systems' standpoint, it increases throughput and resource utilization. Running a parallel program on a distributed computing system, however, could be a very complicated process. Therefore, to place the complexity in perspective, data flow of running a typical parallel program on a distributed system is further explained in this chapter.

### 6.2.1 Parallel Computing and Programming Paradigms

Consider a distributed computing system consisting of a set of networked nodes or workers. The system issues for running a typical parallel program in either a parallel or a distributed manner would include the following [36–39]:

- **Partitioning** This is applicable to both computation and data as follows:
- **Computation partitioning** This splits a given job or a program into smaller tasks.

Partitioning greatly depends on correctly identifying portions of the job or program that can be performed concurrently. In other words, upon identifying parallelism in the structure of the program, it can be divided into parts to be run on different

workers. Different parts may process different data or a copy of the same data.

- **Data partitioning** This splits the input or intermediate data into smaller pieces. Similarly, upon identification of parallelism in the input data, it can also be divided into pieces to be processed on different workers. Data pieces may be processed by different parts of a program or a copy of the same program.
- **Mapping** This assigns the either smaller parts of a program or the smaller pieces of data to underlying resources. This process aims to appropriately assign such parts or pieces to be run simultaneously on different workers and is usually handled by resource allocators in the system.
- **Synchronization** Because different workers may perform different tasks, synchronization and coordination among workers is necessary so that race conditions are prevented and data dependency among different workers is properly managed. Multiple accesses to a shared resource by different workers may raise race conditions, whereas data dependency happens when a worker needs the processed data of other workers.
- **Communication** Because data dependency is one of the main reasons for communication among workers, communication is always triggered when the intermediate data is sent to workers.
- **Scheduling** For a job or program, when the number of computation parts (tasks) or data pieces is more than the number of available workers, a scheduler selects a sequence of tasks or data pieces to be assigned to the workers. It is worth noting that the resource allocator performs the actual mapping of the computation or data pieces to workers, while the scheduler only picks the next part from the queue of unassigned tasks based on a set of rules called the scheduling policy. For multiple jobs or programs, a scheduler selects a sequence of jobs or programs to be run on the distributed computing system. In this case, scheduling is also necessary when system resources are not sufficient to simultaneously run multiple jobs or programs.

### **6.2.1.1 Motivation for Programming Paradigms**

Because handling the whole data flow of parallel and distributed programming is very time-consuming and requires specialized knowledge of programming, dealing with these issues may affect the productivity of the programmer and may even result in affecting the program's time to market. Furthermore, it may detract the programmer from concentrating on the logic of the program itself. Therefore, parallel and distributed programming paradigms or models are offered to abstract many parts of the data flow from users.

In other words, these models aim to provide users with an abstraction layer to hide implementation details of the data flow which users formerly ought to write codes for. Therefore, simplicity of writing parallel programs is an important metric for parallel and distributed programming paradigms. Other motivations behind parallel and distributed programming models are (1) to improve productivity of programmers, (2) to decrease programs' time to market, (3) to leverage underlying resources more efficiently, (4) to increase system throughput, and (5) to support higher levels of abstraction [40].

MapReduce, Hadoop, and Dryad are three of the most recently proposed parallel and distributed programming models. They were developed for information retrieval applications but have been shown to be applicable for a variety of important applications [41]. Further, the loose coupling of components in these paradigms makes them suitable for VM implementation and leads to much better fault tolerance and scalability for some applications than traditional parallel computing models such as MPI [42–44].

## 6.2.2 MapReduce, Twister, and Iterative MapReduce

MapReduce, as introduced in Section 6.1.4, is a software framework which supports parallel and distributed computing on large data sets [27,37,45,46]. This software framework abstracts the data flow of running a parallel program on a distributed computing system by providing users with two interfaces in the form of two functions: *Map* and *Reduce*. Users can override these two functions to interact with and manipulate the data flow of running their programs. Figure 6.1 illustrates the logical data flow from the *Map* to the *Reduce* function in MapReduce frameworks. In this framework, the “value” part of the data,  $(key, value)$ , is the actual data, and the “key” part is only used by the MapReduce controller to control the data flow [37].

### 6.2.2.1 Formal Definition of MapReduce

The MapReduce software framework provides an abstraction layer with the data flow and flow of control to users, and hides the implementation of all data flow steps such as data partitioning, mapping, synchronization, communication, and scheduling. Here, although the data flow in such frameworks is predefined, the abstraction layer provides two well-defined interfaces in the form of two functions: *Map* and *Reduce* [47]. These two main functions can be overridden by the user to achieve specific objectives. Figure 6.1 shows the MapReduce framework with data flow and control flow.

Therefore, the user overrides the *Map* and *Reduce* functions first and then invokes the provided *MapReduce (Spec, & Results)* function from the library to start the flow of data. The *MapReduce* function, *MapReduce (Spec, & Results)*, takes an important parameter which is a specification object, the *Spec*. This object is first initialized inside the user’s program, and then the user writes code to fill it with the names of input and output files, as well as other optional tuning parameters. This object is also filled with the name of the *Map* and *Reduce* functions to identify these user-defined functions to the MapReduce library.

The overall structure of a user’s program containing the *Map*, *Reduce*, and the *Main* functions is given below. The *Map* and *Reduce* are two major subroutines. They will be called to implement the desired function performed in the main program.

```
Map Function (... . )
```

```
{  
... ...  
}
```

```
Reduce Function (... . .)
```

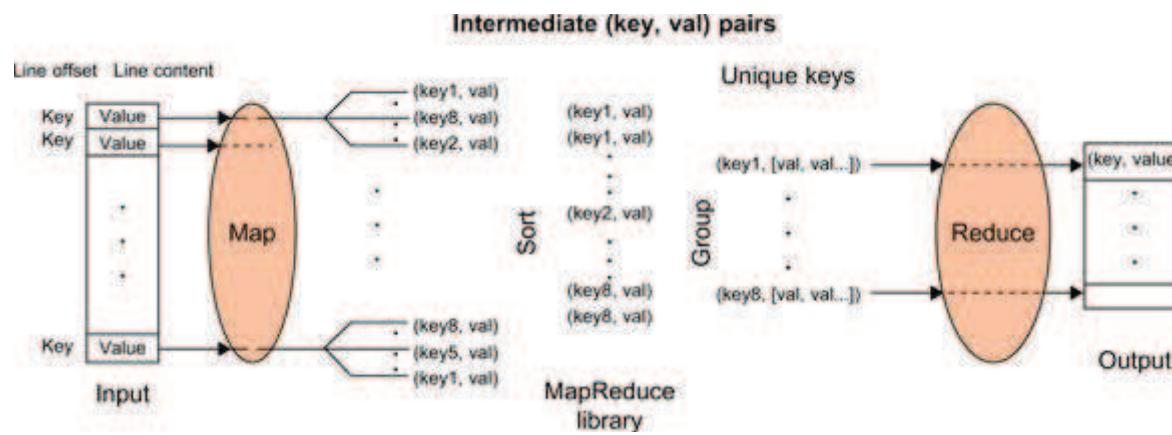
```
{  
... ...  
}
```

```
Main Function (... . .)
```

```
{  
Initialize Spec object  
... ...  
MapReduce (Spec, & Results)  
}
```

### 6.2.2.2 MapReduce Logical Data Flow

The input data to both the *Map* and the *Reduce* functions has a particular structure. This also pertains for the output data. The input data to the *Map* function is in the form of a (key, value) pair. For example, the key is the line offset within the input file and the value is the content of the line. The output data from the *Map* function is structured as (key, value) pairs called intermediate (key, value) pairs. In other words, the user-defined *Map* function processes each input (key, value) pair and produces a number of (zero, one, or more) intermediate (key, value) pairs. Here, the goal is to process all input (key, value) pairs to the *Map* function in parallel (Figure 6.2).

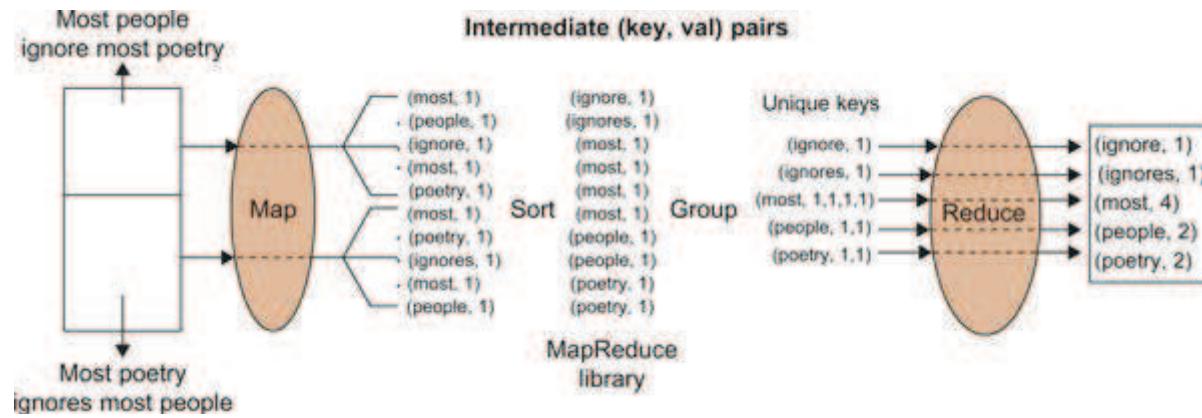


**FIGURE 6.2** MapReduce logical data flow in 5 processing stages over successive (key, value) pairs.

In turn, the *Reduce* function receives the intermediate (key, value) pairs in the form of a group of intermediate values associated with one intermediate key, *(key, [set of values])*. In fact, the MapReduce framework forms these groups by first sorting the intermediate (key, value) pairs and then grouping values with the same key. It should be noted that the data is sorted to simplify the grouping process. The *Reduce* function processes each *(key, [set of values])* group and produces a set of (key, value) pairs as output.

To clarify the data flow in a sample MapReduce application, one of the well-known

MapReduce problems, namely word count, to count the number of occurrences of each word in a collection of documents is presented here. Figure 6.3 demonstrates the data flow of the word-count problem for a simple input file containing only two lines as follows: (1) “most people ignore most poetry” and (2) “most poetry ignores most people.” In this case, the *Map* function simultaneously produces a number of intermediate (key, value) pairs for each line of content so that each word is the intermediate key with 1 as its intermediate value; for example, (*ignore*, 1). Then the MapReduce library collects all the generated intermediate (key, value) pairs and sorts them to group the 1’s for identical words; for example, (*people*, [1,1]). Groups are then sent to the *Reduce* function in parallel so that it can sum up the 1 values for each word and generate the actual number of occurrence for each word in the file; for example, (*people*, 2).



**FIGURE 6.3** The data flow of a word-count problem using the MapReduce functions (Map, Sort, Group and Reduce) in a cascade operations.

### 6.2.2.3 Formal Notation of MapReduce Data Flow

The *Map* function is applied in parallel to every input (key, value) pair, and produces new set of intermediate (key, value) pairs [37] as follows:

(6.1)

Then the MapReduce library collects all the produced intermediate (key, value) pairs from all input (key, value) pairs, and sorts them based on the “key” part. It then groups the values of all occurrences of the same key. Finally, the *Reduce* function is applied in parallel to each group producing the collection of values as output as illustrated here:

(6.2)

### 6.2.2.4 Strategy to Solve MapReduce Problems

As mentioned earlier, after grouping all the intermediate data, the values of all

occurrences of the same key are sorted and grouped together. As a result, after grouping, each key becomes unique in all intermediate data. Therefore, finding unique keys is the starting point to solving a typical MapReduce problem. Then the intermediate (key, value) pairs as the output of the *Map* function will be automatically found. The following three examples explain how to define keys and values in such problems:

**Problem 1:** Counting the number of occurrences of each word in a collection of documents

*Solution:* unique “key”: each word, intermediate “value”: number of occurrences

**Problem 2:** Counting the number of occurrences of words having the same size, or the same number of letters, in a collection of documents

*Solution:* unique “key”: each word, intermediate “value”: size of the word

**Problem 3:** Counting the number of occurrences of anagrams in a collection of documents. Anagrams are words with the same set of letters but in a different order (e.g., the words “listen” and “silent”).

*Solution:* unique “key”: alphabetically sorted sequence of letters for each word (e.g., “eilnst”), intermediate “value”: number of occurrences

### 6.2.2.5 MapReduce Actual Data and Control Flow

The main responsibility of the MapReduce framework is to efficiently run a user’s program on a distributed computing system. Therefore, the MapReduce framework meticulously handles all partitioning, mapping, synchronization, communication, and scheduling details of such data flows [48,49]. We summarize this in the following distinct steps:

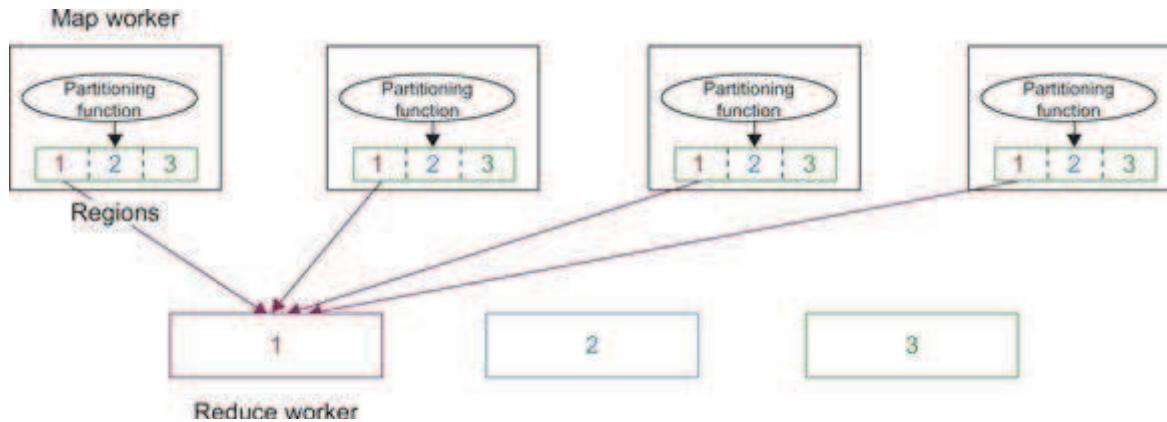
1. **Data partitioning** The MapReduce library splits the input data (files), already stored in GFS, into  $M$  pieces that also correspond to the number of map tasks.
2. **Computation partitioning** This is implicitly handled (in the MapReduce framework) by obliging users to write their programs in the form of the *Map* and *Reduce* functions. Therefore, the MapReduce library only generates copies of a user program (e.g., by a fork system call) containing the *Map* and the *Reduce* functions, distributes them, and starts them up on a number of available computation engines.
3. **Determining the master and workers** The MapReduce architecture is based on a master-worker model. Therefore, one of the copies of the user program becomes the master and the rest become workers. The master picks idle workers, and assigns the map and reduce tasks to them.  
A map/reduce *worker* is typically a computation engine such as a cluster node to run map/reduce *tasks* by executing *Map/Reduce functions*. Steps 4–7 describe the map workers.
4. **Reading the input data (data distribution)** Each map worker reads its corresponding portion of the input data, namely the input data split, and sends it to its *Map* function. Although a map worker may run more than one *Map* function, which means it has been assigned more than one input data split, each worker is usually assigned one input split only.
5. **Map function** Each *Map* function receives the input data split as a set of (key, value)

pairs to process and produce the intermediated (key, value) pairs.

6. **Combiner function** This is an optional local function within the map worker which applies to intermediate (key, value) pairs. The user can invoke the *Combiner* function inside the user program. The *Combiner* function runs the same code written by users for the *Reduce* function as its functionality is identical to it. The *Combiner* function merges the local data of each map worker before sending it over the network to effectively reduce its communication costs. As mentioned in our discussion of logical data flow, the MapReduce framework sorts and groups the data before it is processed by the *Reduce* function. Similarly, the MapReduce framework will also sort and group the local data on each map worker if the user invokes the *Combiner* function.
7. **Partitioning function** As mentioned in our discussion of the MapReduce data flow, the intermediate (key, value) pairs with identical keys are grouped together because all values inside each group should be processed by only one *Reduce* function to generate the final result. However, in real implementations, since there are  $M$  map and  $R$  reduce tasks, intermediate (key, value) pairs with the same key might be produced by different map tasks, although they should be grouped and processed together by one *Reduce* function only.  
Therefore, the intermediate (key, value) pairs produced by each map worker are partitioned into  $R$  regions, equal to the number of reduce tasks, by the *Partitioning* function to guarantee that all (key, value) pairs with identical keys are stored in the same region. As a result, since reduce worker  $i$  reads the data of region  $i$  of all map workers, all (key, value) pairs with the same key will be gathered by reduce worker  $i$  accordingly (see [Figure 6.4](#)). To implement this technique, a *Partitioning* function could simply be a hash function (e.g.,  $\text{Hash}(\text{key}) \bmod R$ ) that forwards the data into particular regions. It is also worth noting that the locations of the buffered data in these  $R$  partitions are sent to the master for later forwarding of data to the reduce workers.  
[Figure 6.5](#) shows the data flow implementation of all data flow steps. The following are two networking steps:
8. **Synchronization** MapReduce applies a simple synchronization policy to coordinate map workers with reduce workers, in which the communication between them starts when all map tasks finish.
9. **Communication** Reduce worker  $i$ , already notified of the location of region  $i$  of all map workers, uses a remote procedure call to read the data from the respective region of all map workers. Since all reduce workers read the data from all map workers, all-to-all communication among all map and reduce workers, which incurs network congestion, occurs in the network. This issue is one of the major bottlenecks in increasing the performance of such systems [50–52]. A data transfer module was proposed to schedule data transfers independently [55].  
Steps 10 and 11 correspond to the reduce worker domain:
10. **Sorting and Grouping** When the process of reading the input data is finalized by a reduce worker, the data is initially buffered in the local disk of the reduce worker. Then the reduce worker groups intermediate (key, value) pairs by sorting the data based on their keys, followed by grouping all occurrences of identical keys. Note that the

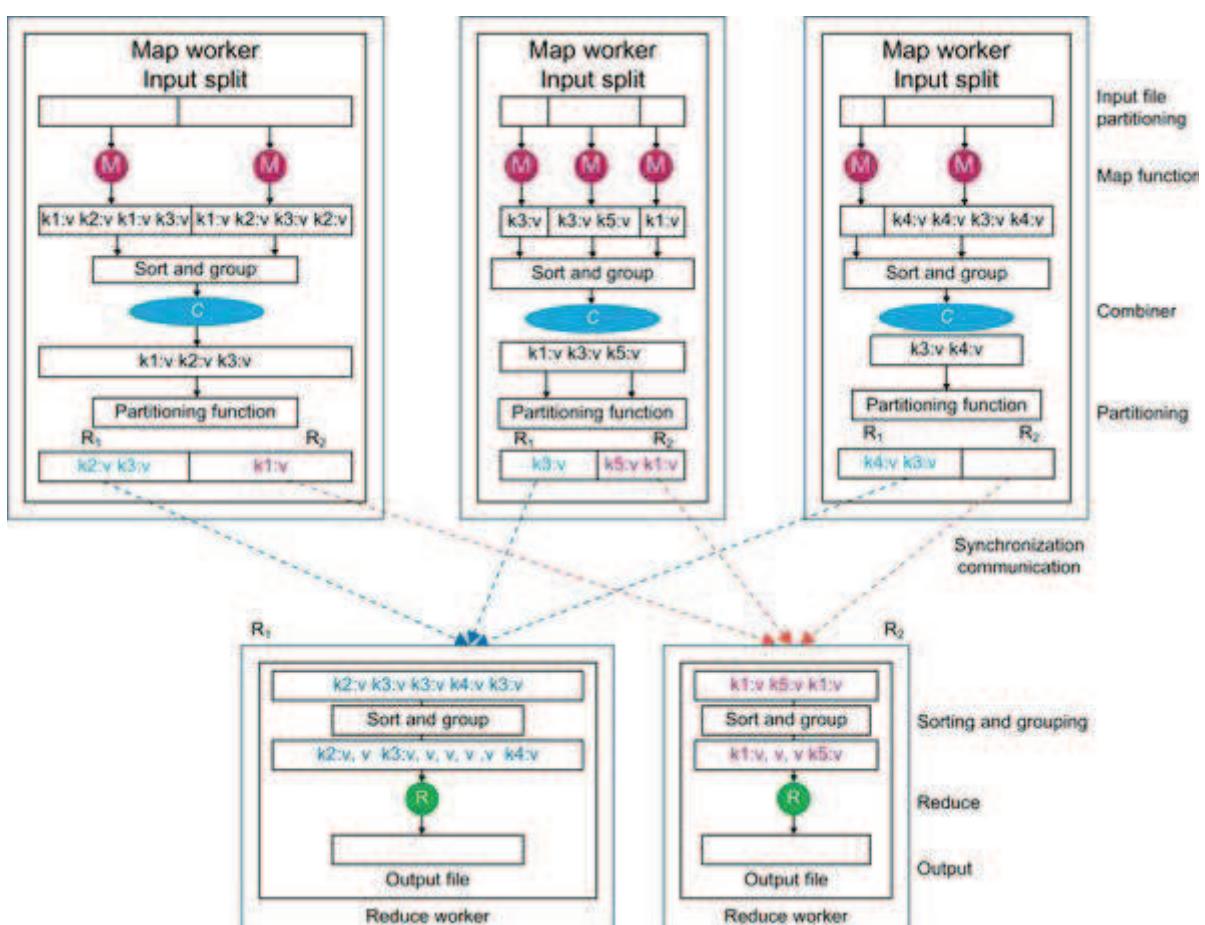
buffered data is sorted and grouped because the number of unique keys produced by a map worker may be more than  $R$  regions in which more than one key exists in each region of a map worker (see [Figure 6.4](#)).

- 1. **Reduce function** The reduce worker iterates over the grouped (key, value) pairs, and for each unique key, it sends the key and corresponding values to the *Reduce* function. Then this function processes its input data and stores the output results in predetermined files in the user's program.

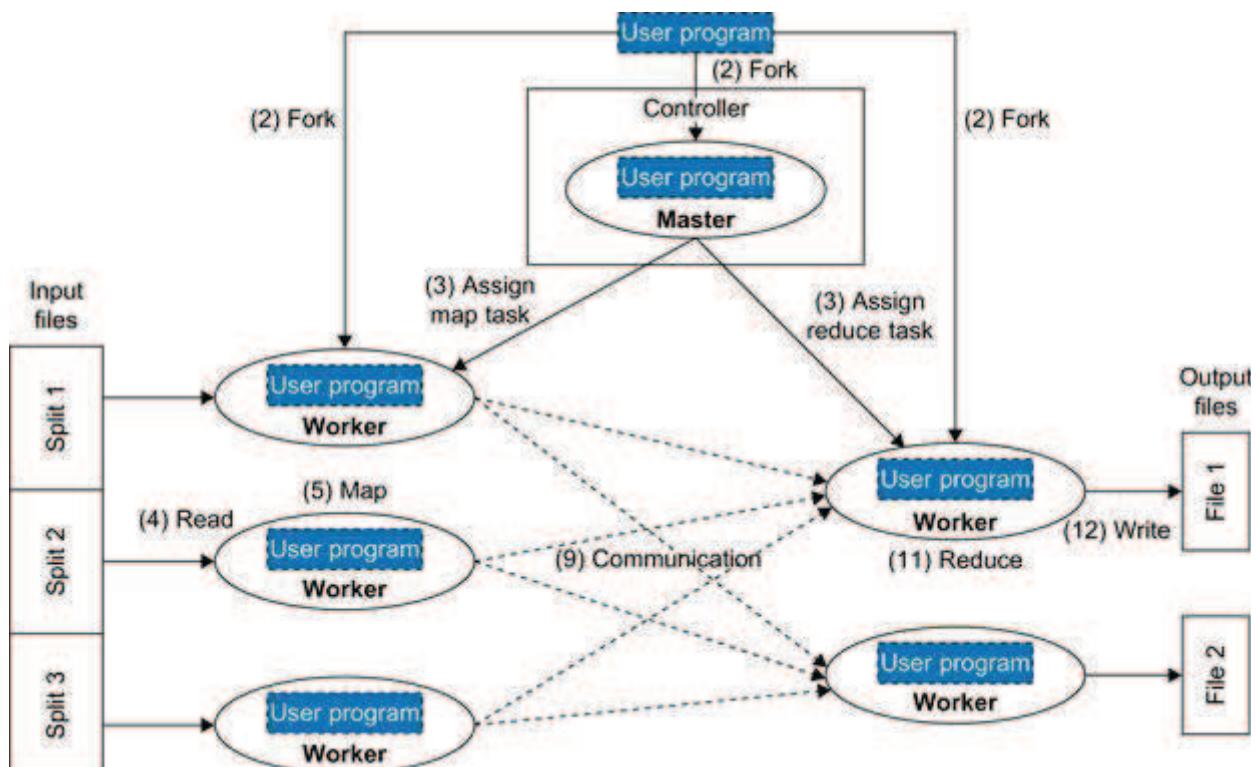


**FIGURE 6.4** Use of MapReduce *partitioning* function to link the Map and Reduce workers.

To better clarify the interrelated data control and control flow in the MapReduce framework, [Figure 6.6](#) shows the exact order of processing control in such a system contrasting with dataflow in [Figure 6.5](#).



**FIGURE 6.5** Data flow implementation of many functions in the Map workers and in the Reduce workers through multiple sequences of partitioning, combining, synchronization and communication, sorting and grouping, and reduce operations.



**FIGURE 6.6** Control flow implementation of the MapReduce functionalities in Map workers and Reduce workers (running user programs) from input files to the output files under the control of the master user program. Courtesy of

### 6.2.2.6 Compute-Data Affinity

The MapReduce software framework was first proposed and implemented by Google. The first implementation was coded in C. The implementation takes advantage of GFS [53] as the underlying layer. MapReduce could perfectly adapt itself to GFS. GFS is a distributed file system where files are divided into fixed-size blocks (chunks) and blocks are distributed and stored on cluster nodes.

As stated earlier, the MapReduce library splits the input data (files) into fixed-size blocks, and ideally performs the *Map* function in parallel on each block. In this case, as GFS has already stored files as a set of blocks, the MapReduce framework just needs to send a copy of the user's program containing the *Map* function to the nodes' already stored data blocks. This is the notion of sending computation toward data rather than sending data toward computation. Note that the default GFS block size is 64 MB which is identical to that of the MapReduce framework.

### 6.2.2.7 Twister and Iterative MapReduce

It is important to understand the performance of different runtimes and, in particular, to compare MPI and MapReduce [43,44,55,56]. The two major sources of parallel overhead are load imbalance and communication (which is equivalent to synchronization overhead as communication synchronizes parallel units [threads or processes] in Categories 2 and 6 of Table 6.10). The communication overhead in MapReduce can be quite high, for two reasons:

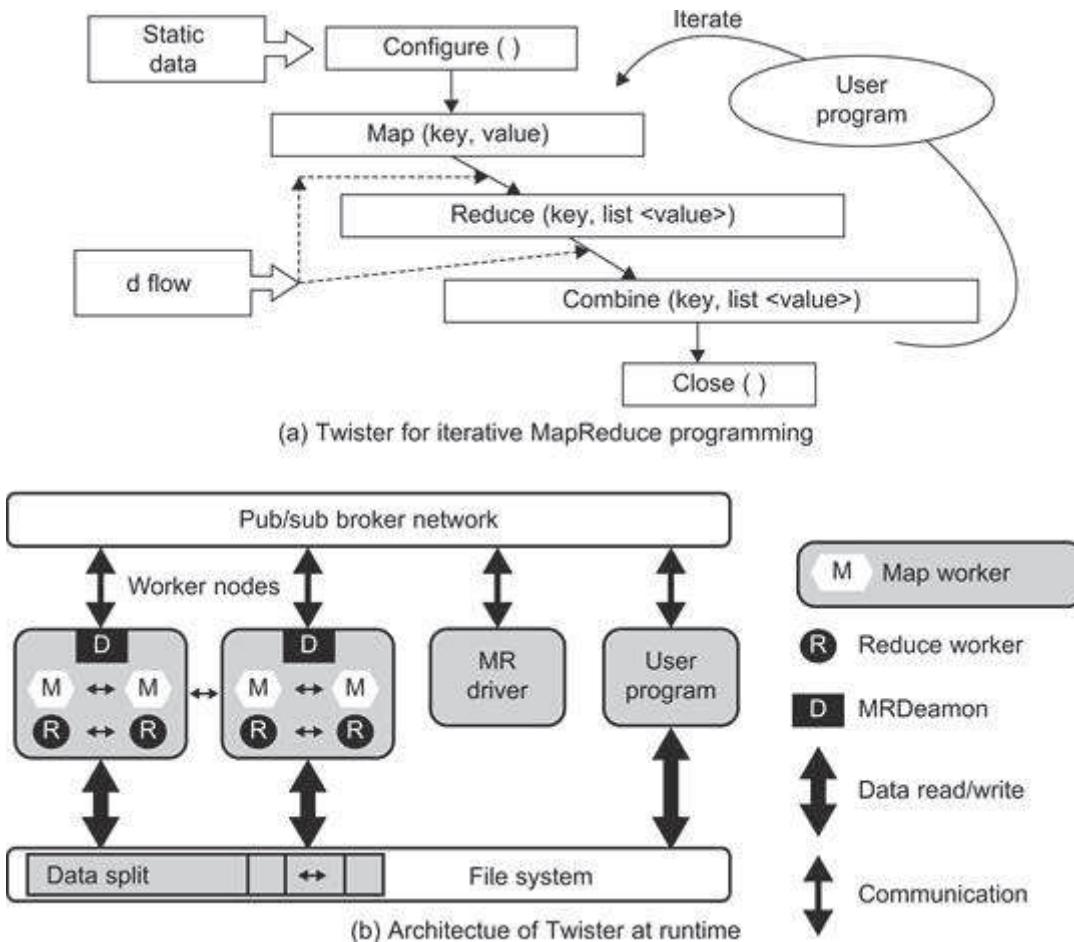
- MapReduce reads and writes via files, whereas MPI transfers information directly between nodes over the network.
- MPI does not transfer all data from node to node, but just the amount needed to update information. We can call the MPI flow  $\delta$  flow and the MapReduce flow *full data flow*.

The same phenomenon is seen in all “classic parallel” loosely synchronous applications which typically exhibit an iteration structure over compute phases followed by communication phases. We can address the performance issues with two important changes:

1. Stream information between steps without writing intermediate steps to disk.
2. Use long-running threads or processors to communicate the  $\delta$  (between iterations) flow.

These changes will lead to major performance increases at the cost of poorer fault tolerance and ease to support dynamic changes such as the number of available nodes. This concept [42] has been investigated in several projects [34,57–59] while the direct idea of using MPI for MapReduce applications is investigated in [44]. The Twister programming paradigm and its implementation architecture at run time are illustrated in Figure 6.7(a, b). In Example 6.1, we summarize Twister [60] whose performance results for K means are shown in Figure 6.8 [55,56], where Twister is much faster than traditional

MapReduce. Twister distinguishes the static data which is never reloaded from the dynamic  $\delta$  flow that is communicated.

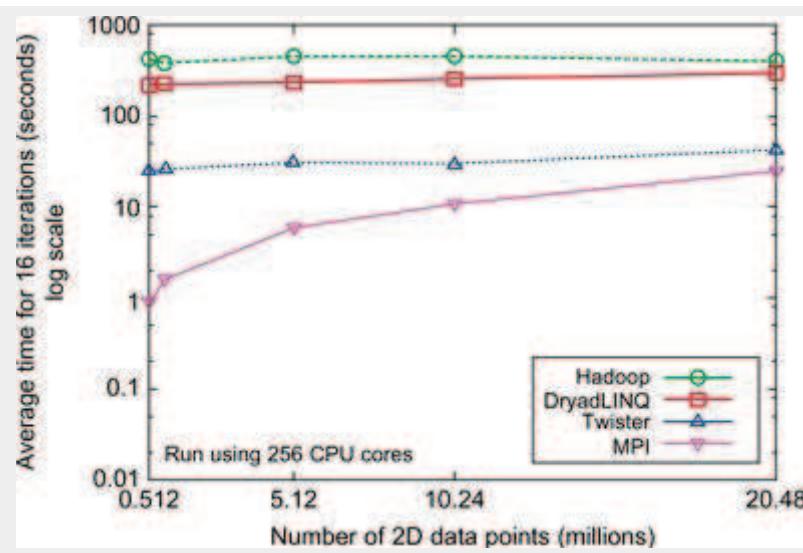


**FIGURE 6.7** Twister: An iterative MapReduce programming paradigm for repeated MapReduce executions.

### Example 6.1

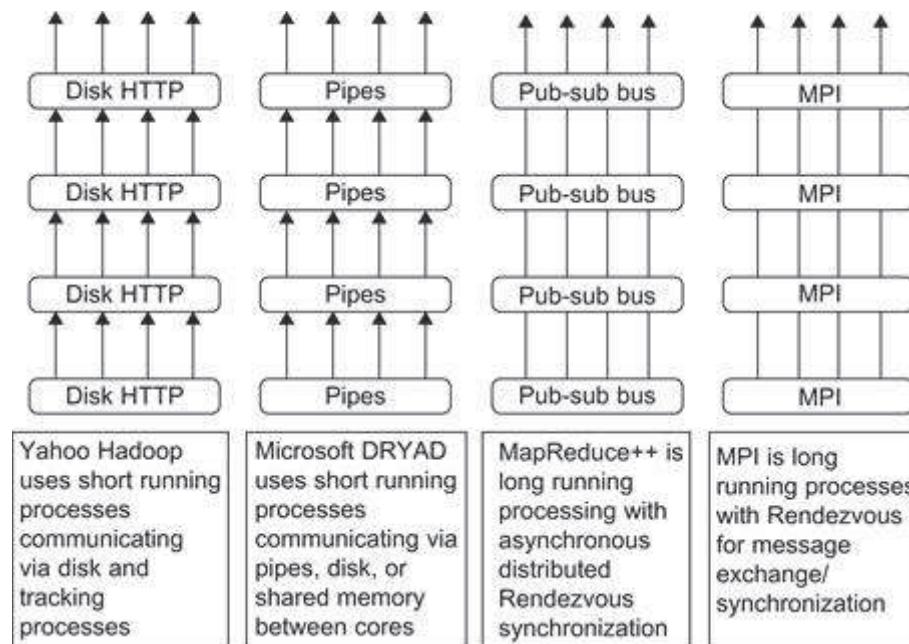
## Performance of K Means Clustering in MPI, Twister, Hadoop, and DryadLINQ

The MapReduce approach leads to fault tolerance and flexible scheduling, but for some applications the performance degradation compared to MPI is serious, as illustrated in [Figure 6.8](#) for a simple parallel K means clustering algorithm. Hadoop and DryadLINQ are more than a factor of 10 slower than MPI for the largest data set, and perform even more poorly for smaller data sets. One could use many communication mechanisms in Iterative MapReduce, but Twister chose a publish-subscribe network using a distributed set of brokers, as described in [Section 5.2](#) with similar performance achieved with ActiveMQ and NaradaBrokering.



**FIGURE 6.8** Performance of K means clustering for MPI, Twister, Hadoop, and DryadLINQ.

The Map-Reduce pair is iteratively executed in long-running threads. We compare in Figure 6.9. the different thread and process structures of 4 parallel programming paradigms: namely Hadoop, Dryad, Twister (also called MapReduce++), and MPI. Note that Dryad can use pipes and avoids costly disk writing according to the original papers [26,27].

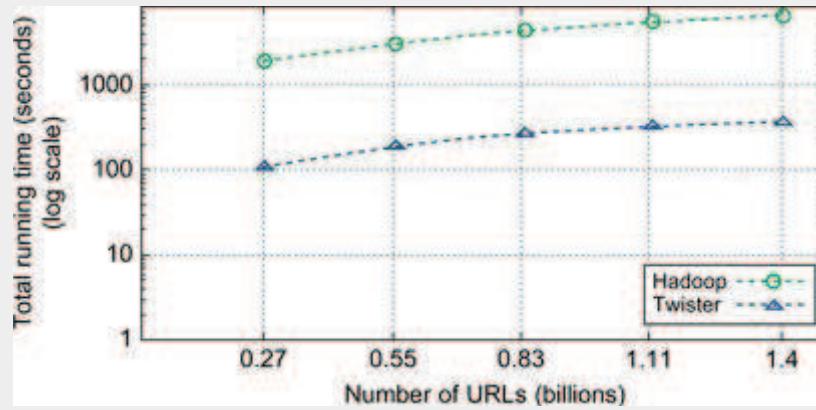


**FIGURE 6.9** Thread and process structure of four parallel programming paradigms at runtimes.

## Example 6.2

### Performance of Hadoop and Twister on ClueWeb Data Set over 256 Processor Cores

Important research areas for Iterative MapReduce include fault tolerance and scalable approaches to communication. Figure 6.10 shows [55] that iterative algorithms are found in information retrieval. This figure shows the famous Page Rank algorithm (with a kernel of iterative matrix vector multiplication) run on the public ClueWeb data sets, and independent of size, Twister is about 20 times faster than Hadoop as revealed by the gap between the top and lower curves in Figure 6.10.



**FIGURE 6.10** Performance of Hadoop and Twister on ClueWeb data set using 256 processing cores.

### 6.2.3 Hadoop Library from Apache

Hadoop is an open source implementation of MapReduce coded and released in Java (rather than C) by Apache. The Hadoop implementation of MapReduce uses the *Hadoop Distributed File System (HDFS)* as its underlying layer rather than GFS. The Hadoop core is divided into two fundamental layers: the MapReduce engine and HDFS. The MapReduce engine is the computation engine running on top of HDFS as its data storage manager. The following two sections cover the details of these two fundamental layers.

**HDFS:** HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.

**HDFS Architecture:** HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves). To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes). The mapping of blocks to DataNodes is

determined by the NameNode. The NameNode (master) also manages the file system's metadata and namespace. In such systems, the namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall management of all files. For example, NameNode in the metadata stores all information regarding the location of input splits/blocks in all DataNodes. Each DataNode, usually one per node in a cluster, manages the storage attached to the node. Each DataNode is responsible for storing and retrieving its file blocks [61].

**HDFS Features:** Distributed file systems have special requirements, such as performance, scalability, concurrency control, fault tolerance, and security requirements [62], to operate efficiently. However, because HDFS is not a general-purpose file system, as it only executes specific types of applications, it does not need all the requirements of a general distributed file system. For example, security has never been supported for HDFS systems. The following discussion highlights two important characteristics of HDFS to distinguish it from other generic distributed file systems [63].

**HDFS Fault Tolerance:** One of the main aspects of HDFS is its fault tolerance characteristic. Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception. Therefore, Hadoop considers the following issues to fulfill reliability requirements of the file system [64]:

- **Block replication** To reliably store data in HDFS, file blocks are replicated in this system. In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.
- **Replica placement** The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS. Although storing replicas on different nodes (DataNodes) located in different racks across the whole cluster provides more reliability, it is sometimes ignored as the cost of communication between two nodes in different racks is relatively high in comparison with that of different nodes located in the same rack. Therefore, sometimes HDFS compromises its reliability to achieve lower communication costs. For example, for the default replication factor of three, HDFS stores one replica in the same node the original data is stored, one replica on a different node but in the same rack, and one replica on a different node in a different rack to provide three copies of the data [65].
- **Heartbeat and Blockreport messages** Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly, while each Blockreport contains a list of all blocks on a DataNode [65]. The NameNode receives such messages because it is the sole decision maker of all replicas in the system.

**HDFS High-Throughput Access to Large Data Sets (Files):** Because HDFS is primarily designed for batch processing rather than interactive processing, data access

throughput in HDFS is more important than latency. Also, because applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64 MB) to allow HDFS to decrease the amount of metadata storage required per file. This provides two advantages: The list of blocks per file will shrink as the size of individual blocks increases, and by keeping large amounts of data sequentially within a block, HDFS provides fast streaming reads of data.

**HDFS Operation:** The control flow of HDFS operations such as write and read can properly highlight roles of the NameNode and DataNodes in the managing operations. In this section, the control flow of the main operations of HDFS on files is further described to manifest the interaction between the user, the NameNode, and the DataNodes in such systems [63].

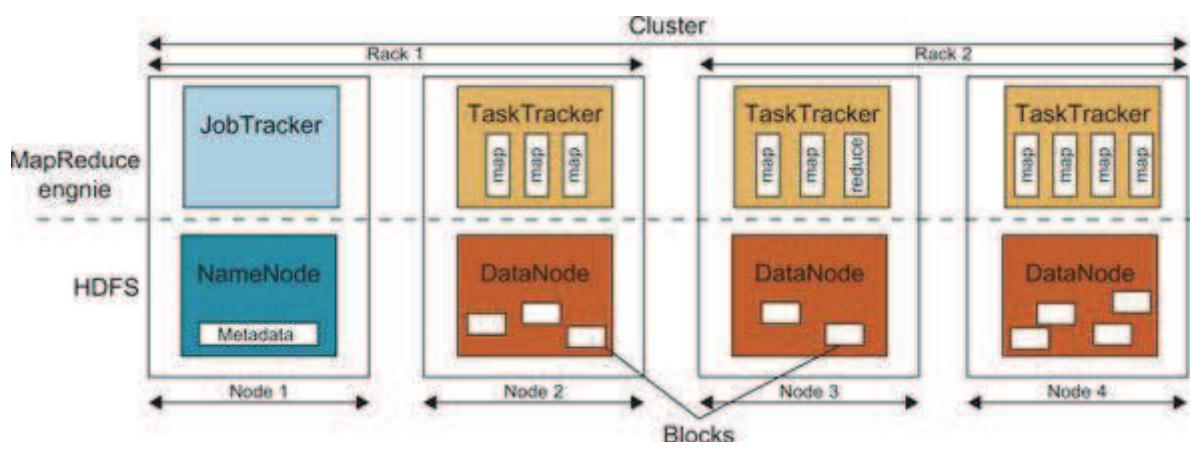
- **Reading a file** To read a file in HDFS, a user sends an “open” request to the NameNode to get the location of file blocks. For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. The number of addresses depends on the number of block replicas. Upon receiving such information, the user calls the *read* function to connect to the closest DataNode containing the first block of the file. After the first block is streamed from the respective DataNode to the user, the established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.

- **Writing to a file** To write a file in HDFS, a user sends a “create” request to the NameNode to create a new file in the file system namespace. If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the *write* function. The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode. Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.

The steamer then stores the block in the first allocated DataNode. Afterward, the block is forwarded to the second DataNode by the first DataNode. The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode. Once this replication process is finalized, the same process starts for the second block and continues until all blocks of the file are stored and replicated on the file system.

### 6.2.3.1 Architecture of MapReduce in Hadoop

The topmost layer of Hadoop is the MapReduce engine that manages the data flow and control flow of MapReduce jobs over distributed computing systems. Figure 6.11 shows the MapReduce engine architecture cooperating with HDFS. Similar to HDFS, the MapReduce engine also has a master/slave architecture consisting of a single JobTracker as the master and a number of TaskTrackers as the slaves (workers). The JobTracker manages the MapReduce job over a cluster and is responsible for monitoring jobs and assigning tasks to TaskTrackers. The TaskTracker manages the execution of the map and/or reduce tasks on a single computation node in the cluster.

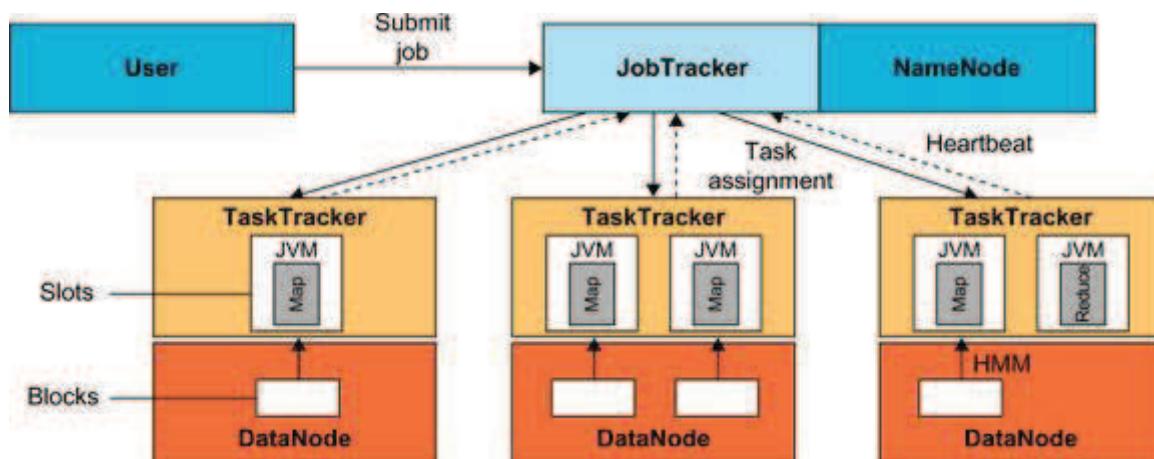


**FIGURE 6.11** HDFS and MapReduce architecture in Hadoop where boxes with different shadings refer to different functional nodes applied to different blocks of data.

Each TaskTracker node has a number of simultaneous execution slots, each executing either a map or a reduce task. Slots are defined as the number of simultaneous threads supported by CPUs of the TaskTracker node. For example, a TaskTracker node with  $N$  CPUs, each supporting  $M$  threads, has  $M * N$  simultaneous execution slots [66]. It is worth noting that each data block is processed by one map task running on a single slot. Therefore, there is a one-to-one correspondence between map tasks in a TaskTracker and data blocks in the respective DataNode.

### 6.2.3.2 Running a Job in Hadoop

Three components contribute in running a job in this system: a user node, a JobTracker, and several TaskTrackers. The data flow starts by calling the *runJob(conf)* function inside a user program running on the user node, in which *conf* is an object containing some tuning parameters for the MapReduce framework and HDFS. The *runJob(conf)* function and *conf* are comparable to the *MapReduce(Spec, &Results)* function and *Spec* in the first implementation of MapReduce by Google. Figure 6.12 depicts the data flow of running a MapReduce job in Hadoop [63].



**FIGURE 6.12** Data flow in running a MapReduce job at various task trackers using the Hadoop library.

- **Job Submission** Each job is submitted from a user node to the JobTracker node that

might be situated in a different node within the cluster through the following procedure:

- A user node asks for a new job ID from the JobTracker and computes input file splits.
- The user node copies some resources, such as the job's JAR file, configuration file, and computed input splits, to the JobTracker's file system.
- The user node submits the job to the JobTracker by calling the *submitJob()* function.
- **Task assignment** The JobTracker creates one map task for each computed input split by the user node and assigns the map tasks to the execution slots of the TaskTrackers. The JobTracker considers the localization of the data when assigning the map tasks to the TaskTrackers. The JobTracker also creates reduce tasks and assigns them to the TaskTrackers. The number of reduce tasks is predetermined by the user, and there is no locality consideration in assigning them.
- **Task execution** The control flow to execute a task (either map or reduce) starts inside the TaskTracker by copying the job JAR file to its file system. Instructions inside the job JAR file are executed after launching a Java Virtual Machine (JVM) to run its map or reduce task.
- **Task running check** A task running check is performed by receiving periodic heartbeat messages to the JobTracker from the TaskTrackers. Each heartbeat notifies the JobTracker that the sending TaskTracker is alive, and whether the sending TaskTracker is ready to run a new task.

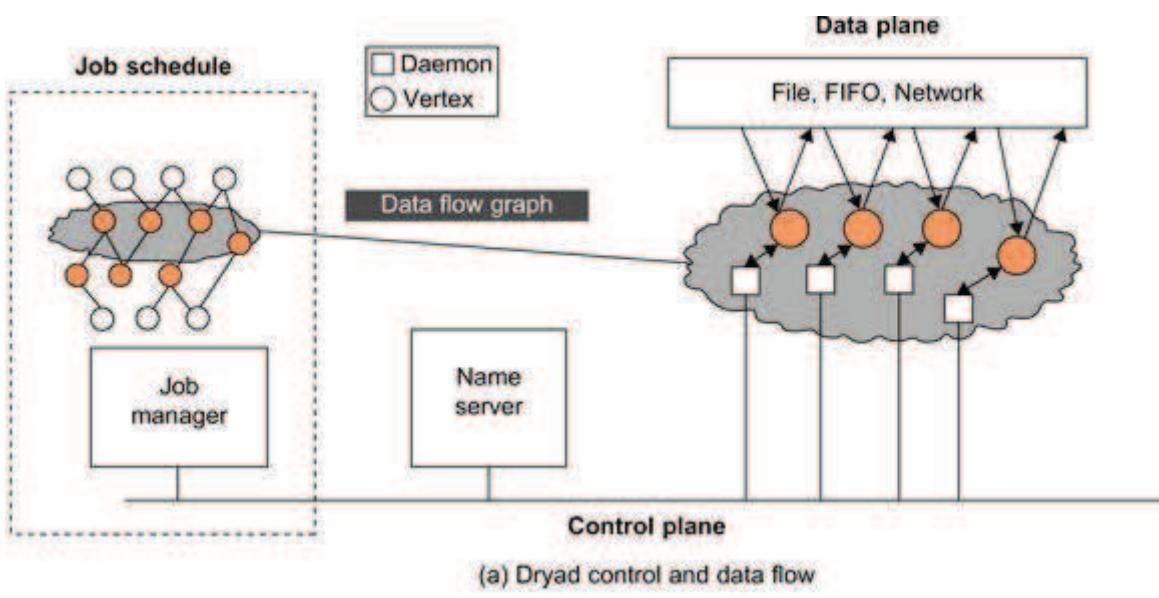
## 6.2.4 Dryad and DryadLINQ from Microsoft

Two runtime software environments are reviewed in this section for parallel and distributed computing, namely the Dryad and DryadLINQ, both developed by Microsoft.

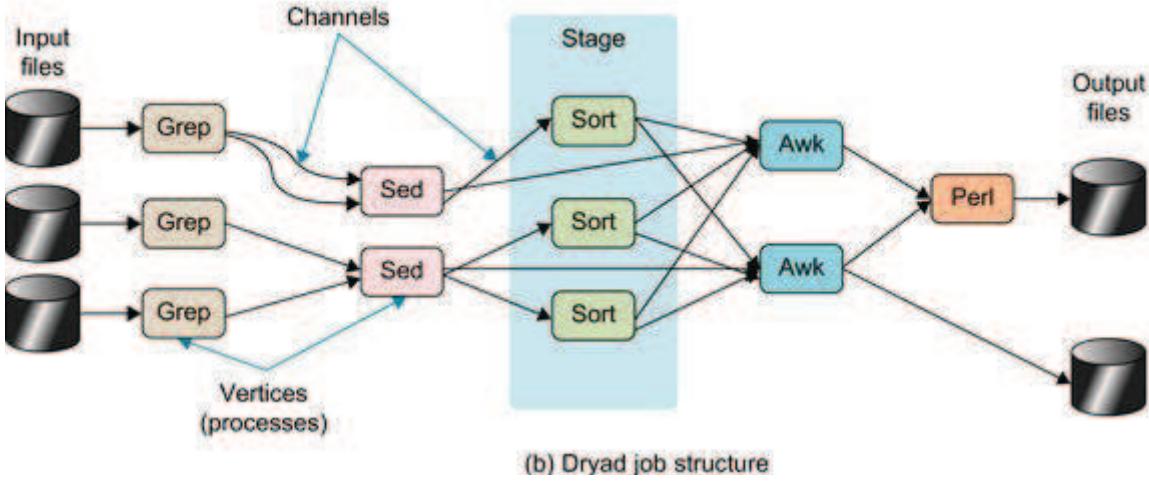
### 6.2.4.1 Dryad

Dryad is more flexible than MapReduce as the data flow of its applications is not dictated/predetermined and can be easily defined by users. To achieve such flexibility, a Dryad program or job is defined by a *directed acyclic graph* (DAG) where vertices are computation engines and edges are communication channels between vertices. Therefore, users or application developers can easily specify arbitrary DAGs to specify data flows in jobs.

Given a DAG, Dryad assigns the computational vertices to the underlying computation engines (cluster nodes) and controls the data flow through edges (communication between cluster nodes). Data partitioning, scheduling, mapping, synchronization, communication, and fault tolerance are major implementation details hidden by Dryad to facilitate its programming environment. Because the data flow of a job is arbitrary for this system, only the control flow of the runtime environment is further explained here. As shown in [Figure 6.13\(a\)](#), the two main components handling the control flow of Dryad are the job manager and the name server.



(a) Dryad control and data flow



(b) Dryad job structure

**FIGURE 6.13** Dryad framework and its job structure, control and data flow. Courtesy of Isard, et al., ACM SIGOPS Operating Systems Review, 2007 [26]

In Dryad, the distributed job is represented as a DAG where each vertex is a program and edges represent data channels. Thus, the whole job will be constructed by the application programmer who defines the processing procedures as well as the flow of data. This logical computation graph will be automatically mapped onto the physical nodes by the Dryad runtime. A Dryad job is controlled by the job manager, which is responsible for deploying the program to the multiple nodes in the cluster. It runs either within the computing cluster or as a process in the user's workstation which can access the cluster. The job manager has the code to construct the DAG as well as the library to schedule the work running on top of the available resources. Data transfer is done via channels without involving the job manager. Thus, the job manager should not be the performance bottleneck. In summary, the job manager

1. Constructs a job's communication graph (data flow graph) using the application-specific program provided by the user.
2. Collects the information required to map the data flow graph to the underlying resources (computation engine) from the name server.

The cluster has a name server which is used to enumerate all the available computing resources in the cluster. Thus, the job manager can contact the name server to get the

topology of the whole cluster and make scheduling decisions. A *processing daemon* runs in each computing node in the cluster. The binary of the program will be sent to the corresponding processing node directly from the job manager. The daemon can be viewed as a proxy so that the job manager can communicate with the remote vertices and monitor the state of the computation. By gathering this information, the name server provides the job manager with a perfect view of the underlying resources and network topology. Therefore, the job manager is able to:

1. Map the data flow graph to the underlying resources.
2. Schedule all necessary communications and synchronization *across* the respective resources.

It also considers data and computation locality when mapping the data flow graph to the underlying resources [26]. When the data flow graph is mapped on a set of computation engines, a light daemon runs on each cluster node to run the assigned tasks. Each task is defined by the user using an application-specific program. During runtime, the job manager communicates with each daemon to monitor the state of the computation of the node and its communication with its preceding and succeeding nodes. At runtime, the channels are used to transport the structured items between vertices which represent the processing programs. There are several types of communication mechanisms for implementing channels such as shared memory, TCP sockets, or even distributed file systems.

The execution of a Dryad job can be considered a 2D distributed set of pipes. Traditional UNIX pipes are 1D pipes, with each node in the pipe as a single program. Dryad's 2D distributed pipe system has multiple processing programs in each vertex node. In this way, large-scale data can be processed simultaneously. Figure 6.13(b) shows the Dryad 2D pipe job structure. During 2D pipe execution, Dryad defines many operations to construct and change the DAG dynamically. The operations include creating new vertices, adding graph edges, merging two graphs, as well as handling job input and output. Dryad also has a fault-tolerant mechanism built in. As it is built on a DAG, there are typically two types of failures: vertex failures and channel failures, which are handled differently.

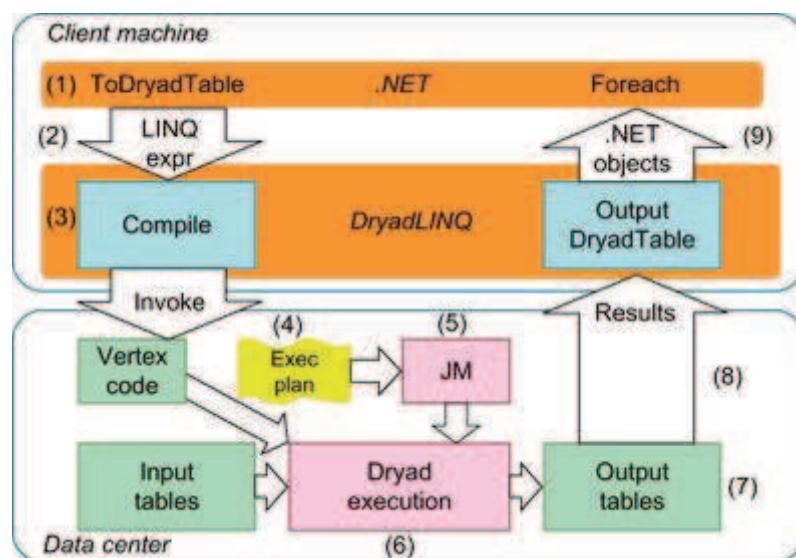
As there are many nodes in the cluster, the job manager can choose another node to re-execute the corresponding job assigned to the failed node. In case of an edge failure, the vertex that created the channel will be re-executed and a new channel will be created and touch the corresponding nodes again. Dryad provides other mechanisms in addition to the runtime graph refinements which are used for improving execution performance. As a general framework, Dryad can be used in many situations, including scripting language support, map-reduce programming, and SQL service integration.

#### **6.2.4.2 DryadLINQ from Microsoft**

DryadLINQ is built on top of Microsoft's Dryad execution framework (see <http://research.microsoft.com/en-us/projects/DryadLINQ/>). Dryad can perform acyclic task scheduling and run on large-scale servers. The goal of DryadLINQ is to make large-scale, distributed cluster computing available to ordinary programmers. Actually,

DryadLINQ, as the name implies, combines two important components: the Dryad distributed execution engine and .NET Language Integrated Query (LINQ). LINQ is particularly for users familiar with a database programming model. Figure 6.14 shows the flow of execution with DryadLINQ. The execution is divided into nine steps as follows:

1. A .NET user application runs, and creates a DryadLINQ expression object. Because of LINQ's deferred evaluation, the actual execution of the expression has not occurred.
2. The application calls *ToDryadTable* triggering a data-parallel execution. The expression object is handed to *DryadLINQ*.
3. DryadLINQ compiles the LINQ expression into a distributed Dryad execution plan. The expression is decomposed into subexpressions, each to be run in a separate Dryad vertex. Code and static data for the remote Dryad vertices are generated, followed by the serialization code for the required data types.
4. DryadLINQ invokes a custom Dryad job manager which is used to manage and monitor the execution flow of the corresponding task.
5. The job manager creates the job graph using the plan created in step 3. It schedules and spawns the vertices as resources become available.
6. Each Dryad vertex executes a vertex-specific program.
7. When the Dryad job completes successfully it writes the data to the out table(s).
8. The job manager process terminates, and it returns control back to DryadLINQ. DryadLINQ creates the local *DryadTable* objects encapsulating the output of the execution. The *DryadTable* objects here might be the input to the next phase.
9. Control returns to the user application. The iterator interface over a *DryadTable* allows the user to read its contents as .NET objects.



**FIGURE 6.14** LINQ-expression execution in DryadLINQ. Courtesy of Yu, et al. [27]

Not all programs go through all nine steps. Some programs may go through fewer steps. Based on the preceding description, DryadLINQ enables users to integrate their current programming language (C#) with a compiler and a runtime execution engine. The following example shows how to write a histogram in DryadLINQ.

## Example 6.3

### A Histogram Written in DryadLINQ

We show a histogram program written in DryadLINQ for counting the frequency of each word in a text file. The entire program is listed here for readers to trace through to familiarize with the high-level language for Dryad programming. For details, the reader is referred to the article that introduces by Yu, et al. [27].

```
[Serializable]

public struct Pair {

    string word;
    int count;
    public Pair(string w, int c)
    {
        word = w;
        count = c;
    }
    public override string ToString() {
        return word + ":" + count.ToString();
    }
}

public static IQueryable<Pair> Histogram(
    string directory,
    string filename,
    int k)
{

    DryadDataContext ddc = new DryadDataContext("file://" + directory);
    DryadTable<LineRecord> table = ddc.GetPartitionedTable<LineRecord>(filename);
    IQueryable<string> words = table.SelectMany(x => x.line.Split(' '))
        .AsEnumerable();
    IQueryable<IGrouping<string, string>> groups = words.GroupBy(x => x);
    IQueryable<Pair> counts = groups.Select(x => new Pair(x.Key, x.Count()));
    IQueryable<Pair> ordered = counts.OrderByDescending(x => x.Count);
    IQueryable<Pair> top = ordered.Take(k);
```

```
return top;
```

```
}
```

## Example 6.4

### Sample Execution of a Histogram of the Word-Count Problem

The execution flow of such programs is very similar to the word-count program in the MapReduce framework. Table 6.6 shows how this program can be executed on a sample text input. Each row in the table presents the execution effect of each code line in the program. The execution of the program will go through all of the steps that a typical DryadLINQ program will involve.

**Table 6.6**

**Sample Execution of Histogram**

Operator	Output
Table	"A line of words of wisdom"
SelectMany	[“A”, “line”, “of”, “words”, “of”, “wisdom”]
GroupBy	[[“A”], [“line”], [“of”, “of”], [“words”], [“wisdom”]]
Select	[{“A”, 1}, {“line”, 1}, {“of”, 2}, {“words”, 1}, {“wisdom”, 1}]
OrderByDescending	[{“of”, 2}, {“A”, 1}, {“line”, 1}, {“words”, 1}, {“wisdom”, 1}]
Take(3)	[{“of”, 2}, {“A”, 1}, {“line”, 1}]

Programmers do not have to worry about parallel execution of the program or consider fault tolerance. Scalability, reliability, and some other difficult issues involved in distributed systems are all hidden inside the DryadLINQ framework, and the programs are more concerned with the application program logic. This can greatly reduce the level of programming skill required for parallel data process programming.

## Example 6.5

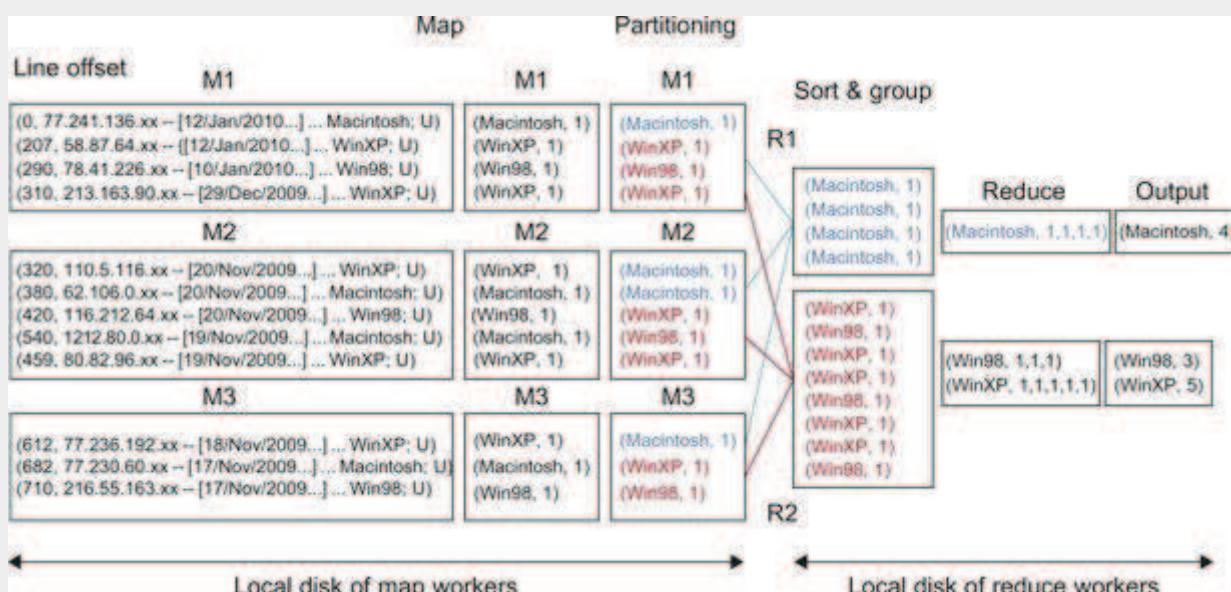
### Hadoop Implementation of a MapReduce WebVisCounter Program

In this example, we present a practical MapReduce program coded in

Hadoop. Called WebVisCounter, this sample program counts the number of times users connect to or visit a given web site using a particular operating system (e.g., Windows XP or Linux Ubuntu). The input data is shown here. A single line of a typical web server log file has eight fields separated by tabs or spaces with the following meanings:

1. **176.123.143.12** (IP address of the machine connected)
  2. – (a separator)
  3. **[10/Sep/2010:01:11:30-1100]** (A timestamp of the visit in the format: DD:Mon:YYYY HH:MM:SS, and the -11:00 is the offset from Greenwich Mean Time)
  4. **"GET /gse/apply/int\_research\_app\_form.pdf HTTP/1.0"** (GET requests the file, /gse/apply/int\_research\_app\_form.pdf, using the HTTP/1.0 protocol)
  5. **200** (The status code to reflect success of the user's request)
  6. **1363148** (The number of bytes transferred)
  7. **"http://www.eng.usyd.edu.au"** (User starts here before reaching server)
  8. **"Mozilla/4.7[en](WinXp; U)"** (The browser used to get to the web site, the version of the browser, the language version, and the operating system)

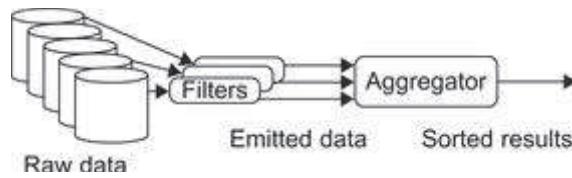
Because the output of interest is the number of times users connect to a given web site using a particular operating system, the *Map* function parses each line to extract the type of the operating system used (e.g., *WinXp*) as a key and assigns a value (1 in this case) to it. The *Reduce* function in turn sums up the number of 1s for each unique key (operating system type in this case). Figure 6.15 shows the associated data flow for the WebVisCounter program.



**FIGURE 6.15** Data flow in WebVisCounter program execution.

## 6.2.5 Sawzall and Pig Latin High-Level Languages

Sawzall is a high-level language [31] built on top of Google's MapReduce framework. Sawzall is a scripting language that can do parallel data processing. As with MapReduce, Sawzall can do distributed, fault-tolerant processing of very large-scale data sets, even at the scale of the data collected from the entire Internet. Sawzall was developed by Rob Pike with an initial goal of processing Google's log files. In this regard it was hugely successful and changed a batch daylong enterprise into an interactive session, enabling new approaches to using such data to be developed. Figure 6.16 shows the overall model of data flow and processing procedures in the Sawzall framework. Sawzall has recently been released as an open source project.



**FIGURE 6.16** The overall flow of filtering, aggregating, and collating in Sawzall.

First the data is partitioned and processed locally with an on-site processing script. The local data is filtered to get the necessary information. The aggregator is used to get the final results based on the emitted data. Many of Google's applications fit this model. Users write their applications using the Sawzall scripting language. The Sawzall runtime engine translates the corresponding scripts to MapReduce programs running on many nodes. The Sawzall program can harness the power of cluster computing automatically as well as attain reliability from redundant servers.

### Example 6.6

#### The File Summary Program in Sawzall

Here is a simple example of using Sawzall for data processing on clusters (the example is from the Sawzall paper at [31]). Suppose we want to process a set of files with records in each file. Each record contains one floating-point number. We want to calculate the number of records, the sum of the values, and the sum of the squares of the values. The relevant code is as follows:

```
count: table sum of int;  
total: table sum of float;  
sum_of_squares: table sum of float;  
x: float = input;
```

```

emit count <- 1;

emit total <- x;

emit sum_of_squares <- x * x;

```

The first three lines declare the aggregators as *count*, *total*, and *sum\_of\_squares*. *table* is a keyword which defines an aggregator type. These particular tables are *sum* tables and they will automatically add up the values emitted to them. For each input record, Sawzall initializes the predefined variable *input* to the uninterpreted byte string of the input. The line *x: float = input;* converts the input into a floating-point number and stores it in local variable *x*. The three *emit* statements send the intermediate values to the aggregators. One can translate the Sawzall scripts into MapReduce programs and run them with multiple servers.

### 6.2.5.1 Pig Latin

Pig Latin is a high-level data flow language developed by Yahoo! [33] that has been implemented on top of Hadoop in the Apache Pig project [67]. Pig Latin, Sawzall and DryadLINQ are different approaches to building languages on top of MapReduce and its extensions. They are compared in Table 6.7.

**Table 6.7**

**Comparison of High-Level Data Analysis Languages**

	Sawzall	Pig Latin	DryadLINQ
Origin	Google	Yahoo!	Microsoft
Data Model	Google protocol buffer or basic	Atom, Tuple, Bag, Map	Partition file
Typing	Static	Dynamic	Static
Category	Interpreted	Compiled	Compiled
Programming Style	Imperative	Procedural: sequence of declarative steps	Imperative and declarative
Similarity to SQL	Least	Moderate	A lot!
Extensibility (User-Defined Functions)	No	Yes	Yes
Control Structures	Yes	No	Yes
Execution Model	Record operations + fixed aggregations	Sequence of MapReduce operations	DAGs
Target Runtime	Google MapReduce	Hadoop (Pig)	Dryad

DryadLINQ is building directly on SQL while the other two languages are of NOSQL heritage, although Pig Latin supports major SQL constructs including Join, which is absent in Sawzall. Each language automates the parallelism, so you only think about

manipulation of individual elements and then invoke supported collective operations. This is possible, of course, because needed parallelism can be cleanly implemented by independent tasks with “side effects” only presented in supported collective operations. This is an important general approach to parallelism and was seen, for example, a long time ago in High Performance Fortran [68]. There are several discussions of Pig and Pig Latin in the literature [69,70], and here we summarize the language features. Table 6.8 lists the four data types in Pig Latin and Table 6.9 the 14 operators.

**Table 6.8**  
**Pig Latin Data Types**

Data Type	Description	Example
Atom	Simple atomic value	'Clouds'
Tuple	Sequence of fields of any Pig Latin type	('Clouds', 'Grids')
Bag	Collection of tuples with each member of the bag allowed a different schema	{ ('Clouds', 'Grids') ('Clouds', ('IaaS', 'PaaS')) }
Map	A collection of data items associated with a set of keys; the keys are a bag of atomic data	[ 'Microsoft' → { ('Windows') ('Azure') } 'Redhat' → 'Linux' ]

**Table 6.9**  
**Pig Latin Operators**

Command	Description
LOAD	Read data from the file system.
STORE	Write data to the file system.
FOREACH GENERATE	Apply an expression to each record and output one or more records.
FILTER	Apply a predicate and remove records that do not return true.
GROUP/COGROUP	Collect records with the same key from one or more inputs.
JOIN	Join two or more inputs based on a key.
CROSS	Cross product two or more inputs.
UNION	Merge two or more data sets.
SPLIT	Split data into two or more sets, based on filter conditions.
ORDER	Sort records based on a key.
DISTINCT	Remove duplicate tuples.
STREAM	Send all records through a user-provided binary.
DUMP	Write output to stdout.
LIMIT	Limit the number of records.

## Example 6.7

### Parallel Programming with Sawzall and Pig Latin

First one would read in data with commands such as:

```
Queries = LOAD 'filewithdata.txt' USING myUDF() AS (userId, queryString, tim
```

The *LOAD* command returns a handle to the bag *Queries*. *myUDF()* is an optional custom reader which is an example of a user-defined function. The *AS* syntax defines the schema of the tuples that make up the bag *Queries*. The data can now be processed by commands such as:

```
Expanded_queries = FOREACH queries GENERATE userID, expandQueryUDF(queryStri
```

The example maps each tuple in *queries* as determined by the user-defined function *expandQueryUDF*. *FOREACH* runs over all tuples in the bag. Alternatively, one could use *FILTER* as in the following example to remove all tuples with a *userID* equal to *Alice* as a string:

```
Real_queries = FILTER queries BY userID neq 'Alice';
```

Pig Latin offers the equivalent of a SQL JOIN capability using COGROUP as in

```
Grouped_data = COGROUP results BY queryString, revenue BY queryString;
```

Where results and revenue are bags of tuples (either from LOAD or processing of LOADED data)

*Results*: *queryString, url, position*)

*Revenue*: (*queryString, adSlot, amount*)

COGROUP is more general than JOIN in the sense that COGROUP does not produce a set of tuples (*queryString, url, position, adSlot, amount*), but rather a tuple consisting of three fields. The first field is *querystring*, the second field is all tuples from *Results* with the value of *querystring*, and the third field is all tuples from *Revenue* with the value of *queryString*. FLATTEN can map the result of COGROUP to SQL Join (*queryString, url, position, adSlot, amount*) syntax.

Pig Latin operations are performed in the order listed as a data flow pipeline. This is in contrast to declarative SQL where one just specifies “what” has to be done, not how it is to be done. Pig Latin supports user-defined functions, as illustrated in the preceding code, as first-class operations in the language which could be an advantage over SQL. User-defined functions can be placed in *Load, Store, Group, Filter, and Foreach* operators,

depending on user preference. Note that the rich set of data flow operations allowed in Pig Latin makes it similar to a scripting approach to workflow, as we discussed in [Section 5.5.5](#). The Pig! Apache project [69] maps Pig Latin into a sequence of MapReduce operations implemented in Hadoop.

## 6.2.6 Mapping Applications to Parallel and Distributed Systems

In the past, Fox has discussed mapping applications to different hardware and software in terms of five application architectures [71]. These initial five categories are listed in [Table 6.10](#), followed by a sixth emerging category to describe data-intensive computing [72,73]. The original classifications largely described simulations and were not aimed directly at data analysis. It is instructive to briefly summarize them and then explain the new category. Category 1 was popular 20 years ago, but is no longer significant. It describes applications that can be parallelized with lock-step operations controlled by hardware.

---

**Table 6.10**  
**Application Classification for Parallel and Distributed Systems**

---

Category	Class	Description	Machine Architecture
1	Synchronous	The problem class can be implemented with instruction-level lockstep operation as in SIMD architectures.	SIMD
2	Loosely synchronous (BSP or bulk synchronous processing)	These problems exhibit iterative compute-communication stages with independent compute (map) operations for each CPU that are synchronized with a communication step. This problem class covers many successful MPI applications including partial differential equation solutions and particle dynamics applications.	MIMD on MPP (massively parallel processor)
3	Asynchronous	Illustrated by Compute Chess and Integer Programming; combinatorial search is often supported by dynamic threads. This is rarely important in scientific computing, but it is at the heart of operating systems and concurrency in consumer applications such as Microsoft Word.	Shared memory
4	Pleasingly parallel	Each component is independent. In 1988, Fox estimated this at 20 percent of the total number of applications, but that percentage has grown with the use of grids and data analysis applications including, for example, the Large Hadron Collider analysis for particle physics.	Grids moving to clouds
5	Metaproblems	These are coarse-grained (asynchronous or data flow) combinations of categories 1-4 and 6. This area has also grown in importance and is well supported by grids and described by workflow in Section 3.5.	Grids of clusters
6	MapReduce++ (Twister)	<p>This describes file (database) to file (database) operations which have three subcategories (see also Table 6.11):</p> <ul style="list-style-type: none"> <li>6a) Pleasingly Parallel Map Only (similar to category 4)</li> <li>6b) Map followed by reductions</li> <li>6c) Iterative "Map followed by reductions" (extension of current technologies that supports linear algebra and data mining)</li> </ul>	Data-intensive clouds a) Master-worker or MapReduce b) MapReduce c) Twister

Such a configuration would run on SIMD (*single-instruction and multiple-data*) machines, whereas category 2 is now much more important and corresponds to a SPMD (*single-program and multiple-data*) model running on MIMD (*multiple instruction multiple data*) machines. Here, each decomposed unit executes the same program, but at any given time, there is no requirement that the same instruction be executed. Category 1 corresponds to regular problems, whereas category 2 includes dynamic irregular cases with complex geometries for solving partial differential equations or particle dynamics. Note that synchronous problems are still around, but they are run on MIMD machines with the SPMD model. Also note that category 2 consists of compute-communicate phases and the computations are synchronized by communication. No additional synchronization is needed.

Category 3 consists of asynchronously interacting objects and is often considered the people's view of a typical parallel problem. It probably does describe the concurrent threads in a modern operating system, as well as some important applications, such as event-driven simulations and areas such as search in computer games and graph

algorithms. Shared memory is natural due to the low latency often needed to perform dynamic synchronization. It wasn't clear in the past, but now it appears that this category is not very common in large-scale parallel problems of importance.

Category 4 is the simplest algorithmically, with disconnected parallel components. However, the importance of this category has probably grown since the original 1988 analysis when it was estimated to account for 20 percent of all parallel computing. Both grids and clouds are very natural for this class, which does not need high-performance communication between different nodes.

Category 5 refers to the coarse-grained linkage of different "atomic" problems, and this was fully covered in [Section 5.5](#). This area is clearly common and is expected to grow in importance. Remember the critical observation in [Section 5.5](#) that we use a two-level programming model with the metaproblem (workflow) linkage specified in one fashion and the component problems with approaches such as those in this chapter. Grids or clouds are suitable for metaproblems as coarse-grained decomposition does not usually require stringent performance.

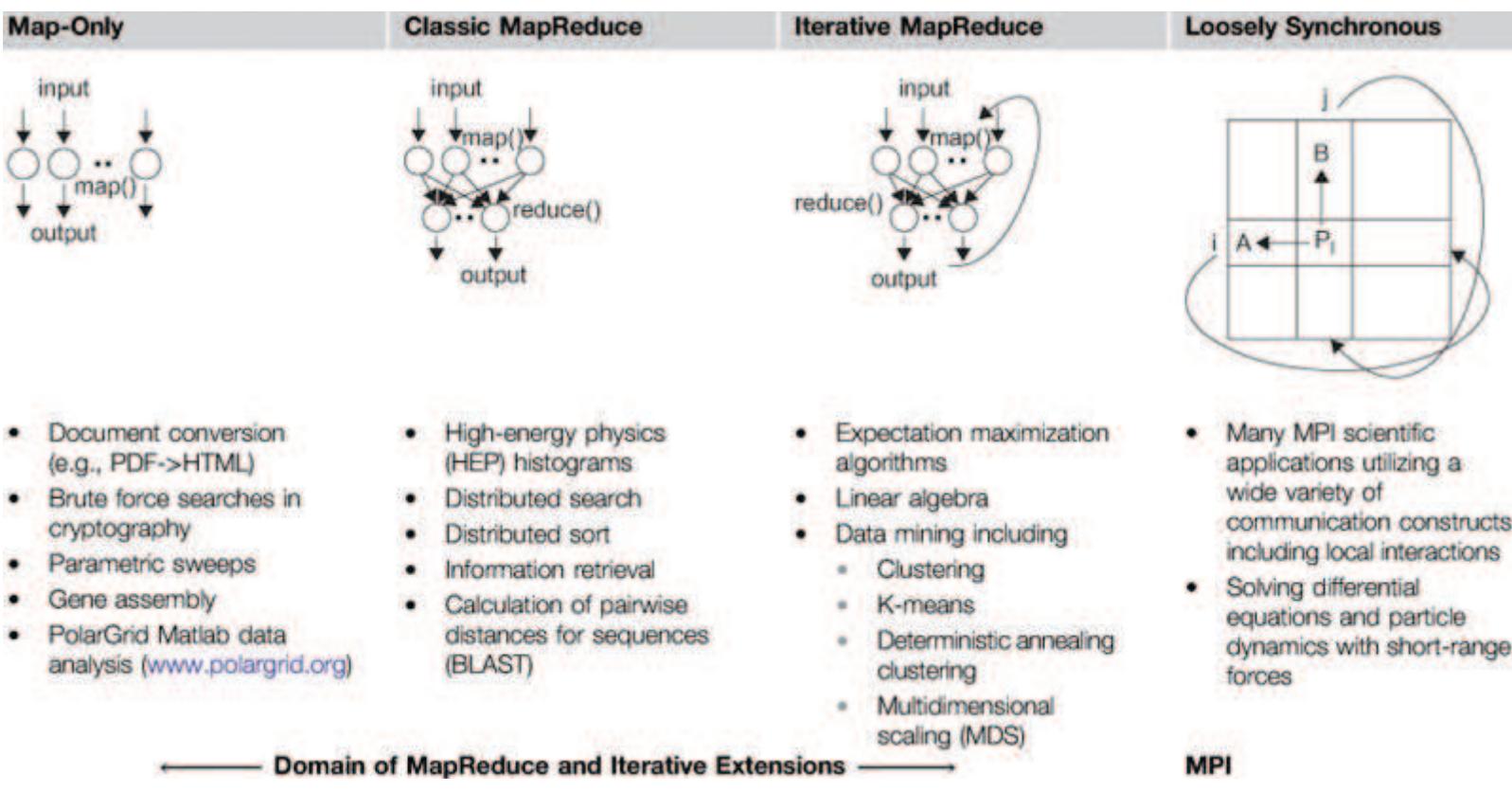
As noted earlier, we added a sixth category to cover data-intensive applications motivated by the clear importance of MapReduce as a new programming model. We call this category MapReduce++, and it has three subcategories: "map only" applications similar to pleasingly parallel category 4 [41,43,74,75]; the classic MapReduce with file-to-file operations consisting of parallel maps followed by parallel reduce operations; and a subcategory that captures the extended MapReduce version covered in [Section 6.2.2](#). Note that category 6 is a subset of categories 2 and 4 with additional reading and writing of data and a specialized, loosely synchronous structure compared to that used in data analysis. This comparison is made clearer in [Table 6.11](#).

---

**Table 6.11**

**Comparison of MapReduce++ Subcategories along with the Loosely Synchronous Category Used in MPI**

---

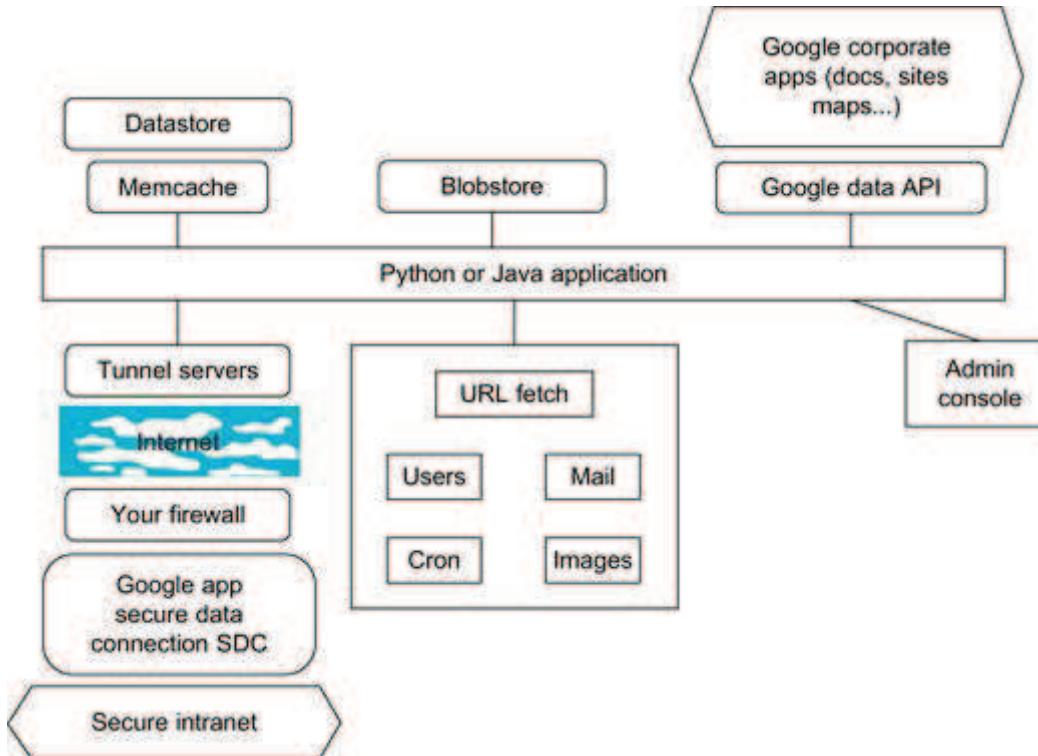


## 6.3 Programming Support of Google App Engine

Section 4.4.2 introduced the Google App Engine infrastructure in Figures 4.20. In this section, we describe the programming model supported by GAE. The access links to the GAE platform are provided in Chapter 4.

### 6.3.1 Programming the Google App Engine

Several web resources (e.g., <http://code.google.com/appengine/>) and specific books and articles (e.g., [www.byteonic.com/2009/overview-of-java-support-in-google-app-engine/](http://www.byteonic.com/2009/overview-of-java-support-in-google-app-engine/)) discuss how to program GAE. Figure 6.17 summarizes some key features of GAE programming model for two supported languages: Java and Python. A client environment that includes an *Eclipse* plug-in for Java allows you to debug your GAE on your local machine. Also, the GWT Google Web Toolkit is available for Java web application developers. Developers can use this, or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. Python is often used with frameworks such as Django and CherryPy, but Google also supplies a built in *webapp* Python environment.



**FIGURE 6.17** Programming environment for Google AppEngine.

There are several powerful constructs for storing and accessing data. The data store is a NOSQL data management system for entities that can be, at most, 1 MB in size and are labeled by a set of schema-less properties. Queries can retrieve entities of a given kind filtered and sorted by the values of the properties. Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the open source Data Nucleus Access platform, while Python has a SQL-like query language called GQL. The data store is strongly consistent and uses optimistic concurrency control.

An update of an entity occurs in a transaction that is retried a fixed number of times if other processes are trying to update the same entity simultaneously. Your application can execute multiple data store operations in a single transaction which either all succeed or all fail together. The data store implements transactions across its distributed network using “entity groups.” A transaction manipulates entities within a single group. Entities of the same group are stored together for efficient execution of transactions. Your GAE application can assign entities to groups when the entities are created. The performance of the data store can be enhanced by in-memory caching using the *memcache*, which can also be used independently of the data store.

Recently, Google added the *blobstore* which is suitable for large files as its size limit is 2 GB. There are several mechanisms for incorporating external resources. The *Google SDC Secure Data Connection* can tunnel through the Internet and link your intranet to an external GAE application. The *URL Fetch* operation provides the ability for applications to fetch resources and communicate with other hosts over the Internet using HTTP and HTTPS requests. There is a specialized mail mechanism to send e-mail from your GAE application.

Applications can access resources on the Internet, such as web services or other data, using GAE’s URL fetch service. The URL fetch service retrieves web resources using the

same high-speed Google infrastructure that retrieves web pages for many other Google products. There are dozens of Google “corporate” facilities including maps, sites, groups, calendar, docs, and YouTube, among others. These support the *Google Data API* which can be used inside GAE.

An application can use Google Accounts for *user* authentication. Google Accounts handles user account creation and sign-in, and a user that already has a Google account (such as a Gmail account) can use that account with your app. GAE provides the ability to manipulate image data using a dedicated *Images* service which can resize, rotate, flip, crop, and enhance images. An application can perform tasks outside of responding to web requests. Your application can perform these tasks on a schedule that you configure, such as on a daily or hourly basis using “cron jobs,” handled by the *Cron* service.

Alternatively, the application can perform tasks added to a queue by the application itself, such as a background task created while handling a request. A GAE application is configured to consume resources up to certain limits or quotas. With quotas, GAE ensures that your application won’t exceed your budget, and that other applications running on GAE won’t impact the performance of your app. In particular, GAE use is free up to certain quotas.

### 6.3.2 Google File System (GFS)

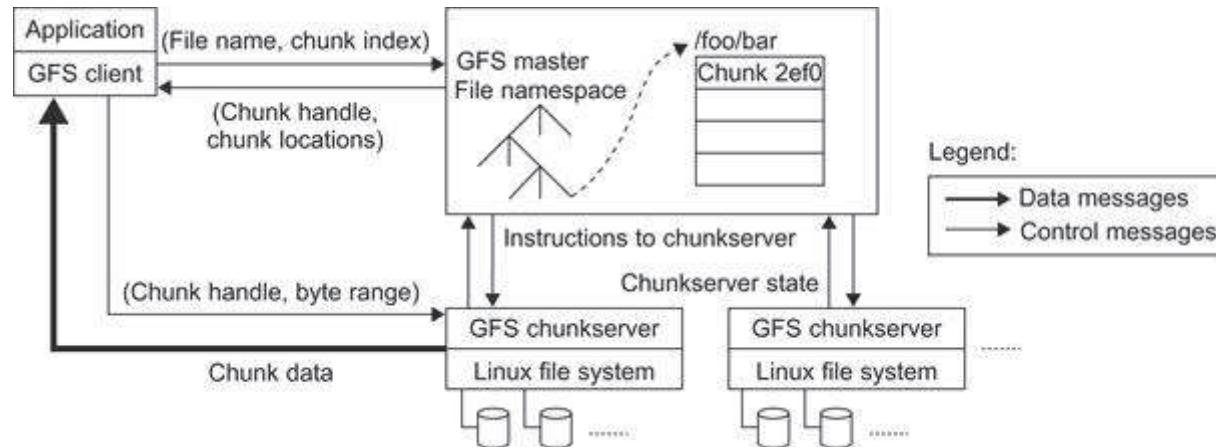
GFS was built primarily as the fundamental storage service for Google’s search engine. As the size of the web data that was crawled and saved was quite substantial, Google needed a distributed file system to redundantly store massive amounts of data on cheap and unreliable computers. None of the traditional distributed file systems can provide such functions and hold such large amounts of data. In addition, GFS was designed for Google applications, and Google applications were built for GFS. In traditional file system design, such a philosophy is not attractive, as there should be a clear interface between applications and the file system, such as a POSIX interface.

There are several assumptions concerning GFS. One is related to the characteristic of the cloud computing hardware infrastructure (i.e., the high component failure rate). As servers are composed of inexpensive commodity components, it is the norm rather than the exception that concurrent failures will occur all the time. Another concerns the file size in GFS. GFS typically will hold a large number of huge files, each 100 MB or larger, with files that are multiple GB in size quite common. Thus, Google has chosen its file data block size to be 64 MB instead of the 4 KB in typical traditional file systems. The I/O pattern in the Google application is also special. Files are typically written once, and the write operations are often the appending data blocks to the end of files. Multiple appending operations might be concurrent. There will be a lot of large streaming reads and only a little random access. As for large streaming reads, highly sustained throughput is much more important than low latency.

Thus, Google made some special decisions regarding the design of GFS. As noted earlier, a 64 MB block size was chosen. Reliability is achieved by using replications (i.e., each chunk or data block of a file is replicated across more than three chunk servers). A

single master coordinates access as well as keeps the metadata. This decision simplified the design and management of the whole cluster. Developers do not need to consider many difficult issues in distributed systems, such as distributed consensus. There is no data cache in GFS as large streaming reads and writes represent neither time nor space locality. GFS provides a similar, but not identical, POSIX file system accessing interface. The distinct difference is that the application can even see the physical location of file blocks. Such a scheme can improve the upper-layer applications. The customized API can simplify the problem and focus on Google applications. The customized API adds snapshot and record append operations to facilitate the building of Google applications.

Figure 6.18 shows the GFS architecture. It is quite obvious that there is a single master in the whole cluster. Other nodes act as the chunk servers for storing data, while the single master stores the metadata. The file system namespace and locking facilities are managed by the master. The master periodically communicates with the chunk servers to collect management information as well as give instructions to the chunk servers to do work such as load balancing or fail recovery.



**FIGURE 6.18** Architecture of Google File System (GFS). Courtesy of S. Ghemawat, et al. [53]

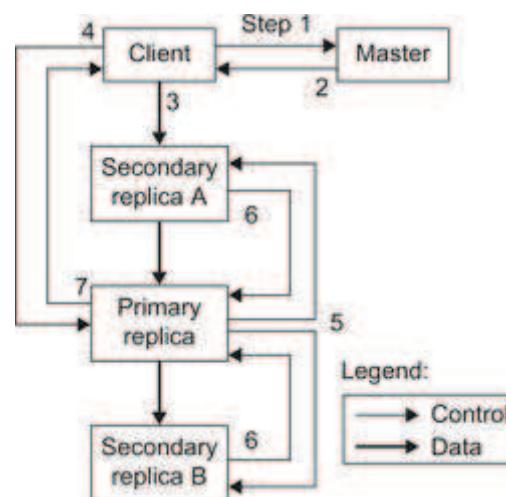
The master has enough information to keep the whole cluster in a healthy state. With a single master, many complicated distributed algorithms can be avoided and the design of the system can be simplified. However, this design does have a potential weakness, as the single GFS master could be the performance bottleneck and the single point of failure. To mitigate this, Google uses a shadow master to replicate all the data on the master, and the design guarantees that all the data operations are performed directly between the client and the chunk server. The control messages are transferred between the master and the clients and they can be cached for future use. With the current quality of commodity servers, the single master can handle a cluster of more than 1,000 nodes.

Figure 6.19 shows the data mutation (write, append operations) in GFS. Data blocks must be created for all replicas. The goal is to minimize involvement of the master. The mutation takes the following steps:

1. The client asks the master which chunk server holds the current lease for the chunk and the locations of the other replicas. If no one has a lease, the master grants one to a

replica it chooses (not shown).

2. The master replies with the identity of the primary and the locations of the other (secondary) replicas. The client caches this data for future mutations. It needs to contact the master again only when the primary becomes unreachable or replies that it no longer holds a lease.
3. The client pushes the data to all the replicas. A client can do so in any order. Each chunk server will store the data in an internal LRU buffer cache until the data is used or aged out. By decoupling the data flow from the control flow, we can improve performance by scheduling the expensive data flow based on the network topology regardless of which chunk server is the primary.
4. Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary. The request identifies the data pushed earlier to all the replicas. The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients, which provides the necessary serialization. It applies the mutation to its own local state in serial order.
5. The primary forwards the write request to all secondary replicas. Each secondary replica applies mutations in the same serial number order assigned by the primary.
6. The secondaries all reply to the primary indicating that they have completed the operation.
7. The primary replies to the client. Any errors encountered at any replicas are reported to the client. In case of errors, the write corrects at the primary and an arbitrary subset of the secondary replicas. The client request is considered to have failed, and the modified region is left in an inconsistent state. Our client code handles such errors by retrying the failed mutation. It will make a few attempts at steps 3 through 7 before falling back to a retry from the beginning of the write.



**FIGURE 6.19** Data mutation sequence in GFS.

Thus, besides the writing operation provided by GFS, special appending operations can be used to append the data blocks to the end of the files. The reason for providing such operations is that some of the Google applications need a lot of append operations.

For example, while crawlers are gathering data from the web, the contents of web pages will be appended to page files. Thus, the appending operation is provided and optimized. The client specifies data to be appended and GFS appends it to the file atomically at least once. GFS picks the offset and the clients cannot decide the offset of the data position. The appending operation works for concurrent writers.

GFS was designed for high fault tolerance and adopted some methods to achieve this goal. Master and chunk servers can be restarted in a few seconds, and with such a fast recovery capability, the window of time in which the data is unavailable can be greatly reduced. As we mentioned before, each chunk is replicated in at least three places and can tolerate at least two data crashes for a single chunk of data. The shadow master handles the failure of the GFS master. For data integrity, GFS makes checksums on every 64 KB block in each chunk. With the previously discussed design and implementation, GFS can achieve the goals of high availability (HA), high performance, and large scale. GFS demonstrates how to support large-scale processing workloads on commodity hardware designed to tolerate frequent component failures optimized for huge files that are mostly appended and read.

### 6.3.3 BigTable, Google's NOSQL System

In this section, we continue discussing key technologies in the Google cloud environment. We already discussed the most well-known Google technology MapReduce, in [Section 6.2.2](#), and Sawzall in [Section 6.2.5](#). Here, we focus on another innovative Google technology: BigTable. We will cover Chubby in [Section 6.3.4](#) and covered GFS in previous section. BigTable was designed to provide a service for storing and retrieving structured and semistructured data. BigTable applications include storage of web pages, per-user data, and geographic locations. Here we use web pages to represent URLs and their associated data, such as contents, crawled metadata, links, anchors, and page rank values. Per-user data has information for a specific user and includes such data as user preference settings, recent queries/search results, and the user's e-mails. Geographic locations are used in Google's well-known Google Earth software. Geographic locations include physical entities (shops, restaurants, etc.), roads, satellite image data, and user annotations.

The scale of such data is incredibly large. There will be billions of URLs, and each URL can have many versions, with an average page size of about 20 KB per version. The user scale is also huge. There are hundreds of millions of users and there will be thousands of queries per second. The same scale occurs in the geographic data, which might consume more than 100 TB of disk space.

It is not possible to solve such a large scale of structured or semistructured data using a commercial database system. This is one reason to rebuild the data management system; the resultant system can be applied across many projects for a low incremental cost. The other motivation for rebuilding the data management system is performance. Low-level storage optimizations help increase performance significantly, which is much harder to do when running on top of a traditional database layer.

The design and implementation of the BigTable system has the following goals. The applications want asynchronous processes to be continuously updating different pieces of data and want access to the most current data at all times. The database needs to support very high read/write rates and the scale might be millions of operations per second. Also, the database needs to support efficient scans over all or interesting subsets of data, as well as efficient joins of large one-to-one and one-to-many data sets. The application may need to examine data changes over time (e.g., contents of a web page over multiple crawls).

Thus, BigTable can be viewed as a distributed multilevel map. It provides a fault-tolerant and persistent database as in a storage service. The BigTable system is scalable, which means the system has thousands of servers, terabytes of in-memory data, petabytes of disk-based data, millions of reads/writes per second, and efficient scans. Also, BigTable is a self-managing system (i.e., servers can be added/removed dynamically and it features automatic load balancing). Design/initial implementation of BigTable began at the beginning of 2004. BigTable is used in many projects, including Google Search, Orkut, and Google Maps/Google Earth, among others. One of the largest BigTable cell manages ~200 TB of data spread over several thousand machines.

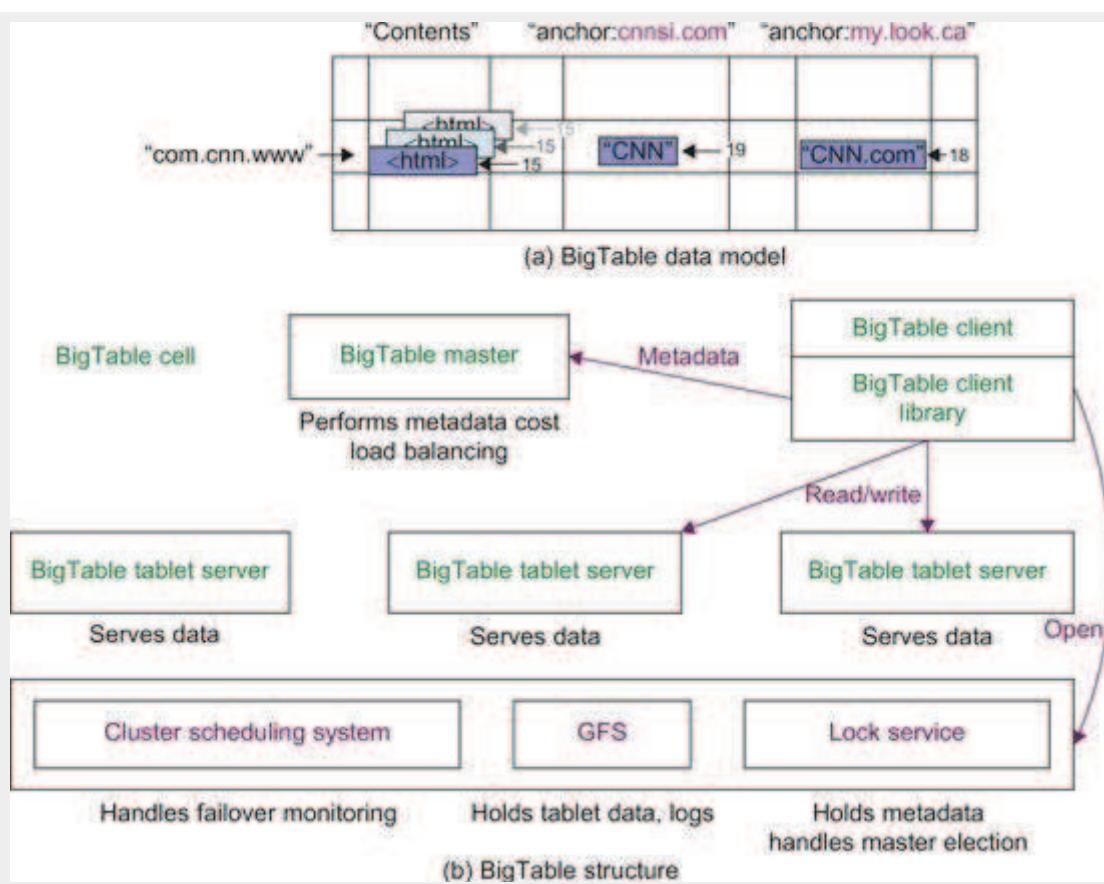
The BigTable system is built on top of an existing Google cloud infrastructure. BigTable uses the following building blocks:

1. GFS: stores persistent state
2. Scheduler: schedules jobs involved in BigTable serving
3. Lock service: master election, location bootstrapping
4. MapReduce: often used to read/write BigTable data

### **Example 6.8**

#### **BigTable Data Model Used in Mass Media**

BigTable provides a simplified data model compared to traditional database systems. [Figure 6.20\(a\)](#) shows the data model of a sample table, Web Table. Web Table stores the data about a web page. Each web page can be accessed by the URL. The URL is considered the row index. The column provides different data related to the corresponding URL—for example, different versions of the contents, and the anchors appearing in the web page. In this sense, BigTable is a distributed multidimensional stored sparse map.



**FIGURE 6.20** BigTable data model and system structure. Courtesy of Chang, et al. [11]

The map is indexed by row key, column key, and timestamp—that is, (row: string, column: string, time:int64) maps to string (cell contents). Rows are ordered in lexicographic order by row key. The row range for a table is dynamically partitioned and each row range is called “Tablet.” Syntax for columns is shown as a (family:qualifier) pair. Cells can store multiple versions of data with timestamps.

Such a data model is a good match for most of Google’s (and other organizations’) applications. For rows, *Name* is an arbitrary string and access to data in a row is atomic. This is different from the traditional relational database which provides abundant atomic operations (transactions). Row creation is implicit upon storing data. Rows are ordered lexicographically, that is, close together lexicographically, usually on one or a small number of machines.

Large tables are broken into tablets at row boundaries. A tablet holds a contiguous range of rows. Clients can often choose row keys to achieve locality. The system aims for about 100 MB to 200 MB of data per tablet. Each serving machine is responsible for about 100 tablets. This can achieve faster recovery times as 100 machines each pick up one tablet from the failed machine. This also results in fine-grained load balancing, that is, migrating tablets away from the overloaded machine. Similar to the design in GFS, a master machine in BigTable makes load-balancing decisions.

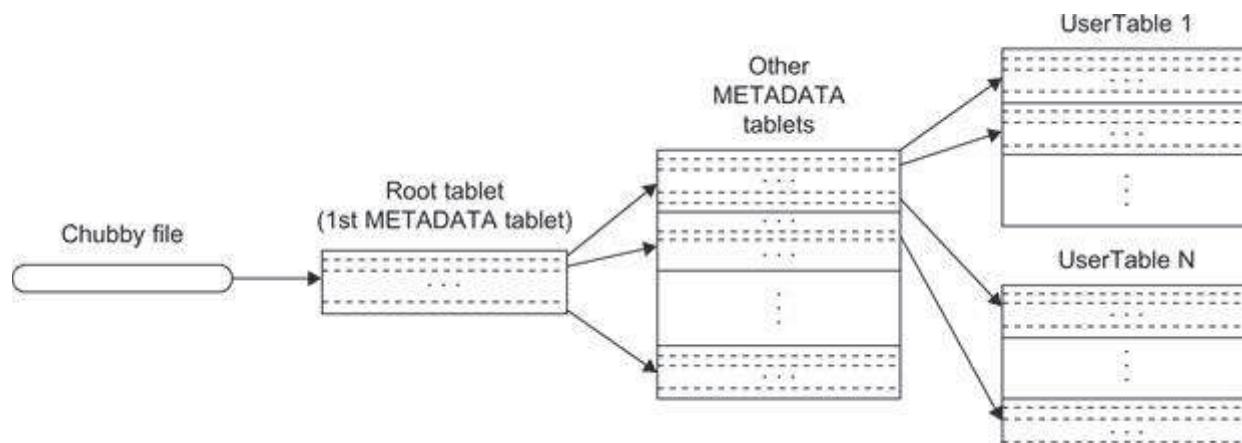
Figure 6.20(b) shows the BigTable system structure. A BigTable master

manages and stores the metadata of the BigTable system. BigTable clients use the BigTable client programming library to communicate with the BigTable master and tablet servers. BigTable relies on a highly available and persistent distributed lock service called Chubby [76] discussed in Section 6.3.4.

### 6.3.3.1 Tablet Location Hierarchy

Figure 6.21 shows how to locate the BigTable data starting from the file stored in Chubby. The first level is a file stored in Chubby that contains the location of the root tablet. The root tablet contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. The root tablet is just the first tablet in the METADATA table, but is treated specially; it is never split to ensure that the tablet location hierarchy has no more than three levels.

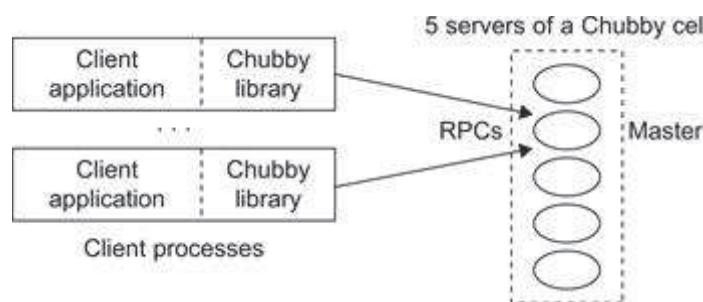
The METADATA table stores the location of a tablet under a row key that is an encoding of the tablet's table identifier and its end row. BigTable includes many optimizations and fault-tolerant features. Chubby can guarantee the availability of the file for finding the root tablet. The BigTable master can quickly scan the tablet servers to determine the status of all nodes. Tablet servers use compaction to store data efficiently. Shared logs are used for logging the operations of multiple tablets so as to reduce the log space as well as keep the system consistent.



**FIGURE 6.21** Tablet location hierarchy in using the BigTable.

## 6.3.4 Chubby, Google's Distributed Lock Service

Chubby [76] is intended to provide a coarse-grained locking service. It can store small files inside Chubby storage which provides a simple namespace as a file system tree. The files stored in Chubby are quite small compared to the huge files in GFS. Based on the Paxos agreement protocol, the Chubby system can be quite reliable despite the failure of any member node. Figure 6.22 shows the overall architecture of the Chubby system.



**FIGURE 6.22** Structure of Google Chubby for distributed lock service.

Each Chubby cell has five servers inside. Each server in the cell has the same file system namespace. Clients use the Chubby library to talk to the servers in the cell. Client applications can perform various file operations on any server in the Chubby cell. Servers run the Paxos protocol to make the whole file system reliable and consistent. Chubby has become Google's primary internal name service. GFS and BigTable use Chubby to elect a primary from redundant replicas.

## 6.4 Programming on Amazon AWS and Microsoft Azure

In this section, we will consider the programming support in the AWS platform. First we will review the AWS platform and its updated service offerings. Then we will study the EC2, S3, and Simple DB services with programming examples. Returning to the programming environment features in Figures 4.22 and 4.23, Amazon (like Azure) offers a Relational Database Service (RDS) with a messaging interface sketched in Section 6.1.3. The Elastic MapReduce capability is equivalent to Hadoop running on the basic EC2 offering. Amazon has NOSQL support in SimpleDB introduced in Section 6.1.3 and discussed in Section 6.4.5. However, Amazon does not directly support BigTable as described in Section 6.3.4.

Now we will highlight a few more capabilities that are listed in Table 4.6. Amazon offers the Simple Queue Service (SQS) and Simple Notification Service (SNS), which are the cloud implementations of services discussed in Sections 5.2 and 5.4.5. Note that brokering systems run very efficiently in clouds and offer a striking model for controlling sensors and giving back-office support for a growing number of smartphones and tablets [77]. We further note the auto-scaling and elastic load balancing services which support related capabilities. Auto-scaling enables you to automatically scale your Amazon EC2 capacity up or down according to conditions that you define. With auto-scaling, you can ensure that the number of Amazon EC2 instances you're using scales up seamlessly during demand spikes to maintain performance, and scales down automatically during demand lulls to minimize cost.

Elastic load balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances and allows you to avoid nonoperating nodes and to equalize load on functioning images. Both auto-scaling and elastic load balancing are enabled by CloudWatch which monitors running instances. CloudWatch is a web service that provides monitoring for AWS cloud resources, starting with Amazon EC2. It provides customers with visibility into resource utilization, operational performance, and

overall demand patterns—including metrics such as CPU utilization, disk reads and writes, and network traffic.

## 6.4.1 Programming on Amazon EC2

Amazon was the first company to introduce VMs in application hosting. Customers can rent VMs instead of physical machines to run their own applications. By using VMs, customers can load any software of their choice. The elastic feature of such a service is that a customer can create, launch, and terminate server instances as needed, paying by the hour for active servers. Amazon provides several types of preinstalled VMs. Instances are often called *Amazon Machine Images (AMIs)* which are preconfigured with operating systems based on Linux or Windows, and additional software.

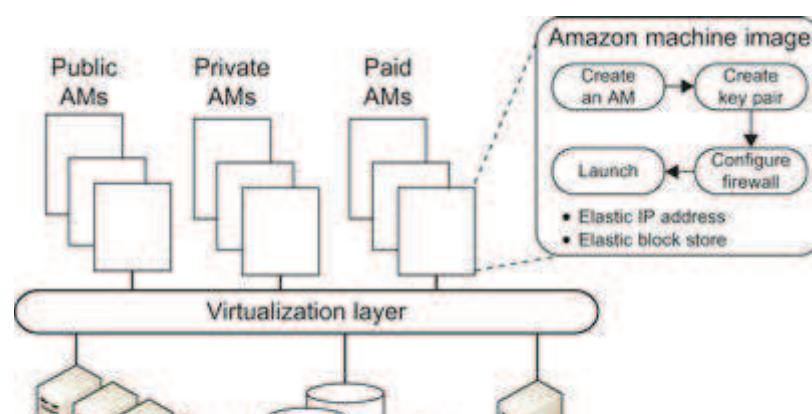
Table 6.12 defines three types of AMI. Figure 6.24 shows an execution environment. AMIs are the templates for instances, which are running VMs. The workflow to create a VM is

(6.3)

**Table 6.12**  
**Three Types of AMI**

Image Type	AMI Definition
Private AMI	Images created by you, which are private by default. You can grant access to other users to launch your private images.
Public AMI	Images created by users and released to the AWS community, so anyone can launch instances based on them and use them any way they like. AWS lists all public images at <a href="http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171">http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171</a> .
Paid QAMI	You can create images providing specific functions that can be launched by anyone willing to pay you per each hour of usage on top of Amazon's charges.

This sequence is supported by public, private, and paid AMIs shown in Figure 6.24. The AMIs are formed from the virtualized compute, storage, and server resources shown at the bottom of Figure 6.23.





**FIGURE 6.23** Amazon EC2 execution environment.

### Example 6.9

## Use of EC2 Services in the AWS Platform

Table 6.13 defines the IaaS instances available in October 2010 in five broad classes:

1. **Standard instances** are well suited for most applications.
2. **Micro instances** provide a small number of consistent CPU resources and allow you to burst CPU capacity when additional cycles are available. They are well suited for lower throughput applications and web sites that consume significant compute cycles periodically.
3. **High-memory instances** offer large memory sizes for high-throughput applications, including database and memory caching applications.
4. **High-CPU instances** have proportionally more CPU resources than memory (RAM) and are well suited for compute-intensive applications.
5. **Cluster compute instances** provide proportionally high CPU resources with increased network performance and are well suited for high-performance computing (HPC) applications and other demanding network-bound applications. They use 10 Gigabit Ethernet interconnections.

**Table 6.13**

**Instance Types Available on Amazon EC2 (October 6, 2010)**

Compute Instance	Memory GB	ECU or EC2 Units	Virtual Cores	Storage GB	32/64 Bit
Standard: small	1.7	1	1	160	32
Standard: large	7.5	4	2	850	64
Standard: extra large	15	8	4	1690	64
Micro	0.613	Up to 2		Only EBS	32 or 64
High-memory	17.1	6.5	2	420	64
High-memory: double	34.2	13	4	850	64
High-memory: quadruple	68.4	26	8	1690	64
High-CPU: medium	1.7	5	2	350	32
High-CPU: extra large	7	20	8	1690	64
Cluster compute	23	33.5	8	1690	64

The cost in the third column is expressed in terms of EC2 Compute Units

(ECUs) where one ECU provides the CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. This leads to the cost per hour for CPUs shown in [Table 6.14](#). Note that a real-world use of EC2 must pay for use of many different resources; the CPU charge in [Table 6.14](#) is just one component and all charges (which naturally change often and so the reader should get the latest values online) are available on the AWS web site.

**Table 6.14**

**Cost of Amazon On-Demand VM Instant Types (October 6, 2010)**

VM Instance Type	Size	Linux/UNIX Usage	Windows Usage
Standard instances	Small (default)	\$0.085 per hour	\$0.12 per hour
	Large	\$0.34 per hour	\$0.48 per hour
	Extra large	\$0.68 per hour	\$0.96 per hour
Micro instances	Micro	\$0.02 per hour	\$0.03 per hour
High-memory instances	Extra large	\$0.50 per hour	\$0.62 per hour
	Double extra large	\$1.00 per hour	\$1.24 per hour
	Quadruple extra large	\$2.00 per hour	\$2.48 per hour
Cluster compute instances	Quadruple extra large	\$1.60 per hour	Not available

## 6.4.2 Amazon Simple Storage Service (S3)

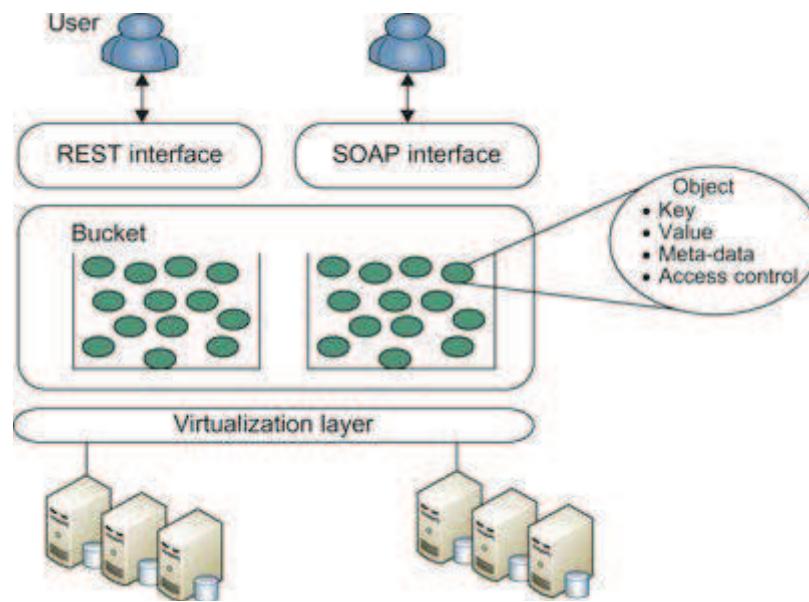
Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. S3 provides the object-oriented storage service for users. Users can access their objects through *Simple Object Access Protocol* (SOAP) with either browsers or other client programs which support SOAP. SQS is responsible for ensuring a reliable message service between two processes, even if the receiver processes are not running. [Figure 6.24](#) shows the S3 execution environment.

The fundamental operation unit of S3 is called an *object*. Each object is stored in a *bucket* and retrieved via a unique, developer-assigned key. In other words, the bucket is the container of the object. Besides unique key attributes, the object has other attributes such as values, metadata, and access control information. From the programmer's perspective, the storage provided by S3 can be viewed as a very coarse-grained key-value pair. Through the key-value programming interface, users can write, read, and delete objects containing from 1 byte to 5 gigabytes of data each. There are two types of web service interface for the user to access the data stored in Amazon clouds. One is a REST (web 2.0) interface, and the other is a SOAP interface. Here are some key features of S3:

- Redundant through geographic dispersion.
- Designed to provide 99.99999999 percent durability and 99.99 percent availability of

objects over a given year with cheaper reduced redundancy storage (RRS).

- Authentication mechanisms to ensure that data is kept secure from unauthorized access. Objects can be made private or public, and rights can be granted to specific users.
- Per-object URLs and ACLs (access control lists).
- Default download protocol of HTTP. A BitTorrent protocol interface is provided to lower costs for high-scale distribution.
- \$0.055 (more than 5,000 TB) to 0.15 per GB per month storage (depending on total amount).
- First 1 GB per month input or output free and then \$.08 to \$.15 per GB for transfers outside an S3 region.
- There is no data transfer charge for data transferred between Amazon EC2 and Amazon S3 within the same region or for data transferred between the Amazon EC2 Northern Virginia region and the Amazon S3 U.S. Standard region (as of October 6, 2010).



**FIGURE 6.24** Amazon S3 execution environment.

## 6.4.3 Amazon Elastic Block Store (EBS) and SimpleDB

The *Elastic Block Store* (EBS) provides the volume block interface for saving and restoring the virtual images of EC2 instances. Traditional EC2 instances will be destroyed after use. The status of EC2 can now be saved in the EBS system after the machine is shut down. Users can use EBS to save persistent data and mount to the running instances of EC2. Note that S3 is “Storage as a Service” with a messaging interface. EBS is analogous to a distributed file system accessed by traditional OS disk access mechanisms. EBS allows you to create storage volumes from 1 GB to 1 TB that can be mounted as EC2 instances.

Multiple volumes can be mounted to the same instance. These storage volumes behave

like raw, unformatted block devices, with user-supplied device names and a block device interface. You can create a file system on top of Amazon EBS volumes, or use them in any other way you would use a block device (like a hard drive). Snapshots are provided so that the data can be saved incrementally. This can improve performance when saving and restoring data. In terms of pricing, Amazon provides a similar pay-per-use schema as EC2 and S3. Volume storage charges are based on the amount of storage users allocate until it is released, and is priced at \$0.10 per GB/month. EBS also charges \$0.10 per 1 million I/O requests made to the storage (as of October 6, 2010). The equivalent of EBS has been offered in open source clouds such Nimbus.

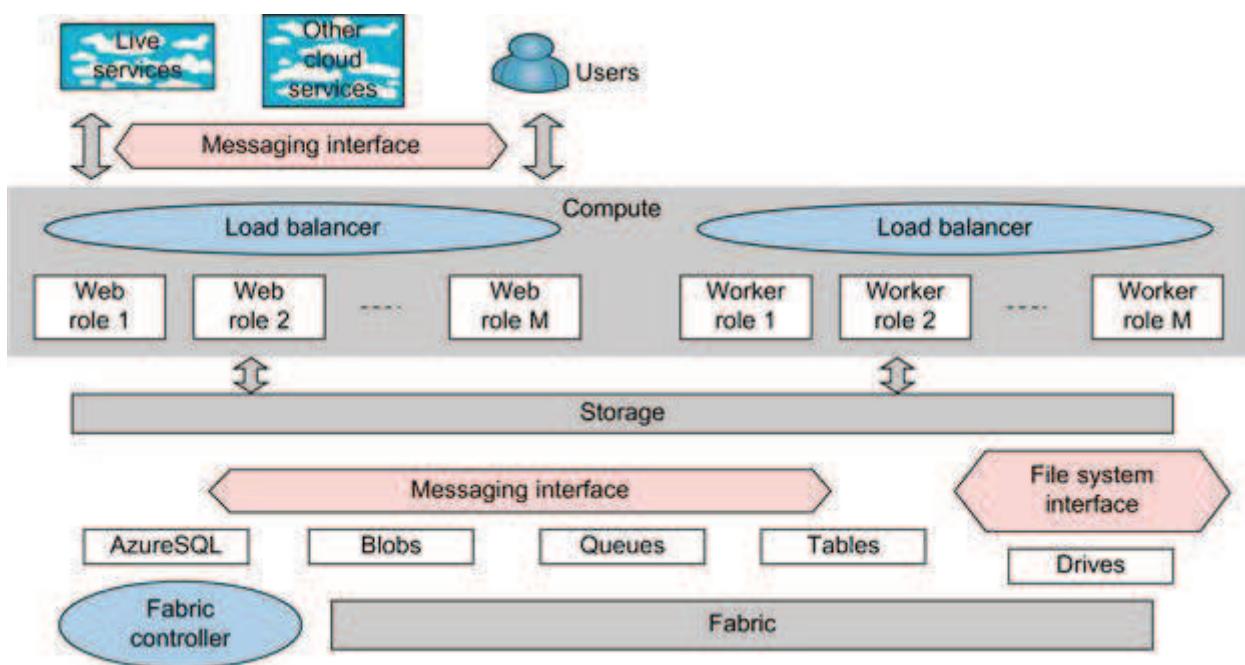
#### 6.4.3.1 Amazon SimpleDB Service

SimpleDB provides a simplified data model based on the relational database data model. Structured data from users must be organized into domains. Each domain can be considered a table. The items are the rows in the table. A cell in the table is recognized as the value for a specific attribute (column name) of the corresponding row. This is similar to a table in a relational database. However, it is possible to assign multiple values to a single cell in the table. This is not permitted in a traditional relational database which wants to maintain data consistency.

Many developers simply want to quickly store, access, and query the stored data. SimpleDB removes the requirement to maintain database schemas with strong consistency. SimpleDB is priced at \$0.140 per Amazon SimpleDB Machine Hour consumed with the first 25 Amazon SimpleDB Machine Hours consumed per month free (as of October 6, 2010). SimpleDB, like Azure Table, could be called “LittleTable,” as they are aimed at managing small amounts of information stored in a distributed table; one could say BigTable is aimed at basic big data, whereas LittleTable is aimed at metadata. Amazon Dynamo [78] is an early research system along the lines of the production SimpleDB system.

#### 6.4.4 Microsoft Azure Programming Support

Section 4.4.4 has introduced the Azure cloud system. In this section, we describe the programming model in more detail. Some key programming components, including the client development environment, SQLAzure, and the rich storage and programming subsystems, are shown in Figure 6.25. We focus on the features of importance in developing Azure programs. First we have the underlying Azure fabric consisting of virtualized hardware together with a sophisticated control environment implementing dynamic assignment of resources and fault tolerance. This implements *domain name system* (DNS) and monitoring capabilities. Automated service management allows service models to be defined by an XML template and multiple service copies to be instantiated on request.



**FIGURE 6.25** Features of the Azure cloud platform.

When the system is running, services are monitored and one can access event logs, trace/debug data, performance counters, IIS web server logs, crash dumps, and other log files. This information can be saved in Azure storage. Note that there is no debugging capability for running cloud applications, but debugging is done from a trace. One can divide the basic features into *storage* and *compute* capabilities. The Azure application is linked to the Internet through a customized compute VM called a *web role* supporting basic Microsoft web hosting. Such configured VMs are often called *appliances*. The other important compute class is the *worker role* reflecting the importance in cloud computing of a pool of compute resources that are scheduled as needed. The roles support HTTP(S) and TCP. Roles offer the following methods:

- The *OnStart()* method which is called by the Fabric on startup, and allows you to perform initialization tasks. It reports a Busy status to the load balancer until you return *true*.
- The *OnStop()* method which is called when the role is to be shut down and gives a graceful exit.
- The *Run()* method which contains the main logic.

As discussed in [Chapter 4](#), the Azure concept of roles is an interesting idea that we can expect to be expanded in terms of role types and use in other cloud environments. [Figure 6.25](#) shows that compute roles can be *load-balanced*, similar to what GAE and AWS clouds have done (see [Section 4.1](#)).

#### 6.4.4.1 SQLAzure

Azure offers a very rich set of *storage* capabilities, as shown in [Figure 6.25](#). The *SQLAzure* service offers SQL Server as a service and is described in detail in [Example 6.10](#). All the storage modalities are accessed with REST interfaces except for the recently introduced *Drives* that are analogous to Amazon EBS discussed in [Section 6.4.3](#), and offer a file

system interface as a durable NTFS volume backed by blob storage. The REST interfaces are automatically associated with URLs and all storage is replicated three times for fault tolerance and is guaranteed to be consistent in access.

The basic storage system is built from *blobs* which are analogous to S3 for Amazon. Blobs are arranged as a three-level hierarchy: *Account* → *Containers* → *Page or Block Blobs*. Containers are analogous to directories in traditional file systems with the account acting as the root. The *block blob* is used for streaming data and each such blob is made up as a sequence of blocks of up to 4 MB each, while each block has a 64 byte ID. Block blobs can be up to 200 GB in size. *Page blobs* are for random read/write access and consist of an array of pages with a maximum blob size of 1 TB. One can associate metadata with blobs as <name, value> pairs with up to 8 KB per blob.

#### **6.4.4.2 Azure Tables**

The Azure Table and Queue storage modes are aimed at much smaller data volumes. *Queues* provide reliable message delivery and are naturally used to support work spooling between web and worker roles. Queues consist of an unlimited number of messages which can be retrieved and processed at least once with an 8 KB limit on message size. Azure supports PUT, GET, and DELETE message operations as well as CREATE and DELETE for queues. Each account can have any number of Azure *tables* which consist of rows called *entities* and columns called *properties*.

There is no limit to the number of entities in a table and the technology is designed to scale well to a large number of entities stored on distributed computers. All entities can have up to 255 general properties which are <name, type, value> triples. Two extra properties, *PartitionKey* and *RowKey*, must be defined for each entity, but otherwise, there are no constraints on the names of properties—this table is very flexible! *RowKey* is designed to give each entity a unique label while *PartitionKey* is designed to be shared and entities with the same *PartitionKey* are stored next to each other; a good use of *PartitionKey* can speed up search performance. An entity can have, at most, 1 MB storage; if you need large value sizes, just store a link to a blob store in the *Table* property value. ADO.NET and LINQ support table queries.

#### **Example 6.10**

#### **SQLAzure Data Services**

Azure provides a sophisticated database programming interface. (More details can be found in [www.microsoft.com/azure/sql.mspx](http://www.microsoft.com/azure/sql.mspx)). The SQLAzure data services can be viewed as more of a traditional relational database. Built on top of the current mature commercial software packages, SQLAzure can be considered a highly scalable, on-demand data storage and query processing utility service. The service interface of SQLAzure is based on standard web protocols and SQLAzure supports both SOAP and REST. As

based on the relational database, the data model provided by SQL Data Services (SDS) is richer than the two NOSQL data management services discussed earlier (BigTable and SimpleDB).

The data model in SQLAzure includes the concepts of authority, container, and entity, with a flexible schema. After a user signs up for the data service, she can create an authority which is represented as a DNS name such as mydomain.data.database.windows.net. Here, mydomain is an authority created by the user and data.database.windows.net refers to the service. Users can create multiple authorities at any time. This DNS name will be resolved to a specific IP address and maps to a specific data center. Thus, an authority as well as its data will be located at a single data center. Under the highest level “authority” comes the concept of a container. An authority can have multiple containers (or none at all). A container has its identifications which can be used as the handle to find the corresponding container in an authority.

Containers are the places where users can put their data. Just like SimpleDB, users can begin to put data in a container without worrying about the data schema. Entities are the units that are stored in a container. An entity can store any number of user-defined properties and associated values (i.e., like the attributes and values in SimpleDB). There are two different kinds of containers: homogeneous and heterogeneous containers. A homogeneous container has all entities of the same type, like a relational database table. A heterogeneous container does not have such limitations. [Figure 6.25](#) shows these concepts.

SDS is one of the building blocks of the Azure platform for building cloud applications. It does provide an enterprise-class data platform. Microsoft has built multiple data centers all over the world to host cloud applications from third parties. Multiple data centers can give the stored data high availability and security. Users do not need to worry about loss of data. Another key feature that SQLAzure provides is ease of development. SDS (in fact, the total Azure Platform SDK) can be integrated with Microsoft’s powerful Visual Studio development environment. This can greatly improve the effectiveness and efficiency with which developers can make cloud applications.

## 6.5 Emerging Cloud Software Environments

In this section, we will assess popular cloud operating systems and emerging software environments. We cover the open source Eucalyptus and Nimbus, then examine OpenNebula, Sector/Sphere, and Open Stack. These environments were introduced in [Chapter 3](#) from a virtualization perspective. Here, we provide details regarding programming requirements. We will also cover the Aneka cloud programming tools

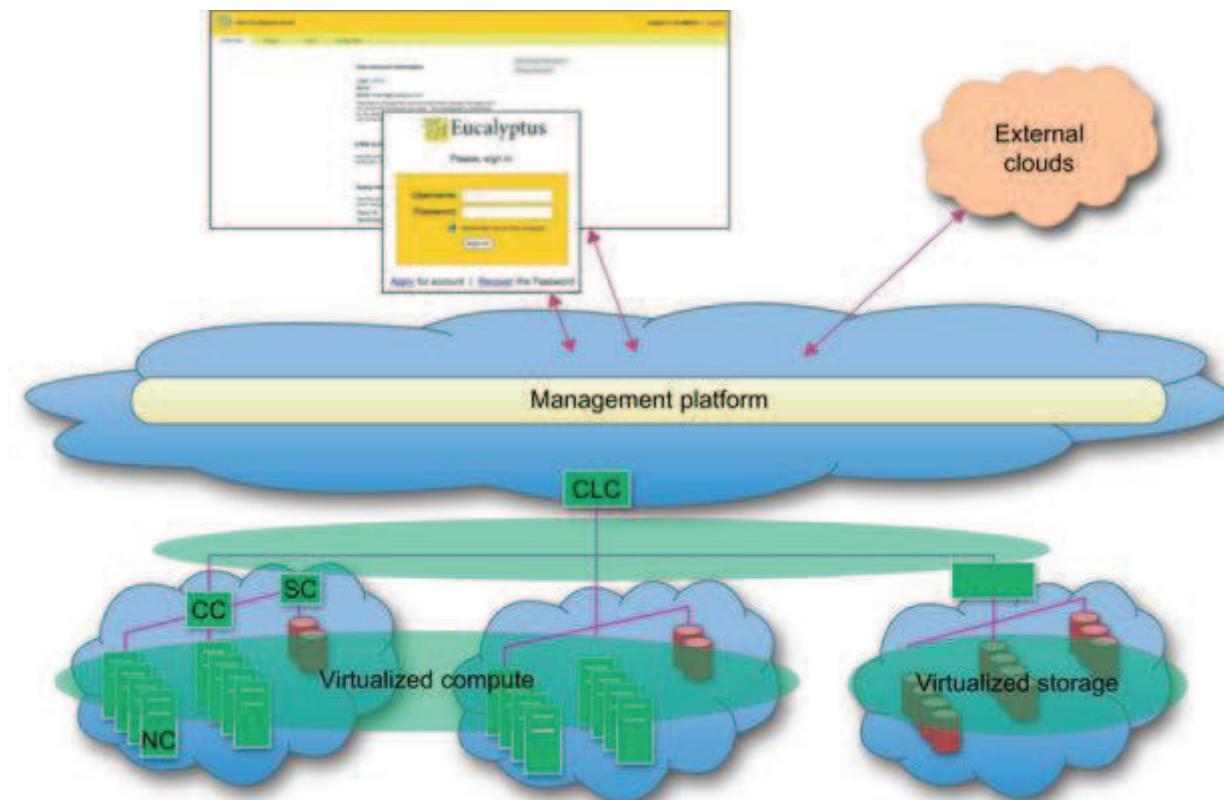
recently developed at the University of Melbourne.

## 6.5.1 Open Source Eucalyptus and Nimbus

Eucalyptus is a product from Eucalyptus Systems ([www.eucalyptus.com](http://www.eucalyptus.com)) that was developed out of a research project at the University of California, Santa Barbara. Eucalyptus was initially aimed at bringing the cloud computing paradigm to academic supercomputers and clusters. Eucalyptus provides an AWS-compliant EC2-based web service interface for interacting with the cloud service. Additionally, Eucalyptus provides services, such as the AWS-compliant Walrus, and a user interface for managing users and images.

### 6.5.1.1 Eucalyptus Architecture

The Eucalyptus system is an open software environment. The architecture was presented in a Eucalyptus white paper [79,80]. Figure 3.31 introduced Eucalyptus from a virtual clustering point of view. Figure 6.26 shows the architecture based on the need to manage VM images. The system supports cloud programmers in VM image management as follows. Essentially, the system has been extended to support the development of both the computer cloud and storage cloud.



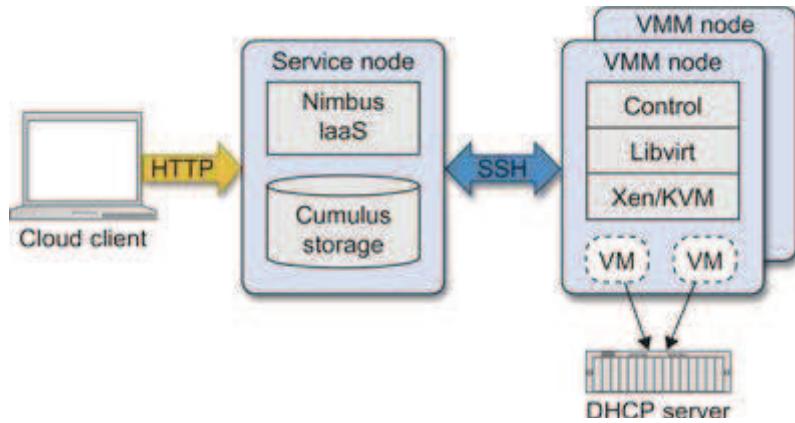
**FIGURE 6.26** The Eucalyptus architecture for VM image management. Courtesy of Eucalyptus LLC [81]

### 6.5.1.2 VM Image Management

Eucalyptus takes many design queues from Amazon's EC2, and its image management system is no different. Eucalyptus stores images in Walrus, the block storage system that is analogous to the Amazon S3 service. As such, any user can bundle her own root file system, and upload and then register this image and link it with a particular kernel and ramdisk image. This image is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime from any availability zone. This allows users to create specialty virtual appliances ([http://en.wikipedia.org/wiki/Virtual\\_appliance](http://en.wikipedia.org/wiki/Virtual_appliance)) and deploy them within Eucalyptus with ease. The Eucalyptus system is available in a commercial proprietary version, as well as the open source version we just described.

### 6.5.1.3 Nimbus

Nimbus [81,82] is a set of open source tools that together provide an IaaS cloud computing solution. Figure 6.27 shows the architecture of Nimbus, which allows a client to lease remote resources by deploying VMs on those resources and configuring them to represent the environment desired by the user. To this end, Nimbus provides a special web interface known as Nimbus Web [83]. Its aim is to provide administrative and user functions in a friendly interface. Nimbus Web is centered around a Python Django [84] web application that is intended to be deployable completely separate from the Nimbus service.



**FIGURE 6.27** Nimbus cloud infrastructure. Courtesy of Nimbus Project [82]

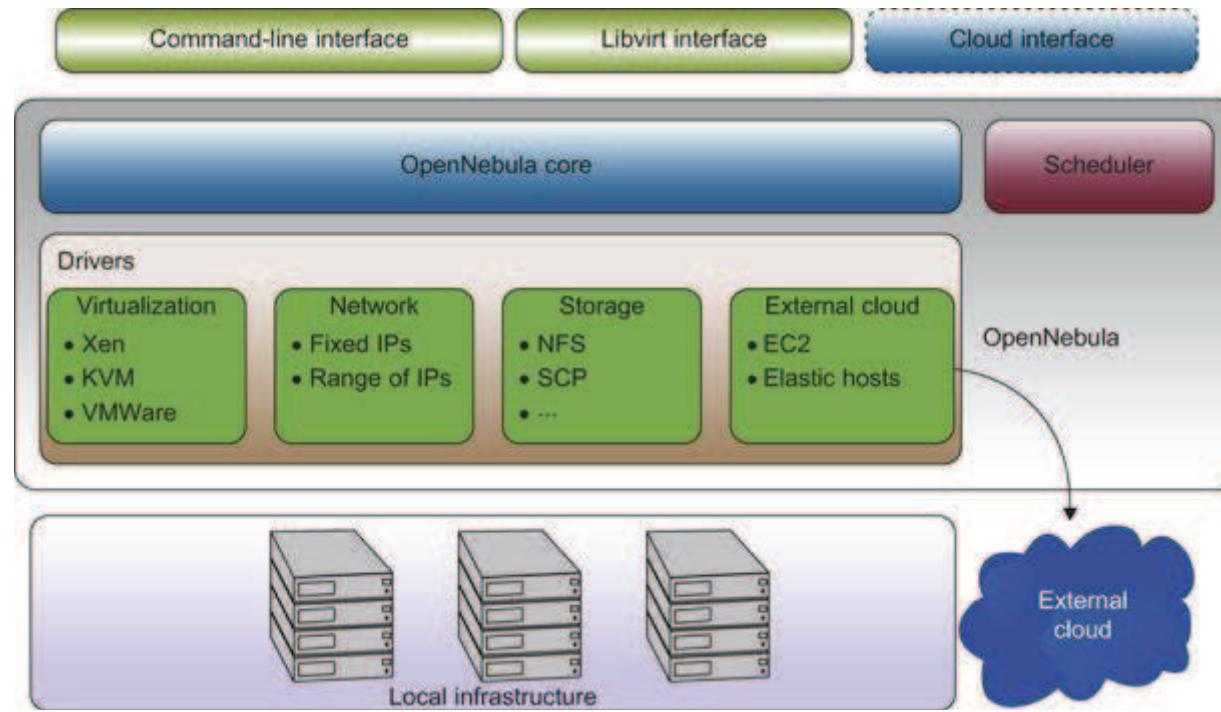
As shown in Figure 6.27, a storage cloud implementation called Cumulus [83] has been tightly integrated with the other central services, although it can also be used stand-alone. Cumulus is compatible with the Amazon S3 REST API [85], but extends its capabilities by including features such as quota management. Therefore, clients such as boto [86] and s2cmd [87], that work against the S3 REST API, work with Cumulus. On the other hand, the Nimbus cloud client uses the Java Jets3t library [88] to interact with Cumulus.

Nimbus supports two resource management strategies. The first is the default “resource pool” mode. In this mode, the service has direct control of a pool of VM manager nodes and it assumes it can start VMs. The other supported mode is called “pilot.” Here, the service makes requests to a cluster’s Local Resource Management

System (LRMS) to get a VM manager available to deploy VMs. Nimbus also provides an implementation of Amazon's EC2 interface [89] that allows users to use clients developed for the real EC2 system against Nimbus-based clouds.

## 6.5.2 OpenNebula, Sector/Sphere, and OpenStack

OpenNebula [90,91] is an open source toolkit which allows users to transform existing infrastructure into an IaaS cloud with cloud-like interfaces. Figure 6.28 shows the OpenNebula architecture and its main components. The architecture of OpenNebula [92] has been designed to be flexible and modular to allow integration with different storage and network infrastructure configurations, and hypervisor technologies. Here, the core is a centralized component that manages the VM full life cycle, including setting up networks dynamically for groups of VMs and managing their storage requirements, such as VM disk image deployment or on-the-fly software environment creation.



**FIGURE 6.28** OpenNebula architecture and its main components. Courtesy of Sotomayor, et al. [94]

Another important component is the capacity manager or scheduler. It governs the functionality provided by the core. The default capacity scheduler is a requirement/rank matchmaker. However, it is also possible to develop more complex scheduling policies, through a lease model and advance reservations [93]. The last main components are the access drivers. They provide an abstraction of the underlying infrastructure to expose the basic functionality of the monitoring, storage, and virtualization services available in the cluster. Therefore, OpenNebula is not tied to any specific environment and can provide a uniform management layer regardless of the virtualization platform.

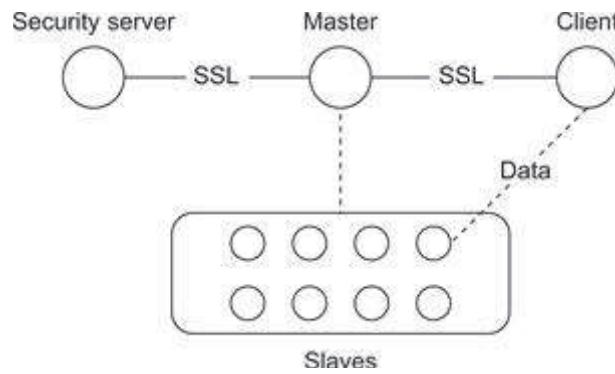
Additionally, OpenNebula offers management interfaces to integrate the core's functionality within other data-center management tools, such as accounting or

monitoring frameworks. To this end, OpenNebula implements the libvirt API [94], an open interface for VM management, as well as a command-line interface (CLI). A subset of this functionality is exposed to external users through a cloud interface. OpenNebula is able to adapt to organizations with changing resource needs, including addition or failure of physical resources [95]. Some essential features to support changing environments are live migration and VM snapshots [90].

Furthermore, when the local resources are insufficient, OpenNebula can support a hybrid cloud model by using cloud drivers to interface with external clouds. This lets organizations supplement their local infrastructure with computing capacity from a public cloud to meet peak demands, or implement HA strategies. OpenNebula currently includes an EC2 driver, which can submit requests to Amazon EC2 [89] and Eucalyptus [80], as well as an ElasticHosts driver [96]. Regarding storage, an Image Repository allows users to easily specify disk images from a catalog without worrying about low-level disk configuration attributes or block device mapping. Also, image access control is applied to the images registered in the repository, hence simplifying multiuser environments and image sharing. Nevertheless, users can also set up their own images.

### 6.5.2.1 Sector/Sphere

Sector/Sphere is a software platform that supports very large distributed data storage and simplified distributed data processing over large clusters of commodity computers, either within a data center or across multiple data centers. The system consists of the Sector distributed file system and the Sphere parallel data processing framework [97,98]. Sector is a *distributed file system (DFS)* that can be deployed over a wide area and allows users to manage large data sets from any location with a high-speed network connection [99]. The fault tolerance is implemented by replicating data in the file system and managing the replicas.



**FIGURE 6.29** The Sector/Sphere system architecture. Courtesy of Gu and Grossman [102]

Since Sector is aware of the network topology when it places replicas, it also provides better reliability, availability, and access throughout. The communication is performed using User Datagram Protocol (UDP) for message passing and user-defined type (UDT) [100] for data transfer. Obviously, UDP is faster than TCP for message passing because it does not require connection setup, but it could become a problem if Sector is used over

the Internet. Meanwhile, UDT is a reliable UDP-based application-level data transport protocol which has been specifically designed to enable high-speed data transfer over wide area high-speed networks [100]. Finally, the Sector client provides a programming API, tools, and a FUSE [101] user space file system module.

On the other hand, Sphere is a parallel data processing engine designed to work with data managed by Sector. This coupling allows the system to make accurate decisions about job scheduling and data location. Sphere provides a programming framework that developers can use to process data stored in Sector. Thus, it allows UDFs to run on all input data segments in parallel. Such data segments are processed at their storage locations whenever possible (data locality). Failed data segments may be restarted on other nodes to achieve fault tolerance. In a Sphere application, both inputs and outputs are Sector files. Multiple Sphere processing segments can be combined to support more complicated applications, with inputs/outputs exchanged/shared via the Sector file system [102].

The Sector/Sphere platform [102] is supported by the architecture shown in Figure 6.29, which is composed of four components. The first component is the security server, which is responsible for authenticating master servers, slave nodes, and users. We also have the master servers that can be considered the infrastructure core. The master server maintains file system metadata, schedules jobs, and responds to users' requests. Sector supports multiple active masters that can join and leave at runtime and can manage the requests. Another component is the slave nodes, where data is stored and processed. The slave nodes can be located within a single data center or across multiple data centers with high-speed network connections. The last component is the client component. This provides tools and programming APIs for accessing and processing Sector data.

Finally, it is worthy to mention that as part of this platform, a new component has been developed. It is called Space [97] and it consists of a framework to support column-based distributed data tables. Therefore, tables are stored by columns and are segmented onto multiple slave nodes. Tables are independent and no relationships between them are supported. A reduced set of SQL operations is supported, including, but not limited to, table creation and modification, key-value updates and lookups, and select UDF operations.

### 6.5.2.2 OpenStack

OpenStack [103] was introduced by Rackspace and NASA in July 2010. The project is building an open source community spanning technologists, developers, researchers, and industry to share resources and technologies with the goal of creating a massively scalable and secure cloud infrastructure. In the tradition of other open source projects, the software is open source and limited to just open source APIs such as Amazon.

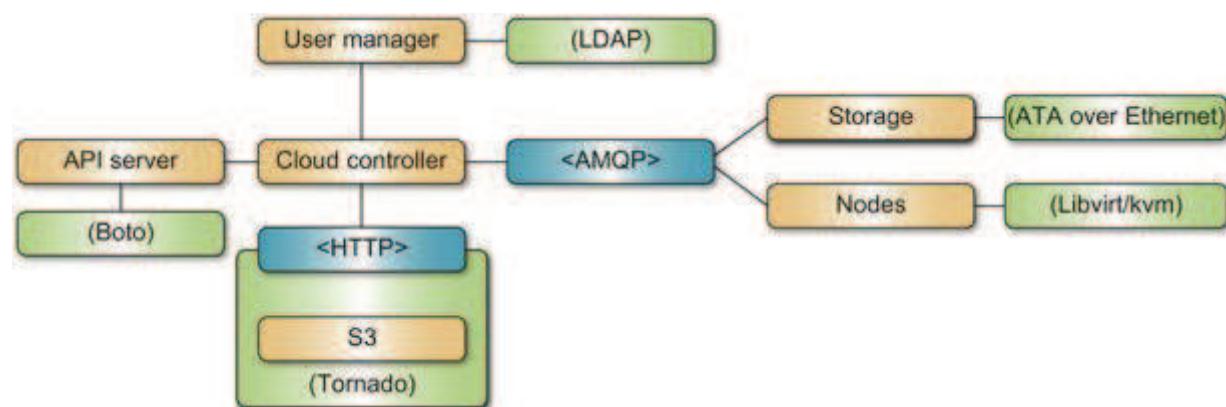
Currently, OpenStack focuses on the development of two aspects of cloud computing to address compute and storage aspects with the OpenStack Compute and OpenStack Storage solutions. “OpenStack Compute is the internal fabric of the cloud creating and managing large groups of virtual private servers” and “OpenStack Object Storage is software for creating redundant, scalable object storage using clusters of commodity

servers to store terabytes or even petabytes of data.” Recently, an image repository was prototyped. The image repository contains an image registration and discovery service and an image delivery service. Together they deliver images to the compute service while obtaining them from the storage service. This development gives an indication that the project is striving to integrate more services into its portfolio.

### 6.5.2.3 OpenStack Compute

As part of its computing support efforts, OpenStack [103] is developing a cloud computing fabric controller, a component of an IaaS system, known as Nova. The architecture for Nova is built on the concepts of shared-nothing and messaging-based information exchange. Hence, most communication in Nova is facilitated by message queues. To prevent blocking components while waiting for a response from others, deferred objects are introduced. Such objects include callbacks that get triggered when a response is received. This is very similar to established concepts from parallel computing, such as “futures,” which have been used in the grid community by projects such as the CoG Kit.

To achieve the shared-nothing paradigm, the overall system state is kept in a distributed data system. State updates are made consistent through atomic transactions. Nova is implemented in Python while utilizing a number of externally supported libraries and components. This includes boto, an Amazon API provided in Python, and Tornado, a fast HTTP server used to implement the S3 capabilities in OpenStack. [Figure 6.30](#) shows the main architecture of Open Stack Compute. In this architecture, the API Server receives HTTP requests from boto, converts the commands to and from the API format, and forwards the requests to the cloud controller.



**FIGURE 6.30** OpenStack Nova system architecture. The AMQP (Advanced Messaging Queuing Protocol) was described in [Section 5.2](#).

The cloud controller maintains the global state of the system, ensures authorization while interacting with the User Manager via Lightweight Directory Access Protocol (LDAP), interacts with the S3 service, and manages nodes, as well as storage workers through a queue. Additionally, Nova integrates networking components to manage private networks, public IP addressing, virtual private network (VPN) connectivity, and

firewall rules. It includes the following types:

- *NetworkController* manages address and virtual LAN (VLAN) allocations
- *RoutingNode* governs the NAT (*network address translation*) conversion of public IPs to private IPs, and enforces firewall rules
- *AddressingNode* runs Dynamic Host Configuration Protocol (DHCP) services for private networks
- *TunnelingNode* provides VPN connectivity

The network state (managed in the distributed object store) consists of the following:

- VLAN assignment to a project
- Private subnet assignment to a security group in a VLAN
- Private IP assignments to running instances
- Public IP allocations to a project
- Public IP associations to a private IP/running instance

#### **6.5.2.4 OpenStack Storage**

The OpenStack storage solution is built around a number of interacting components and concepts, including a proxy server, a ring, an object server, a container server, an account server, replication, updaters, and auditors. The role of the proxy server is to enable lookups to the accounts, containers, or objects in OpenStack storage rings and route the requests. Thus, any object is streamed to or from an object server directly through the proxy server to or from the user. A ring represents a mapping between the names of entities stored on disk and their physical locations.

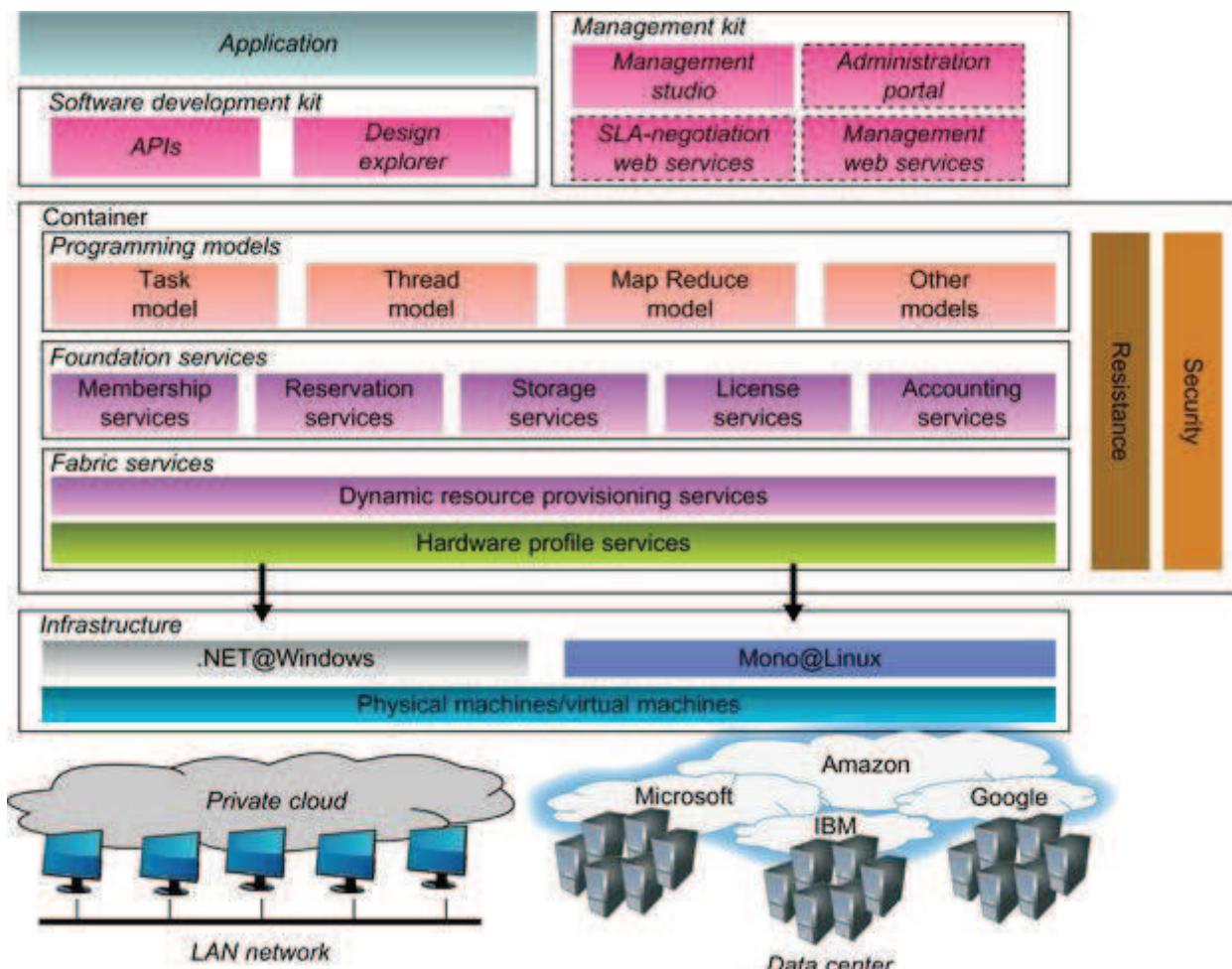
Separate rings for accounts, containers, and objects exist. A ring includes the concept of using zones, devices, partitions, and replicas. Hence, it allows the system to deal with failures, and isolation of zones representing a drive, a server, a cabinet, a switch, or even a data center. Weights can be used to balance the distribution of partitions on drives across the cluster, allowing users to support heterogeneous storage resources. According to the documentation, “the Object Server is a very simple blob storage server that can store, retrieve and delete objects stored on local devices.”

Objects are stored as binary files with metadata stored in the file’s extended attributes. This requires that the underlying file system is built around object servers, which is often not the case for standard Linux installations. To list objects, a container server can be utilized. Listing of containers is handled by the account server. The first release of OpenStack “Austin” Compute and Object Storage was October 22, 2010. This system has a strong developer community.

#### **6.5.3 Manjrasoft Aneka Cloud and Appliances**

Aneka ([www.manjrasoft.com/](http://www.manjrasoft.com/)) is a cloud application platform developed by Manjrasoft, based in Melbourne, Australia. It is designed to support rapid development and deployment of parallel and distributed applications on private or public clouds. It provides a rich set of APIs for transparently exploiting distributed resources and expressing the business logic of applications by using preferred programming

abstractions. System administrators can leverage a collection of tools to monitor and control the deployed infrastructure. It can be deployed on a public cloud such as Amazon EC2 accessible through the Internet to its subscribers, or a private cloud constituted by a set of nodes with restricted access as shown in [Figure 6.31](#).



**FIGURE 6.31** Architecture and components of Aneka. Courtesy of Raj Buyya, Manjrasoft

Aneka acts as a workload distribution and management platform for accelerating applications in both Linux and Microsoft .NET framework environments. Some of the key advantages of Aneka over other workload distribution solutions include:

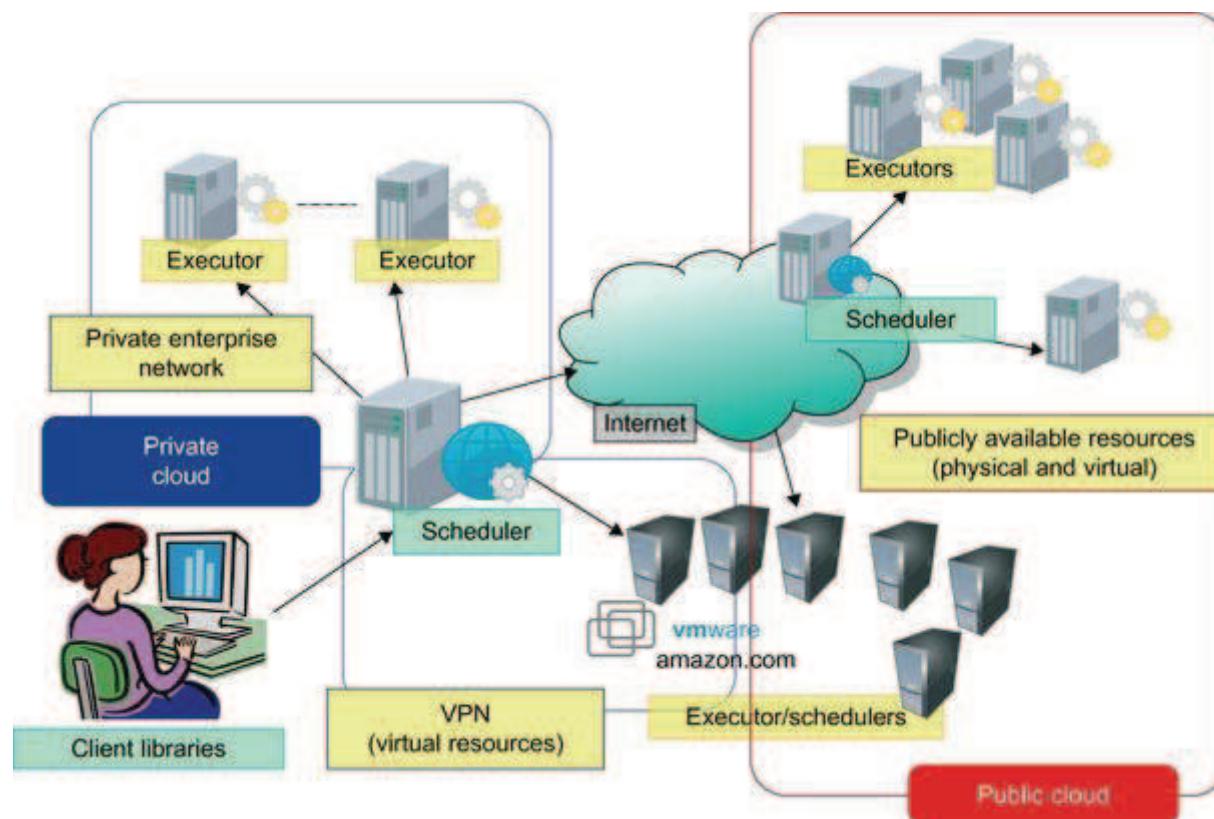
- Support of multiple programming and application environments
- Simultaneous support of multiple runtime environments
- Rapid deployment tools and framework
- Ability to harness multiple virtual and/or physical machines for accelerating application provisioning based on users' Quality of Service/service-level agreement (QoS/SLA) requirements
- Built on top of the Microsoft .NET framework, with support for Linux environments through Mono

Aneka offers three types of capabilities which are essential for building, accelerating, and managing clouds and their applications:

1. **Build** Aneka includes a new SDK which combines APIs and tools to enable users to

rapidly develop applications. Aneka also allows users to build different runtime environments such as enterprise/private cloud by harnessing compute resources in network or enterprise data centers, Amazon EC2, and hybrid clouds by combining enterprise private clouds managed by Aneka with resources from Amazon EC2 or other enterprise clouds built and managed using XenServer.

2. **Accelerate** Aneka supports rapid development and deployment of applications in multiple runtime environments running different operating systems such as Windows or Linux/UNIX. Aneka uses physical machines as much as possible to achieve maximum utilization in local environments. Whenever users set QoS parameters such as deadlines, and if the enterprise resources are insufficient to meet the deadline, Aneka supports dynamic leasing of extra capabilities from public clouds such as EC2 to complete the task within the deadline (see [Figure 6.32](#)).
3. **Manage** Management tools and capabilities supported by Aneka include a GUI and APIs to set up, monitor, manage, and maintain remote and global Aneka compute clouds. Aneka also has an accounting mechanism and manages priorities and scalability based on SLA/QoS which enables dynamic provisioning.



**FIGURE 6.32** Aneka using private cloud resources along with dynamically leased public cloud resources. Courtesy of Raj Buyya, Manjrasoft, [www.manjrasoft.com/](http://www.manjrasoft.com/)

Here are three important programming models supported by Aneka for both cloud and traditional parallel applications:

1. Thread programming model, best solution to adopt for leveraging the computing capabilities of multicore nodes in a cloud of computers
2. Task programming model, which allows for quickly prototyping and implementing an

independent bag of task applications

3. MapReduce programming model, as discussed in [Section 6.2.2](#)

### 6.5.3.1 Aneka Architecture

Aneka as a cloud application platform features a homogeneous distributed runtime environment for applications. This environment is built by aggregating together physical and virtual nodes hosting the Aneka container. The container is a lightweight layer that interfaces with the hosting environment and manages the services deployed on a node. The interaction with the hosting platform is mediated through the *Platform Abstraction Layer (PAL)*, which hides in its implementation all the heterogeneity of the different operating systems.

By means of the PAL it is possible to perform all the infrastructure-related tasks, such as performance and system monitoring. These activities are vital to ensure the desired QoS for applications. The PAL, together with the container, represents the hosting environment of services which implement the core capabilities of the middleware and make it a dynamically composable and extensible system. The available services can be aggregated into three major categories:

**Fabric Services:** Fabric services implement the fundamental operations of the infrastructure of the cloud. These services include HA and failover for improved reliability, node membership and directory, resource provisioning, performance monitoring, and hardware profiling.

**Foundation Services:** Foundation services constitute the core functionalities of the Aneka middleware. They provide a basic set of capabilities that enhance application execution in the cloud. These services provide added value to the infrastructure and are of use to system administrators and developers. Within this category we can list storage management, resource reservation, reporting, accounting, billing, services monitoring, and licensing. Services in this level operate across the range of supported application models.

**Application Services:** Application services deal directly with the execution of applications and are in charge of providing the appropriate runtime environment for each application model. They leverage foundation and fabric services for several tasks of application execution such as elastic scalability, data transfer, and performance monitoring, accounting, and billing. At this level, Aneka expresses its true potential in supporting different application models and distributed programming patterns.

Each supported application model is managed by a different collection of services that interact with the underlying layers and services to carry out application execution. In general, the middleware counterpart of each application model features at least two different services: scheduling and execution. In addition, specific models can require additional services or a different type of support. Aneka provides support for the most well-known application programming patterns, such as distributed threads, bags of tasks, and MapReduce.

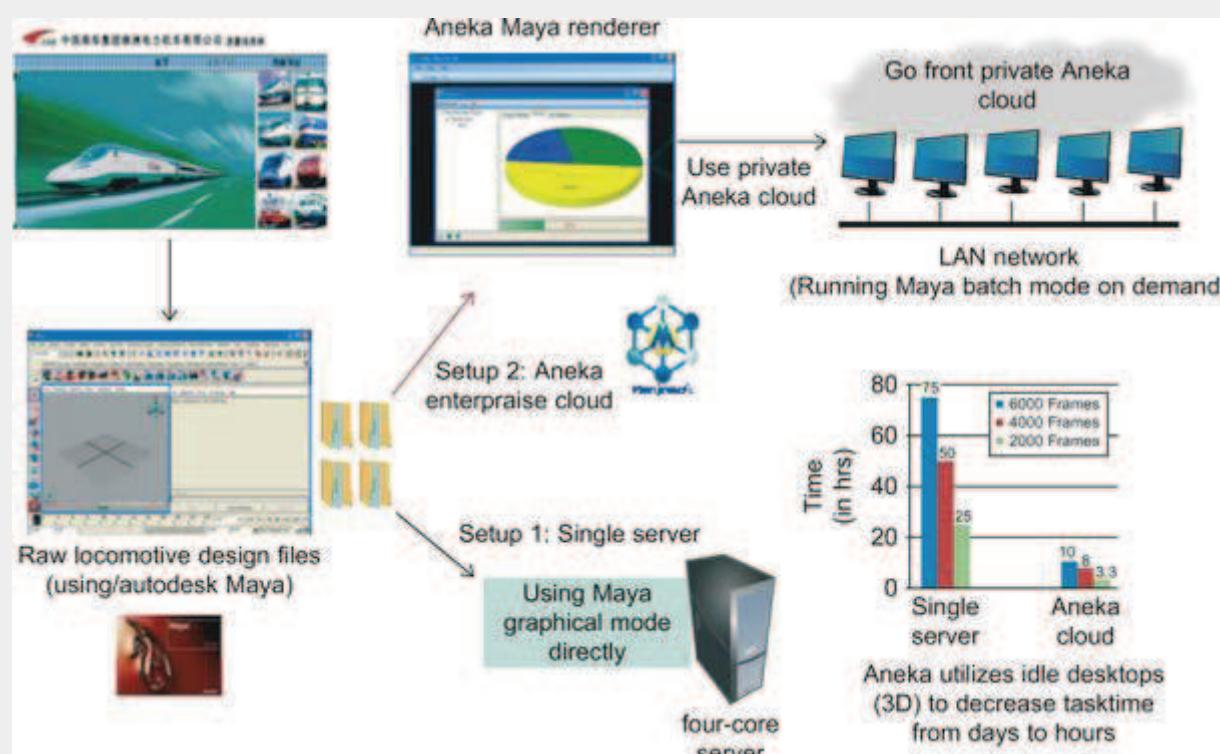
Additional services can be designed and deployed in the system. This is how the infrastructure is enriched with additional features and capabilities. The SDK provides

straightforward interfaces and ready-to-use components for rapid service prototyping. Deployment and integration of new services is painless and immediate: The container leverages the Spring framework and allows for dynamic integration of new components such as services.

## Example 6.11

### Aneka Application of Maya Rendering Case Study

Aneka has been used to create several interesting applications in domains such as life sciences, engineering, and creative media. Applications created using Aneka are able to run on enterprise or public clouds unchanged. We will briefly discuss a case study on high-speed rendering of engineering designs using Aneka software. To reduce this time, GoFront has used Aneka and created an enterprise cloud (Figure 6.33) within its company by utilizing networked PCs. GoFront Group, a unit of China Southern Railways, is China's premier and largest nationwide researcher and manufacturer of rail electric traction equipment.



**FIGURE 6.33** Rendering images of locomotive design on GoFront's private cloud using Aneka.  
Courtesy of Raj Buyya, Manjrasoft

The GoFront Group is responsible for designing the high-speed electric locomotive, metro car, urban transportation vehicle, and motor train. The raw design of the prototypes requires high-quality 3D images using Autodesk's Maya rendering software. By examining the 3D images, engineers identify

problems in the original design and make the appropriate design improvements. However, such designs on a single four-core server took three days to render scenes with 2,000 frames.

To reduce this time, GoFront used Aneka and created an enterprise cloud by utilizing networked PCs. It used Aneka Design Explorer, a tool for rapid creation of parameter sweep applications in which the same program is executed many times on different data items (in this case, executing Maya for rendering different images). A customized Design Explorer (called Maya GUI) has been implemented for Maya rendering. Maya GUI managed parameters, generated Aneka tasks, monitored submitted Aneka tasks, and collected final rendered images. The design image used to take three days to render (2,000+ frames, each frame with more than five different camera angles). Using only a 20-node Aneka cloud, GoFront was able to reduce the rendering time from three days to three hours.

### **6.5.3.2 Virtual Appliances**

Machine virtualization offers a unique opportunity to break software dependencies between applications and their hosting environments. In recent years, resource virtualization has been witnessing a renaissance fueled by efficient, freely available VM monitors for commodity systems (e.g., Xen, VMware Player, KVM, and VirtualBox) and backed by the microprocessor industry (e.g., Intel and AMD virtualization extensions). Modern system VMs provide improved flexibility, security, isolation, and resource control, while supporting a wealth of unmodified applications, making a compelling case for their use in grid computing. Grid applications also require network connectivity, and the increasing use of NAT and IP firewalls breaks the original model of each node in the Internet being a peer, and is recognized as a hindrance to programming and deploying grid computing systems.

In Aneka, VMs and P2P network virtualization technologies can be integrated into self-configuring, prepackaged “virtual appliances” to enable simple deployment of homogeneously configured virtual clusters across a heterogeneous, wide-area distributed system. Virtual appliances are VM images that are installed and configured with the entire software stack (including OS, libraries, binaries, configuration files, and auto-configuration scripts) that is required for a given application to work “out-of-the-box” when the virtual appliance is instantiated. Compared to traditional approaches of software distribution, virtual appliances have the benefit of significantly reducing software dependence on the hosting environment.

VM conversion tools exist for most VMMs and standardization efforts that are underway will further facilitate interoperation across VMMs, such that appliance instantiation across multiple platforms should be seamless. Wide-area overlays of virtual appliances aggregate the capacity of commodity hardware and can be programmed and

managed as a local area network—even though nodes may be distributed across multiple network domains. The preconfigured Grid Appliance image ([www.grid-appliance.org](http://www.grid-appliance.org)) can encapsulate complex distributed system software in a manner that is transparent to users for easy deployment. Grid appliances run on free VMs available for modern commodity servers and desktops.

## 6.6 Bibliographic Notes and Homework Problems

Detailed references for core technologies have been given throughout this chapter, and here we give some additional useful references. Most of the topics are so new that there are few comprehensive treatments, and links for major commercial clouds are changing at this time of rapid evolution; we recommend checking major sites for latest links. A sample is [104–108]. A very good introduction to cloud computing in business and technology is given in the lecture notes by Chou [109]. Various cloud services are treated in [110–115]. Arguments about the benefits and opportunities of cloud computing are treated in [110,116–120]. Cloud technology for HPC was studied in [28,43,73,121–127]. Distributed programming paradigms for cloud applications are treated in [26,28]. Applications of clouds to data-intensive computing are discussed in [25,41–43,74,75].

## Acknowledgement

This chapter was coauthored by Geoffrey Fox of Indiana University and Albert Zomaya of Sydney University, Australia. Their work was assisted by Kang Chen (Tsinghua University, China), Judy Qiu, Gregor von Laszewski, Javier Diaz, Archit Kulshrestha, and Andrew Younge (Indiana University), Reza Moravaeji and Javid Teheri (University of Sydney), and Renato Figueiredo (University of Florida). Rajkumar Buyya (University of Melbourne) has contributed Section 6.5.3. Kai Hwang (USC) edited the final manuscript.

## References

1. Dabrowski C. Reliability in grid computing systems. *Concurr Comput Pract Exper.* 2009;21(8):927–959.
2. Microsoft. Project Trident: A Scientific Workflow Workbench. In: <http://research.microsoft.com/en-us/collaboration/tools/trident.aspx>; 2010.
3. Lu W, Jackson J, Barga R. AzureBlast: A case study of developing science applications on the cloud. In: *ScienceCloud: 1st Workshop on Scientific Cloud Computing co-located with HPDC (High Performance Distributed Computing)*. Chicago, IL: ACM; 21 June 2010.
4. Distributed Systems Laboratory (DSL) at University of Chicago Wiki. Performance Comparison: Remote Usage, NFS, S3-fuse, EBS. 2010.
5. Jensen D. Blog entry on Compare Amazon S3 to EBS data read performance. In: <http://jensendarren.wordpress.com/2009/12/30/compare-amazon-s3-to-ebs-data-read-performance/>; 2009.

6. Zend PHP Company. The Simple Cloud API for Storage, Queues and Table. In: <http://www.simplecloud.org/home>; 2010.
7. Microsoft. Windows Azure Geo-location Live. In: <http://blogs.msdn.com/b/windowsazure/archive/2009/04/30/windows-azure-geo-location-live.aspx>; 2009.
8. Raytheon BBN, SHARD (Scalable, High-Performance, Robust and Distributed) Triple Store based on Hadoop. <http://www.cloudera.com/blog/2010/03/how-raytheon-researchers-are-using-hadoop-to-build-a-scalable-distributed-triple-store/>, 2010.
9. NOSQL Movement. Wikipedia list of resources. In: <http://en.wikipedia.org/wiki/NoSQL>; 2010.
10. NOSQL Link Archive. LIST OF NOSQL DATABASES. In: <http://nosql-database.org/>; 2010.
11. Chang F, Dean J, Ghemawat S, et al. BigTable: A distributed storage system for structured data. In: *OSDI'06: Seventh Symposium on Operating System Design and Implementation*. Seattle, WA: USENIX; 2006.
12. Amazon. Welcome to Amazon SimpleDB. In: <http://docs.amazonwebservices.com/AmazonSimpleDB/latest/DeveloperGuide/index.html> 2010.
13. Haridas J, Nilakantan N, Calder B. Windows Azure Table. In: <http://go.microsoft.com/fwlink/?LinkId=153401>; 2009.
14. International Virtual Observatory Alliance. VOTable Format Definition Version 1.1. In: <http://www.ivoa.net/Documents/VOTable/20040811/>; 2004.
15. Apache Incubator. Heart (Highly Extensible & Accumulative RDF Table) planet-scale RDF data store and a distributed processing engine based on Hadoop & Hbase. In: <http://wiki.apache.org/incubator/HeartProposal>; 2010.
16. King M. Amazon SimpleDB and CouchDB Compared. In: <http://www.automatthew.com/2007/12/amazon-simplesdb-and-couchdb-compared.html>; 2007.
17. Apache. Hbase implementation of BigTable on Hadoop File System. In: <http://hbase.apache.org/>; 2010.
18. Apache. The CouchDB document-oriented database project. In: <http://couchdb.apache.org/index.html>; 2010.
19. M/Gateway Developments Ltd. M/DB Open Source “plug-compatible” alternative to Amazon’s SimpleDB database. In: <http://gradvs1.mgateway.com/main/index.html?path=mdb>; 2009.
20. ActiveMQ. In: <http://activemq.apache.org/>; 2009.
21. Pallickara S, Fox G. NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids. In: *ACM/IFIP/USENIX 2003 International Conference on Middleware*. Rio de Janeiro, Brazil. New York, Inc.: Springer-Verlag; 2003.
22. NaradaBrokering. Scalable Publish Subscribe System. In: <http://www.naradabrokering.org/>; 2010.

23. Apache Hadoop. In: <http://hadoop.apache.org/>; 2009.
24. Ekanayake J, Balkir AS, Gunarathne T, et al. DryadLINQ for scientific analyses. In: *Fifth IEEE International Conference on eScience*. Oxford. 2009.
25. Ekanayake J, Gunarathne T, Qiu J, et al. *Applicability of DryadLINQ to Scientific Applications* Community Grids Laboratory, Indiana University 2009.
26. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: Distributed data-parallel programs from sequential building blocks. In: *ACM SIGOPS Operating Systems Review*. ACM Press 2007.
27. Yu Y, Isard M, Fetterly D, et al. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In: *Symposium on Operating System Design and Implementation (OSDI)*. 2008.
28. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Commun ACM*. 2008;51(1):107–113.
29. Ekanayake J, Li H, Zhang B, et al. Twister: a runtime for iterative MapReduce. In: *Proceedings of the First International Workshop on MapReduce and Its Applications of ACM HPDC 2010 conference*. Chicago, IL: ACM; 20–25 June 2010.
30. Gunarathne T, Wu T, Qiu J, Fox G. MapReduce in the Clouds for Science. In: *CloudCom, IUPUI Conference Center*. Indianapolis. 30 November–3 December 2010.
31. Dorward RS, Griesemer R, Quinlan S. Interpreting the data: parallel analysis with Sawzall. *Scientific Prog J*. 2005;13(4):227–298 (Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure).
32. Pig! Platform for analyzing large data sets. In: <http://hadoop.apache.org/pig/>; 2010.
33. Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig Latin: a not-so-foreign language for data processing. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. Vancouver, Canada: ACM; 2008;1099–1110.
34. Malewicz G, Austern MH, Bik A, et al. Pregel: a system for large-scale graph processing. In: *Proceedings of the twenty-first annual symposium on parallelism in algorithms and architectures*. Calgary, Canada: ACM; 2009;48.
35. Cloudera, CDH: A free, stable Hadoop distribution offering RPM, Debian, AWS and automatic configuration options. <http://www.cloudera.com/hadoop/>, 2010.
36. Grama A, Karypis G, Kumar V, Gupta A. *Introduction to Parallel Computing* second ed. Addison Wesley 2003.
37. Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. In: *Sixth Symposium on Operating Systems Design and Implementation*. 2004;137–150.
38. Kasim H, March V, Zhang R, See S. Survey on Parallel Programming Model. In: Shanghai, China: Springer-Verlag; 2008;266–275. *IFIP International Conference on Network and Parallel Computing, Lecture Notes in Computer Science*. Vol. 5245.
39. Hariri S, Parashar M. *Tools and Environments for Parallel and Distributed Computing, Series on Parallel and Distributed Computing* Wiley 2004; ISBN:978-0471332886.
40. L. Silva, R. Buyya, Parallel Programming Models and Paradigms, (2007).
41. Gunarathne T, Wu T, Qiu J, Fox G. Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications. In: *Proceedings of the Emerging Computational*

42. Fox G. MPI and MapReduce. In: *Clusters, Clouds, and Grids for Scientific Computing* CCGSC. NC: Flat Rock; 8 September 2010; In: <http://grids.ucs.indiana.edu/ptliupages/presentations/CCGSC-Sept8-2010.pptx>; 8 September 2010.
43. Ekanayake J, Qiu X, Gunarathne T, Beason S, Fox G. *High Performance Parallel Computing with Clouds and Cloud Technologies* Cloud Computing and Software Services: Theory and Techniques, CRC Press (Taylor and Francis) 2010.
44. Hoefler T, Lumsdaine A, Dongarra J. Towards Efficient MapReduce Using MPI. In: Espoo Finland: Springer Verlag; 2009;240–249. *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science*. vol. 5759.
45. Ibrahim S, Jin H, Cheng B, Cao H, Wu S, Qi L. CLOUDLET: towards mapreduce implementation on virtual machines. In: *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*. Garching, Germany: ACM; 2009;65–66.
46. Sandholm T, Lai K. MapReduce optimization using regulated dynamic prioritization. In: *Proceedings of the eleventh international joint conference on measurement and modeling of computer systems*. Seattle, WA: ACM; 2009;299–310.
47. Wikipedia. *MapReduce*. <http://en.wikipedia.org/wiki/MapReduce>; 2010; (accessed 06.11.10).
48. J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Presentation at OSDI-2004 Conference.  
<http://labs.google.com/papers/mapreduce-osdi04-slides/index.html>, 2004 (accessed 6.11.10).
49. Lammel R. Google's MapReduce programming model – Revisited. *Sci Comput Prog*. 2007;68(3):208–237.
50. Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I. Improving MapReduce performance in heterogeneous environments. In: *Proceedings of the 8th USENIX conference on operating systems design and implementation*. San Diego, California: USENIX Association; 2008;29–42.
51. Vasic N, Barisits M, Salzgeber V, Kostic D. Making cluster applications energy-aware. In: *Proceedings of the 1st workshop on automated control for datacenters and clouds*. Barcelona, Spain: ACM; 2009;37–42.
52. DeWitt DJ, Paulson E, Robinson E, et al. Clustera: an integrated computation and data management system. *Proc VLDB Endow*. 2008;1(1):28–41.
53. Ghemawat S, Gobioff H, Leung S. The Google File System. In: *19th ACM Symposium on Operating Systems Principles*. 2003;20–43.
54. Google. Introduction to Parallel Programming and MapReduce. In: <http://code.google.com/edu/parallel/mapreduce-tutorial.html>; 2010.
55. Ekanayake J, Li H, Zhang B, et al. Twister: a runtime for iterative MapReduce. In: *Proceedings of the First International Workshop on MapReduce and Its Applications of*

ACM HPDC 2010 Conference. Chicago, IL: ACM; 20–25 June 2010.

56. Zhang B, Ruan Y, Wu T, Qiu J, Hughes A, Fox G. Applying Twister to Scientific Applications. In: *CloudCom 2010, IUPUI Conference Center*. Indianapolis. 30 November–3 December 2010.
57. Malewicz G, Austern MH, Bik A, et al. Pregel: A System for Large-Scale Graph Processing. In: *International conference on management of data*. Indianapolis, Indiana. 2010;135–146.
58. Bu Y, Howe B, Balazinska M, Ernst MD. HaLoop: Efficient Iterative Data Processing on Large Clusters. In: *The 36th International Conference on Very Large Data Bases, VLDB Endowment*. Vol. 3, Singapore. 13–17 September 2010.
59. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster Computing with Working Sets. In: *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)*. Boston. 22 June 2010.
60. SALSA Group. Iterative MapReduce. In: <http://www.iterativemapreduce.org/>; 2010.
61. Yahoo. Yahoo! Hadoop Tutorial. In:  
<http://developer.yahoo.com/hadoop/tutorial/index.html>; 2010.
62. Coulouris G, Dollimore J, Kindberg T. *Distributed Systems: Concepts and Design* International Computer Science Series 4th ed. Addison-Wesley 2004.
63. White T. *Hadoop: The Definitive Guide* Second ed. Yahoo Press 2010.
64. Apache. HDFS Overview. In: <http://hadoop.apache.org/hdfs/>; 2010.
65. Apache. Hadoop MapReduce. In:  
<http://hadoop.apache.org/mapreduce/docs/current/index.html>; 2010.
66. Venner J. *Pro Hadoop* first ed. Apress 2009; ISBN:978-1430219422.
67. Apache! Pig! (part of Hadoop). In: <http://pig.apache.org/>; 2010.
68. Choudhary A, Fox G, Hiranandani S, et al. Unified compilation of Fortran 77D and 90D. *ACM Lett Program Lang Syst*. 1993;2(1–4):95–114.
69. Yahoo, Pig! Tutorial.  
<http://developer.yahoo.com/hadoop/tutorial/module6.html#pig>.
70. Systems@ETH Zurich, Massively Parallel Data Analysis with MapReduce. Lectures on MapReduce, Hadoop and Pig Latin.  
<http://www.systems.ethz.ch/education/past-courses/hs08/map-reduce/map-reduce/lecture-slides>, 2008 (accessed 07.11.10).
71. Fox G, Williams RD, Messina PC. *Parallel computing works!* Morgan Kaufmann Publishers 1994.
72. Ekanayake J, Gunarathne T, Qiu J, et al. Applicability of DryadLINQ to Scientific Applications, Community Grids Laboratory, Indiana University. In:  
<http://grids.ucs.indiana.edu/ptliupages/publications/DryadReport.pdf>; 2010.
73. Qiu J, Ekanayake J, Gunarathne T, et al. Data Intensive Computing for Bioinformatics. In:  
[http://grids.ucs.indiana.edu/ptliupages/publications/DataIntensiveComputing\\_BookChapter.pdf](http://grids.ucs.indiana.edu/ptliupages/publications/DataIntensiveComputing_BookChapter.pdf); 2009.
74. Ekanayake J, Gunarathne T, Qiu J. Cloud Technologies for Bioinformatics Applications. *IEEE Trans Parallel Distrib Syst*. 2010.

75. Qiu J, Gunarathne T, Ekanayake J, et al. Hybrid Cloud and Cluster Computing Paradigms for Life Science Applications. In: *11th Annual Bioinformatics Open Source Conference BOSC*. Boston. 9–10 July 2010.
76. Burrows M. The Chubby Lock Service for Loosely-Coupled Distributed Systems. In: *OSDI'06: Seventh Symposium on Operating System Design and Implementation*. Seattle, WA: USENIX; 2006;335–350.
77. Fox GC, Ho A, Chan E, Wang W. Measured characteristics of distributed cloud computing infrastructure for message-based collaboration applications. In: *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems*. IEEE Computer Society 2009;465–467.
78. DeCandia G, Hastorun D, Jampani M, et al. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper Syst Rev*. 2007;41(6):205–220.
79. Eucalyptus LLC, White Papers. <http://www.eucalyptus.com/whitepapers>.
80. Nurmi D, Wolski R, Grzegorczyk C, et al. The Eucalyptus Open-Source Cloud-Computing System. In: *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Shanghai: CCGRID '09; 18–21 May 2009;124–131.
81. Keahey K, Foster I, Freeman T, Zhang X. Virtual workspaces: achieving quality of service and quality of life in the Grid. *Scientific Prog J*. 2005;13(4):265–275.
82. Nimbus. Cloud computing for science. In: <http://www.nimbusproject.org>; 2010.
83. Nimbus. Frequently Asked Questions. In: <http://www.nimbusproject.org/docs/current/faq.html>; 2010.
84. Django. High-Level Python Web Framework. In: <http://www.djangoproject.com/>; 2010.
85. Amazon. Simple Storage Service API Reference: API Version 2006-03-01. In: <http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf>; 2006.
86. boto. Python interface to Amazon Web Services. In: <http://code.google.com/p/boto/>; 2010.
87. S3tools project. Open source tools for accessing Amazon S3 – Simple Storage Service. In: <http://s3tools.org/s3tools>; 2010.
88. bitbucket, JetS3t: open-source Java toolkit and application suite for Amazon Simple Storage Service (Amazon S3), Amazon CloudFront content delivery network, and Google Storage. <http://bitbucket.org/jmurty/jets3t/wiki/Home>.
89. Amazon, Amazon Elastic Compute Cloud (Amazon EC2). <http://aws.amazon.com/ec2>.
90. OpenNebula, industry standard open source cloud computing tool. <http://opennebula.org/>.
91. Llorente IM, Moreno-Vozmediano R, Montero RS. Cloud Computing for On-Demand Grid Resource Provisioning. In: IOS Press 2009;177–191. *Advances in Parallel Computing: High Speed and Large Scale Scientific Computing*. vol. 18.
92. Montero R, Moreno-Vozmediano R, Llorente I. An elasticity model for high throughput computing clusters. *J Parallel Distrib Comput*. 2010.
93. Sotomayor B, Montero RS, Llorente IM, Foster I. Capacity Leasing in Cloud Systems Using the OpenNebula Engine. In: *2008 Workshop on Cloud Computing*

*and Its Applications* (CCA08). Chicago, IL. 2008; In:

<http://www.cca08.org/papers/Paper20-Sotomayor.pdf>; 2008.

94. libvirt, virtualization API. <http://libvirt.org>.
95. Sotomayor B, Montero RS, Llorente IM, Foster I. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Comp.* 2009;13(5):14–22.
96. ElasticHosts, Flexible servers in the cloud. <http://www.elastichosts.com/>.
97. Sector/Sphere, High Performance Distributed File System and Parallel Data Processing Engine. <http://sector.sourceforge.net>.
98. Gu Y, Grossman R. Sector/Sphere: A Distributed Storage and Computing Platform SC08 Poster. In: <http://sector.sourceforge.net/pub/sector-sc08-poster.pdf>; 2008.
99. Gu Y, Grossman R. Lessons learned from a year's worth of benchmarks of large data clouds. In: *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*. Portland, Oregon: ACM; 2009;1–6.
100. Gu Y, Grossman R. UDT: UDP-based data transfer for high-speed wide area networks. *Comput Netw.* 2007;51(7):1777–1799.
101. FUSE, Filesystem in Userspace. <http://fuse.sourceforge.net>.
102. Gu Y, Grossman RL. Sector and sphere: the design and implementation of a high-performance data cloud. *Phil Trans R Soc A.* 2009;367:2429–2445.
103. Open Stack. Open Source, Open Standards Cloud. In: <http://openstack.org/index.php>; 2010.
104. VSCSE. Big Data for Science Virtual Summer School hosted by SALSA group at Indiana University. In: <http://salsahpc.indiana.edu/tutorial/>; 26 July–30 July 2010.
105. Hey T. The Fourth Paradigm: Data-Intensive Scientific Discovery. In: <http://research.microsoft.com/en-us/um/redmond/events/TonyHey/21216/player.htm>; 2010.
106. Hey T, Tansley S, Tolle K. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. <http://research.microsoft.com/en-us/collaboration/fourthparadigm/>; 2009; (accessed 07.11.10).
107. SALSA Group. Catalog of Cloud Material. In: <http://salsahpc.indiana.edu/content/cloud-materials>; 2010.
108. Microsoft Research. Cloud Futures Workshop. In: <http://research.microsoft.com/en-us/events/cloudfutures2010/default.aspx>; 2010.
109. Chou T. *Introduction to Cloud Computing: Business and Technology* Active Book Press, LLC 2010; p. 252,  
[http://www.lulu.com/items/volume\\_67/8215000/8215197/1/print/8215197.pdf](http://www.lulu.com/items/volume_67/8215000/8215197/1/print/8215197.pdf).
110. Buyya R, Yeo C, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Gener Comput Syst.* 2009;25(6):599–616.
111. Chaganti P. Cloud computing with Amazon Web Services, Part 1: Introduction – When it's smarter to rent than to buy. In:  
<http://www.ibm.com/developerworks/architecture/library/ar-cloudaws1/>; 2008.
112. Cloud computing with Amazon Web Services. Part 2: Storage in the cloud with Amazon Simple Storage Service (S3) – Reliable, flexible, and inexpensive storage

and retrieval of your data. In:

<http://www.ibm.com/developerworks/architecture/library/ar-cloudaws2/>; 2008.

113. Chaganti P. Cloud computing with Amazon Web Services, Part 3: Servers on demand with EC2. In: <http://www.ibm.com/developerworks/architecture/library/ar-cloudaws3/>; 2008.
114. Palankar MR, Iamnitchi A, Ripeanu M, Garfinkel S. Amazon S3 for science grids: a viable solution? In: *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*. Boston, MA: ACM; 2008;55–64.
115. Sun W, Zhang K, Chen S, Zhang X, Liang H. Software as a Service: An Integration Perspective. In: Vienna Austria: Springer Verlag; 2007;558–569. *Fifth International Conference Service-Oriented Computing – ICSOC, Lecture Notes in Computer Science*. Vol. 4749.
116. Lakshmanan G. Cloud Computing Relevance to Enterprise. In:  
<http://www.infosys.com/cloud-computing/white-papers/Documents/relevance-enterprise.pdf>; 2009.
117. Leavitt N. Is cloud computing really ready for prime time? *Computer*. 2009;42(1):15–20.
118. Lin G, Dasmalchi G, Zhu J. Cloud Computing and IT as a Service: Opportunities and Challenges. In: *Web Services, ICWS '08*. Beijing: IEEE; 23–26 September 2008.
119. Linthicum DS. *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide* Addison-Wesley Professional 2009.
120. Mei L, Chan WK, Tse TH. A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues. In: *Asia-Pacific Services Computing Conference. APSCC '08*. Yilan, Taiwan: IEEE; 9–12 December 2008;464–469.
121. Fox G, Bae S, Ekanayake J, Qiu X, Yuan H. *Parallel Data Mining from Multicore to Cloudy Grids, book chapter of High Speed and Large Scale Scientific Computing* Amsterdam: IOS Press; 2009;  
<http://grids.ucs.indiana.edu/ptliupages/publications/CetraroWriteupJune11-09.pdf>.
122. Ekanayake J, Fox G. High Performance Parallel Computing with Clouds and Cloud Technologies. In: *First International Conference CloudComp on Cloud Computing*. Munich, Germany 2009.
123. Chang F, Dean J, Ghemawat S, et al. BigTable: a distributed storage system for structured data. *ACM Trans Comput Syst*. 2008;26(2):1–26.
124. Deelman E, Singh G, Livny M, Berriman B, Good J. The cost of doing science on the cloud: the Montage example. In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. Austin, Texas: IEEE Press; 2008;1–12.  
<http://www.csd.uwo.ca/faculty/hanan/cs843/papers/ewa-ec2.pdf>.
125. Hoffa C, Mehta G, Freeman T, et al. On the Use of Cloud Computing for Scientific Workflows. In: *Proceedings of the 2008 Fourth IEEE International Conference on eScience*. IEEE Computer Society 2008;640–645.
126. Paton N, Marcelo A, De Aragão T, Lee K, Alvaro A, Sakellariou R. Optimizing utility in cloud computing through autonomic workload execution, *Bulletin of the*

IEEE Computer Society Technical Committee on Data Engineering. In:  
<http://www.cs.man.ac.uk/~alvaro/publications/TCDEBull09.pdf>; 2009.

127. Rochwerger B, et al. The Reservoir model and architecture for open federated cloud computing. *IBM J Res Dev.* 2009;53(4):4:1–4:11.

## Homework Problems

### Problem 6.1

Visit the Google App Engine (GAE) web site, download the SDK, and read the Python Guide or the Java Guide to get started. Note that GAE only accepts Python, Ruby, and Java programming languages. The platform does not provide any IaaS services.

- a. Develop a specific cloud application using available software services such as Gmail, Docs, or CRM on the GAE platform. Test your applications on the GAE platform.
- b. Report your application development experience and experimental results in terms of some selected performance measures such as job queuing time, execution time, resource utilization rate, or some QoS attributes such as goal achievement, success rate, fault tolerance, and cost-effectiveness.
- c. Change the problem size or data set size and platform configuration to study the scalability and efficiency issues in your GAE experiments.

### Problem 6.2

Write an application code to run on the GAE platform for backup storage of a large amount of your personal, family, or company data and records such as photos, video, music, sales receipts, documents, news media, inventory, market records, financial matters, supply chain information, human resources, public data sets, and so forth. Note that strict privacy protection is required here. Minimizing the storage cost is another objective function to achieve. The access path, software development tools, and platform information in using GAE can be found in [Chapter 4](#). You should explain your code development experience and report the results in using the GAE platform.

### Problem 6.3

Get set up to use Amazon Web Services; perhaps use the capability Amazon provides at <http://aws.amazon.com/education/> for instructors to obtain details to arrange free education accounts for their students. Test the AWS platform for a SimpleDB application using the relational database service. This application could be similar to that chosen for [Problems 6.1](#) and [6.2](#). You should study the system's performance, scalability, throughput, efficiency, resource utilization, fault tolerance, and cost-effectiveness.

### Problem 6.4

Design and request an EC2 configuration on the AWS platform for parallel

multiplication of two very large matrices with an order exceeding 50,000.

- a. Report your experimental results including execution time, speed performance, VM instances initiated, compute unit and storage utilization, and service charge experienced.
- b. You can also study related issues such as scalability, throughput, efficiency, resource utilization, fault tolerance, and cost-effectiveness in this scientific cloud experiment.

## Problem 6.5

Repeat [Problem 6.4](#) on the AWS platform for an Amazon S3 application. This application could be similar to that chosen for [Problems 6.1–6.2](#). You can also study the performance, scalability, throughput, efficiency, resource utilization, fault tolerance, cost-effectiveness, and related issues.

## Problem 6.6

Study some reported EC2 or S3 applications on the AWS platform by business or service industry or by large enterprises. Many examples or success stories are told in Chou's book [\[109\]](#). For example, Vertica Systems applies EC2 in DBMS (database management system) applications. Eli Lilly uses EC2 in drug development. Animoto offers an online service to facilitate personal video production. Contact a few of those service companies for technical implementation and service details. Submit a study report with suggestions on improving the selected compute and storage service applications.

## Problem 6.7

Visit the Microsoft Windows Azure Developer Center. You can download the Azure development kit to run a local version of Azure. Design an application experiment and test it on a local computer, such as your desktop, or a notebook computer, or a university workstation or server. Report your experiment experiences in using the Azure platform.

## Problem 6.8

In the MapReduce programming model, there is a special case concerning implementing only the map phase, which is also known as a "map-only" problem. This achievement can enhance existing applications/binaries to have high throughput by running them in a parallel fashion; in other words, it helps stand-alone programs to utilize large-scale computing capability. The goal of this exercise is to write a Hadoop "map-only" program with the bioinformatics application BLAST (NCBI BLAST+; <ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.2.23/>) under a Linux/UNIX environment using Twister.

Basically, the BLAST binary is invoked within the scope of a "map-only" program on each map task with assigned input. Here, the assigned input is the map task key-value pair. It may include content of the original input, a path to the input file, or any other

metadata regarding the nature of the binary. Then, each map task passes this input to the external binary during runtime. This “map-only” program terminates after all the inputs have been processed.

```
MapOnly (<key, value>) {  
    input -> keyValueModification (<key, value>);  
    output -> invokeBlastBinary(input);  
    return finish  
}
```

With the given pseudocode, you need to implement the *map()* interface within *RunnerMap.java* ([http://salsahpc.indiana.edu/tutorial/source\\_code/Hadoop-Blast-sketch.zip](http://salsahpc.indiana.edu/tutorial/source_code/Hadoop-Blast-sketch.zip)), which executes an external Java process to run the stand-alone BLAST for each map task with the assigned input query. The “input.fa” needs to be split into chunks (*chunk[1-n].fa*) as a set of input queries. This set of queries is stored on the local disk; you may need to upload them to HDFS before the Hadoop BLAST program executes. The HDFS directory name is one of the program parameters. Then, the Hadoop framework assigns an input query, formed as a <key,value> pair such as <filename,file path on HDFS>, to each worker.

The code of <key,value> generation is done if you use the provided *DataFileInputFormat.java* and *FileRecordReader.java* files as the *InputFormatClass*. The BLAST binary requires a database to find matching hits regarding the input queries. This database can be either stored in a shared file system or located on the local disk using distributed cache; it depends on how you write the program. Either one of the solutions is acceptable. For the final step, collect the result from each mapper and merge them into a single output result.fa file. In addition, select a proper version of the BLAST+ binary (32-bit or 64-bit) regarding the Linux OS version (i686/i386 or x86\_64). For further suggestions, see the SalsaHPC Hadoop BLAST Tutorial with the pseudocode (<http://salsahpc.indiana.edu/tutorials/hadoopblast.html>).

## Problem 6.9

Visit the Manjrasoft Aneka Software Center at [www.manjrasoft.com/](http://www.manjrasoft.com/). Download the Aneka software and create an enterprise cloud using LAN-connected PCs in your university/student laboratory and run various example applications that come with Aneka. Write simple parallel programs using Aneka’s Task, Thread, and MapReduce programming models and carry out their execution on workers ranging from 2 to 20 in steps of 2.

## Problem 6.10

Repeat applications in Problems 6.1–6.5 using the academic/open source packages described in Section 6.5—namely Eucalyptus, Nimbus, OpenStack, OpenNebula, and Sector/Sphere. This software is available on the FutureGrid web site (<http://www.futuregrid.org>) and includes a growing number of tutorials.

## Problem 6.11

Test the large-scale matrix multiplication program of [Problem 6.4](#) on two or three cloud platforms (GAE, AWS, and/or Azure). You can also choose a data-intensive application such as a large-scale search or business processing application. Implement the application on at least two or all three cloud platforms, separately. The major objective is to minimize the execution time of the application. The minor objective is to minimize the user service costs.

- a. Run the service on the GAE platform.
- b. Run the service on the AWS platform.
- c. Run the service on the Windows Azure platform.
- d. Compare your compute and storage costs, design experiences, and experimental results on all three cloud platforms. Report the relative performance and QoS results measured.

## Problem 6.12

There are three implementations of the MapReduce engine and its extensions: Google MapReduce, Apache Hadoop, and Microsoft Dryad. Complete the missing 14 entries to show their similarities and differences in 6 technical dimensions. 4 entries are already given below as example answers. You do not have to explain the table entries in detail, just enter the names of languages, models, methods, mechanisms, and platforms applied.

Programming Environment	Google MapReduce	Apache Hadoop MapReduce	Microsoft Dryad
Coding language and programming model used			
Mechanisms for data handling			Shared directories and local disks
Failure handling methods	Re-execute failed tasks and duplicated execution of the slow tasks		
High-level language for data analysis			
OS and cluster environment			Windows HPCS cluster
Intermediate data transfer method		By file transfer or using the http links	

## Problem 6.13

Using a MapReduce programming model supported in Aneka, develop a program for image filtering of hundreds of photos you have taken using a digital camera. Conduct and report the results of your scalability experiments by varying the number of computing nodes/workers and images of different resolutions or file size on an Aneka-based enterprise cloud.

## Problem 6.14

Examine the tutorials at <http://www.futuregrid.org/tutorials> and compare the word count application on Hadoop using Eucalyptus, Nimbus, and at least one other of OpenStack, OpenNebula, or Sector/Sphere.

## Problem 6.15

Examine the tutorials at <http://salsahpc.indiana.edu> and <http://www.iterativemapreduce.org/samples.html>. Compare Hadoop and Twister on cases specified by the instructor from examples given there. Discuss their relative strengths and weaknesses.

## Problem 6.16

The following program is written in Hadoop, namely WebVisCounter. Trace through the program or run it on a cloud platform to which you have access. Analyze the programming tasks performed by this Hadoop program and learn from using Hadoop library.

```
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
static class OSCountMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        Text UserInfo = new Text();
        Text OSversion = new Text();
        int StartIndex, EndIndex, i;
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens() && i != 8) {
            i++; UserInfo.set(tokenizer.nextToken());
        }
        i = 0;
        while (UserInfo.charAt(i) != ';') {
            if (UserInfo.charAt(i) != '('){StartIndex = i}
```

```

    i++ );
EndIndex = i;
OSversion = UserInfo.substring(StartIndex, EndIndex);
output.collect(OSversion, one) } } ;
static class OSCountReduce
extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values,
Context
context){ int sum = 0;
    while (values.hasNext()) { sum += values.next().get() } ;
    output.collect(key, new IntWritable(sum)); } ;
}

public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("OSCount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));
    JobClient.runJob(conf); } ;
}

```

## Problem 6.17

Twister K-means extends the MapReduce programming model iteratively. Many data analysis techniques require iterative computations. For example, K-means clustering is the application where multiple iterations of MapReduce computations are necessary for overall computation. Twister is an enhanced MapReduce runtime that supports iterative MapReduce computations efficiently. For this problem, study the iterative MapReduce programming model and learn how to implement the K-means algorithm with Twister.

## Problem 6.18

DryadLINQ PageRank is a well-known link analysis algorithm. It calculates numerical values to each element of a hyperlinked set of web pages, which reflects the probability that the random surfer will access that page. Implementing PageRank with MapReduce is

of some difficulty in both efficiency and programmability due to both the iterative structure and the random access model in large-scale web graphs. DryadLINQ provide the SQL-like queries API that helps programmers implement PageRank without much effort. In addition, the Dryad infrastructure helps to scale the application out in an easy way. Learn how to implement a simple PageRank application with DryadLINQ.

## Problem 6.19

Using a thread programming model supported in Aneka, develop a program for parallel multiplication of two very large square matrices on an order greater than 500. Conduct and report the results of your scalability experiments by varying the order of the matrix from 500 to 1,000 in steps of 100 on computing nodes/workers varying from 10 to 50 in steps of 10 in an Aneka-based enterprise cloud.

## Problem 6.20

Compare Pig Latin and “bare” Hadoop for implementing data intensive application specified by your instructor. Discuss their relative advantages and shortcomings.

# PART 3

# Grids, P2P, and the Future Internet

## OUTLINE

---

Introduction

Chapter 7 Grid Computing Systems and Resource Management

Chapter 8 Peer-to-Peer Computing and Overlay Networks

Chapter 9 Ubiquitous Clouds and the Internet of Things

# Introduction

---

Three chapters in Part 3 look into the future trends of network-based distributed computing. [Chapter 7](#) covers computational grids and data grids, including major national grids built in the US, Europe, and China. We study peer-to-peer (P2P) overlay networks and examine real P2P systems built in recent years. Finally, we present sensor networks and radio-frequency identification (RFID) and global positioning systems (GPS) for building the Internet of Things (IoT). Social networks are assessed in web-scale service applications. We examine ubiquitous grids, P2P systems, and clouds for future Internet applications.

# Chapter 7: Grid Computing Systems and Resource Management

We study the design principles, platform architecture, middleware support, resource management, and service standards in computing/data grids. A dozen grid architectures and their software environments are reviewed, including the TeraGrid, EGEE, DataGrid, ChinaGrid, BOINC, and Grid'5000, among others. Several grid middleware packages: Condor-G, SGE, Globus Toolkit, and CGSP are detailed. We also assess security infrastructure needed for trusted grid computing. We present an adaptive method for workload characterization and performance prediction in large-scale grid computing. Past and current grid projects are assessed against the emerging technologies.

This chapter is authored by Yongwei Wu and Kai Hwang. Rajkumar Buyya has contributed [Section 7.3.4](#). Technical assistance was extended from Weimin Zheng and Zhiwei Xu.

## Chapter 8: Peer-to-Peer Computing and Overlay Networks

P2P overlay networks are built over a large number of client computers through virtual networking over the Internet. We present unstructured, structured, and hybrid overlay networks. Distributed hash tables (DHTs) are studied for fast routing and locality preservation operations. Fault tolerance and churn resilience are studied to improve dependability. P2P networks are used for distributed file sharing and content distribution applications. Trust management and copyright protection are studied with reputation systems and content poisoning techniques. Examples illustrate the P2P design principles and the protocols applied.

This chapter is authored by Zhenyu Li and Kai Hwang with technical assistance from Gaogang Xie of Chinese Academy of Sciences.

# Chapter 9: Ubiquitous Clouds and the Internet of Things

The future trends of clouds and Internet computing are assessed in this final chapter. We evaluate system performance and assure security of open network systems. We examine some public and private clouds built by IBM, Salesforce, SGI, Manijarsoft, NASA, and CERN. We report some benchmark results and representative applications of grids and clouds and their variants. We study the Internet of Things with the enabling technologies in sensing networks, RFID tracking and global positioning systems. We assess web-scale social networks and their impacts on the future Internet to be populated with inter-clouds, coordinated data centers, and large-scale social services.

This chapter is primarily authored by Kai Hwang with partial contributions by Judy Qiu ([Sections 9.2.1](#) and [9.2.5](#)). The material development was also assisted by Vikram Dixit, Zhongyuan Qin, and Kaikun Dong at the University of Southern California.

# CHAPTER 7

# Grid Computing Systems and Resource Management

## CHAPTER OUTLINE

### Summary

#### 7.1 Grid Architecture and Service Modeling

- 7.1.1 Grid History and Service Families
- 7.1.2 CPU Scavenging and Virtual Supercomputers
- 7.1.3 Open Grid Services Architecture (OGSA)
- 7.1.4 Data-Intensive Grid Service Models

#### 7.2 Grid Projects and Grid Systems Built

- 7.2.1 National Grids and International Projects
- 7.2.2 NSF TeraGrid in the United States
- 7.2.3 DataGrid in the European Union
- 7.2.4 The ChinaGrid Design Experiences

#### 7.3 Grid Resource Management and Brokering

- 7.3.1 Resource Management and Job Scheduling
- 7.3.2 Grid Resource Monitoring with CGSP
- 7.3.3 Service Accounting and Economy Model
- 7.3.4 Resource Brokering with Gridbus

#### 7.4 Software and Middleware for Grid Computing

- 7.4.1 Open Source Grid Middleware Packages
- 7.4.2 The Globus Toolkit Architecture (GT4)
- 7.4.3 Containers and Resources/Data Management
- 7.4.4 The ChinaGrid Support Platform (CGSP)

#### 7.5 Grid Application Trends and Security Measures

- 7.5.1 Grid Applications and Technology Fusion
- 7.5.2 Grid Workload and Performance Prediction
- 7.5.3 Trust Models for Grid Security Enforcement
- 7.5.4 Authentication and Authorization Methods
- 7.5.5 Grid Security Infrastructure (GSI)

## 7.6 Bibliographic Notes and Homework Problems

### Acknowledgments

### References

### Homework Problems

This chapter is devoted to building grid systems for distributed computing over an integrated network of supercomputers, server clusters, and data centers. First we examine the open grid service architecture. We study a dozen grid architectures and their software environments including the TeraGrid, EGEE, DataGrid, ChinaGrid, BOINC, and Grid'5000, etc. These case studies illustrate grid design principles, service models, and implementation technologies. Several grid middleware packages: Condor-G, SGE, Globus Toolkit, and CGSP are detailed. Finally, we cover grid trends and security infrastructure. Past and current grid projects are reviewed against emerging technologies.

grids, Open Grid Service Architecture (OGSA), national grids, resource management, Globus Toolkit, trust models, security

## Summary

This chapter is devoted to building grid systems for distributed computing over an integrated network of supercomputers, server clusters, and data centers. First we examine the open grid service architecture. We study a dozen grid architectures and their software environments including the TeraGrid, EGEE, DataGrid, ChinaGrid, BOINC, and Grid'5000, etc. These case studies illustrate grid design principles, service models, and implementation technologies. Several grid middleware packages: Condor-G, SGE, Globus Toolkit and CGSP are detailed. Finally, we cover grid trends and security infrastructure. Past and current grid projects are reviewed against emerging technologies.

## 7.1 Grid Architecture and Service Modeling

The grid is a metacomputing infrastructure that brings together computers (PCs, workstations, server clusters, supercomputers, laptops, notebooks, mobile computers, PDAs, etc.) to form a large collection of compute, storage, and network resources to solve large-scale computation problems or to enable fast information retrieval by registered users or user groups. The coupling between hardware and software with special user applications is achieved by leasing the hardware, software, middleware, databases, instruments, and networks as computing utilities. Good examples include the renting of expensive special-purpose application software on demand and transparent access to human genome databases.

The goal of grid computing is to explore fast solutions for large-scale computing problems. This objective is shared by computer clusters and *massively parallel processor* (MPP) systems we have learned about in [Chapter 2](#). However, grid computing takes

advantage of the existing computing resources scattered in a nation or internationally around the globe. In grids, resources owned by different organizations are aggregated together and shared by many users in collective applications. Grids rely heavily on the use of LAN/WAN resources across enterprises, organizations, and governments. The *virtual organizations* or *virtual supercomputers* are new concepts derived from grid or cloud computing. These are virtual resources dynamically configured and are not under the full control of any single user or local administrator.

## 7.1.1 Grid History and Service Families

Network-based distributed computing becomes more and more popular among the Internet users. Recall that the Internet was developed in the 1980s to provide computer-to-computer connections using the *telnet://* protocol. The web service was developed in the 1990s to establish direct linkage between web pages using the *http://* protocol. Ever since the 1990s, grids became gradually available to establish large pools of shared resources. The approach is to link many Internet applications across machine platforms directly in order to eliminate isolated resource islands. We may invent upgraded protocols in the future like "*grid://*" and "*cloud://*" to realize this dream of a socialized cyberspace with greater resource sharing.

The idea of the grid was pioneered by Ian Foster, Carl Kesselman and Steve Tuecke in a 2001 paper [16]. With this ground work, they are often recognized as the fathers of the grids. The Globus Project supported by DARPA has promoted the maturity of grid technology with a rich collection of software and middleware tools for grid computing. In 2007, the concept of cloud computing was thrown out, which in many ways was extending grid computing through virtualized data centers. In this beginning section, we introduce major grid families and review the grid service evolution over the past 15 years.

Grids differ from conventional HPC clusters. Cluster nodes are more homogeneous machines that are better coordinated to work collectively and cooperatively. The grid nodes are heterogeneous computers that are more loosely coupled together over geographically dispersed sites. In 2001, *Forbes Magazine* advocated the emergence of the *great global grid (GGG)* as a new global infrastructure. This GGG evolved from the *World Wide Web (WWW)* technology we have enjoyed for many years. Four major families of grid computing systems were suggested by the *Forbes* GGG categorization as summarized in [Table 7.1](#). We will study some of the listed grids in subsequent sections.

**Table 7.1**

### Four Grid Families Identified in the Great Global Grid (GGG) (*Forbes*, 2001)

Grid Family	Representative Grid Systems and References
Computational Grids or Data Grids	TeraGrid (US), EGEE (EU), DataGrid (EU), Grid'5000 (France), ChinaGrid (China), NAS (NASA), LCG (CERN), e-Science (UK), D-Grid (Nordic), FutureGrid (US), etc.
Information Grids or Knowledge Grids	Semantic Grid, Ontology Platform, BOINC (Berkeley), D4Science, Einstein@Home, Information Power Grid (NASA)

Business Grids	BEinGrid (EU), HP eSpeak, IBM WebSphere, Sun Grid Engine, Microsoft .NET, etc.
P2P/Volunteer Grids	SETI@Home, Parasic Grid, FightAIDS@Home, Foldong@Home, GIMPS, etc.

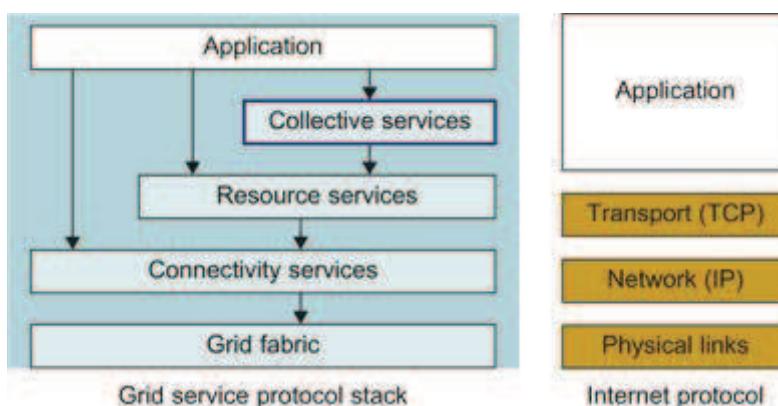
### 7.1.1.1 Four Grid Service Families

Most of today's grid systems are called *computational grids* or *data grids*. Good examples are the NSF TeraGrid installed in the United States and the DataGrid built in the European Union. *Information* or *knowledge grids* post another grid class dedicated to knowledge management and distributed ontology processing. The Semantic web, also known as *semantic grids*, belongs to this family. Ontology platform falls into information or knowledge grids. Other information/knowledge grids include the Berkeley BOINC and NASA's Information Power Grid.

In the business world, we see a family, called *business grids*, built for business data/information processing. These are represented by the HP eSpeak, IBM WebSphere, Microsoft .NET, and Sun One systems. Some business grids are being transformed into *Internet clouds*. The last grid class includes several grid extensions such as *P2P grids* and parasitic grids. This chapter will concentrate mainly in computational or data grids. Business grids are only briefly introduced. We will further compare grids with P2P networks and P2P grids in [Chapter 8](#).

### 7.1.1.2 Grid Service Protocol Stack

To put together the resources needed in a grid platform, a layered grid architecture is shown in [Figure 7.1](#). The top layer corresponds to user applications to run on the grid system. The user applications demand collective services including collective computing and communications. The next layer is formed by the hardware and software resources aggregated to run the user applications under the collective operations. The connectivity layer provides the interconnection among drafted resources. This connectivity could be established directly on physical networks or it could be built with virtual networking technology.



**FIGURE 7.1** The layered grid service protocols and their relationship with the Internet service protocols. Courtesy of Foster, Kesselman, and Tuecke [11]

The connectivity must support the grid fabric, including the network links and virtual

private channels. The fabric layer includes all computational resources, storage systems, catalogs, network resources, sensors, and their network connections. The connectivity layer enables the exchange of data between fabric layer resources. The five-layer grid architecture is closely related to the layered Internet protocol stack shown on the right-hand side of [Figure 7.1](#). The fabric layer corresponds to the link layer in the Internet stack. The connectivity layer is supported by the network and transport layers of the Internet stack. The Internet application layer supports the top three layers.

### 7.1.1.3 Grid Resources

[Table 7.2](#) summarizes typical resources that are required to perform grid computing. Many existing protocols (IP, TCP, HTTP, FTP, and DNS) or some new communication protocols can be used to route and transfer data. The resource layer is responsible for sharing single resources. An interface is needed to claim the static structure and dynamic status of local resources. The grid should be able to accept resource requests, negotiate the *Quality of Service* (QoS), and perform the operations specified in user applications.

**Table 7.2**

#### Control Operations and Enquiries for Aggregating Grid Resources

Resources	Control Operations	Enquiries
Compute resources	Starting, monitoring, and controlling the execution of resultant processes; control over resources: advance reservation	Hardware and software characteristics; relevant load information: current load and queue state
Storage resources	Putting and getting files; control over resources allocated to data transfers: advance reservation	Hardware and software characteristics; relevant load information: available space and bandwidth utilization
Network resources	Control over resources allocated	Network characteristics and load
Code repositories	Managing versioned source and object code	Software files and compile support
Service catalogs	Implementing catalog query and update operations: a relational database	Service order information and agreements

The collective layer handles the interactions among a collection of resources. This layer implements functions such as resource discovery, co-allocation, scheduling, brokering, monitoring, and diagnostics. Other desired features include replication, grid-enabled programming, workload management, collaboration, software discovery, access authorization, and community accounting and payment. The application layer comprises mainly user applications. The applications interact with components in other layers by using well-defined APIs (*application programming interfaces*) and SDKs (*software development kits*).

### 7.1.2 CPU Scavenging and Virtual Supercomputers

In this section, we characterize the process of grid resource aggregation from local and remote sources. Then we link the grid to the concept of virtual organizations in a

dynamic sense. In fact, the distinction between grids and clouds becomes blurred in recent years. Traditionally, grids were formed with allocated resources statically, while clouds were formed with provisioned resources dynamically. As virtualization is applicable to grid components, some grids involving data centers become more like clouds.

Foster, et al. [15] have compared the grid problem with the anatomy problem in biology. The application users expect grids to be designed as flexible, secure, and coordinated resources shared by individuals, institutions, and virtual organizations. The grid resources could come from two possible sources. On the one hand, large-scale HPC grids can be formed with computers from resource-rich supercomputer centers owned by government agencies and research institutions. Alternatively, one could form “*virtual*” grids, casually, out of a large number of small commodity computers owned by ordinary citizens, who volunteer to share their free cycles with other users for a noble cause.

### **7.1.2.1 CPU Scavenging and Virtual Supercomputers**

Both public and virtual grids can be built over large or small machines, that are loosely coupled together to satisfy the application need. Grids differ from the conventional supercomputers in many ways in the context of distributed computing. Supercomputers like MPPs in the Top-500 list are more homogeneously structured with tightly coupled operations, while the grids are built with heterogeneous nodes running non-interactive workloads. These grid workloads may involve a large number of files and individual users. The geographically dispersed grids are more scalable and fault-tolerant with significantly lower operational costs than the supercomputers.

The concept of creating a “grid” from the unused resources in a network of computers is known as *CPU scavenging*. In reality, virtual grids are built over large number of desktop computers by using their free cycles at night or during inactive usage periods. The donors are ordinary citizens on a voluntary participation basis. In practice, these client hosts also donate some disk space, RAM, and network bandwidth in addition to the raw CPU cycles. At present, many volunteer computing grids are built using the CPU scavenging model. The most famous example is the SETI@Home [37], which applied over 3 million computers to achieve 23.37 TFlpos as of Sept. 2001. More recent examples include the BOINC [7] and Folding@Home [14], etc. In practice, these virtual grids can be viewed as virtual supercomputers.

#### **Example 7.1**

#### **BOINC: Berkeley Open Infrastructure for Network Computing**

BOINC is an open infrastructure for running volunteer computing projects. It is an active grid research project conducted at the University of California, Berkeley (<http://boinc.berkeley.edu/>). The BOINC offers an open-source

software for volunteer computing and grid computing. The idea is to use idle times on participants' computers collectively for noble causes or for academic research purposes. These user hosts run with the popular Windows, Mac, or Linux OS to execute many useful applications to cure diseases, study global warming, discover pulses and do many other meaningful scientific research projects.

As reported in April 7, 2011, there were 321,471 volunteers in the BOINC project contributing 516,440 computers. With a 24-hour average, 5.634 PFlops was achieved collectively. Grid computing with BOINC may involve scientists, universities, and companies collectively. At present, the BOINC project covers the following public grid services: message boards, email lists, papers and talks, help wanted, software development, etc. As of 2011, over 6.2 million machines running the BOINC platform are members of the World Community Grid, which exceeded the fastest supercomputer (Tianhe-1A) in 2011. Similarly, the Folding@Home project has reported 5 PFlops performance over 350,000 machines, as of March 7, 2009 [14].

### 7.1.2.2 Grid Resource Aggregation

During the resource aggregation process for grids or clouds, several assumptions are made. First, the compute nodes and other necessary resources for grids do not join or leave the system incidentally, except when some serious faults occur in the grid. Second, cloud resources are mostly provisioned from large data centers. Since security and reliability are very tight in these data centers, resource behavior is not predictable. Third, although resources in P2P systems are casually allocated, we can build P2P grids for distributed file sharing, content delivery, gaming, and entertainment applications. The joining or leaving of some peers has little impact on the needed functions of a P2P grid system.

We envision the grid resource aggregation process in a global setting. Hardware, software, database, and network resources are denoted by R's and are scattered all over the world. The availability and specification of these open resources is provided by *Grid Information Service (GIS)* agencies. The grid resource brokers assist users with fees to allocate available resources. Multiple brokers could compete to serve users. Also, multiple GISes may overlap in their resource coverage. New grid applications are enabled after the coupling of computer databases, instruments, and human operators needed in their specific applications. It should be noted that today's grid computing applications are no longer restricted to using HPC systems. HTC systems, like clouds, are even more in demand in business services.

### 7.1.2.3 Virtual Organization

The grid is a distributed system integrated from shared resources to form a *virtual*

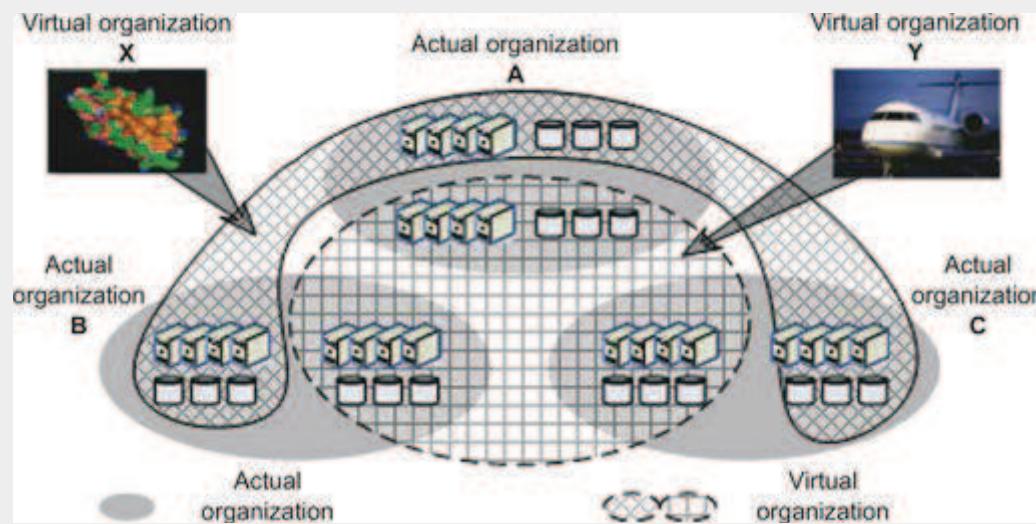
*organization*(VO). The VO offers dynamic cooperation built over multiple physical organizations. The virtual resources contributed by these real organizations are managed autonomously. The grid must deal with the trust relationship in a VO. The applications in a grid vary in terms of workload and resource demand. A flexible grid system should be designed to adapt to varying workloads. In reality, physical organizations include a real company, a university, or a branch of government. These real organizations often share some common objectives.

For example, several research institutes and hospitals may undertake some joint research challenges together to explore a new cancer drug. Another concrete example is the joint venture among IBM, Apple, and Motorola to develop PowerPC processors and their supporting software in the past. The joint venture was based on the VO model. Grids definitely can promote the concept of VOs. Still, joint ventures demand resources and labor from all participants. The following example shows how two VOs or grid configurations can be formed out of three physical organizations.

## Example 7.2

### Dynamic Formation of Virtual Organizations

A new drug development subject needs the molecular simulation software from a research institute and the clinic database from a hospital. At the same time, computer scientists need to perform biological sequence analysis on Linux clusters in a supercomputing center. The members from three organizations contribute all or part of their hardware or software resources to form a VO, as illustrated in [Figure 7.2](#). Here we show how two VOs are formed out of three physical organizations labeled “A,” “B,” and “C.”



**FIGURE 7.2** Two VOs (labeled by X and Y) formed from getting specific resources from three physical organizations (A, B, and C).

Each physical organization owns some resources, such as computers, storage, instruments, and databases. The three organizations cooperate with one another to perform two tasks: new jumbo jet development and new drug exploration. Each physical organization contributes some of its resources to form a VO labeled "X" for drug development and another VO labeled "Y" for aircraft development. Physical organizations may allocate their resources as available to a VO only under light workload conditions.

The owners can also modify the number of cluster servers to be allocated to a specific VO. The participants join or leave the VO dynamically under special service agreement. A new physical organization may also join an existing VO. A participant may leave a VO, once its job is done. The dynamic nature of the resources in a VO poses a great challenge for grid computing. Resources have to cooperate closely to produce a rewarding result. Without an effective resource management system, the grid or VO may be inefficient and waste resources, if poorly managed.

## 7.1.3 Open Grid Services Architecture (OGSA)

The OGSA is an open source grid service standard jointly developed by academia and the IT industry under coordination of a working group in the *Global Grid Forum* (GGF). The standard was specifically developed for the emerging grid and cloud service communities. The OGSA is extended from web service concepts and technologies. The standard defines a common framework that allows businesses to build grid platforms across enterprises and business partners. The intent is to define the standards required for both open source and commercial software to support a global grid infrastructure.

### 7.1.3.1 OGSA Framework

The OGSA was built on two basic software technologies: the *Globus Toolkit* widely adopted as a grid technology solution for scientific and technical computing, and web services (WS 2.0) as a popular standards-based framework for business and network applications. The OGSA is intended to support the creation, termination, management, and invocation of stateful, transient grid services via standard interfaces and conventions [5]. The OGSA framework specifies the physical environment, security, infrastructure profile, resource provisioning, virtual domains, and execution environment for various grid services and API access tools.

Compared with the layered grid architecture in [Figure 7.1](#), the OGSA is service-oriented. A service is an entity that provides some capability to its client by exchanging messages. We feel that greater flexibility is needed in grid service discovery and management. The service-oriented architecture (SOA) presented in [Chapter 5](#) serves as the foundation of grid computing services. The individual and collective states of resources are specified in this service standard. The standard also specifies interactions

between these services within the particular SOA for grids. An important point is that the architecture is not layered, where the implementation of one service is built upon modules that are logically dependent. One may classify this framework as object-oriented. Many web service standards, semantics, and extensions are applied or modified in the OGSA.

### 7.1.3.2 OGSA Interfaces

The OGSA is centered on grid services. These services demand special well-defined application interfaces. These interfaces provide resource discovery, dynamic service creation, lifetime management, notification, and manageability. The conventions must address naming and upgradeability. [Table 7.3](#) summarizes the interfaces proposed by the OGSA working group. While the OGSA defines a variety of behaviors and associated interfaces, all but one of these interfaces (the grid service) is optional. Two key properties of a grid service are transience and statefulness. These properties have significant implications regarding how a grid service is named, discovered, and managed. Being transient means the service can be created and destroyed dynamically; statefulness refers to the fact that one can distinguish one service instance from another.

**Table 7.3**

#### OGSA Grid Service Interfaces Developed by the OGSA Working Group

Port Type	Operation	Brief Description
Grid service	Find service data	Query a grid service instance, including the handle, reference, primary key, home handle map, interface information, and service-specific information. Extensible support for various query languages.
	Termination time	Set (and get) termination time for grid service instance.
	Destroy	Terminate grid service instance.
Notification source	Subscribe to notification topic	Subscribe to notifications of service events. Allow delivery via third-party messaging services.
Notification sink	Deliver notification	Carry out asynchronous delivery of notification messages.
Registry	Register service	Conduct soft-state registration of Grid Service Handles (GSHs).
	Unregister service	Unregister a GSH.
Factory	Create service	Create a new grid service instance.
Handle map	Find by handle	Return the Grid Service Reference (GSR) associated with the GSH.

### 7.1.3.3 Grid Service Handle

A GSH is a globally unique name that distinguishes a specific grid service instance from all others. The status of a grid service instance could be that it exists now or that it will exist in the future. These instances carry no protocol or instance-specific addresses or supported protocol bindings. Instead, these information items are encapsulated along with all other instance-specific information. In order to interact with a specific service instance, a single abstraction is defined as a GSR.

Unlike a GSH, which is time-invariant, the GSR for an instance can change over the lifetime of the service. The OGSA employs a “handle-resolution” mechanism for mapping from a GSH to a GSR. The GSH must be globally defined for a particular instance. However, the GSH may not always refer to the same network address. A service instance may be implemented in its own way, as long as it obeys the associated semantics. For example, the port type on which the service instance was implemented decides which operation to perform in [Table 7.3](#).

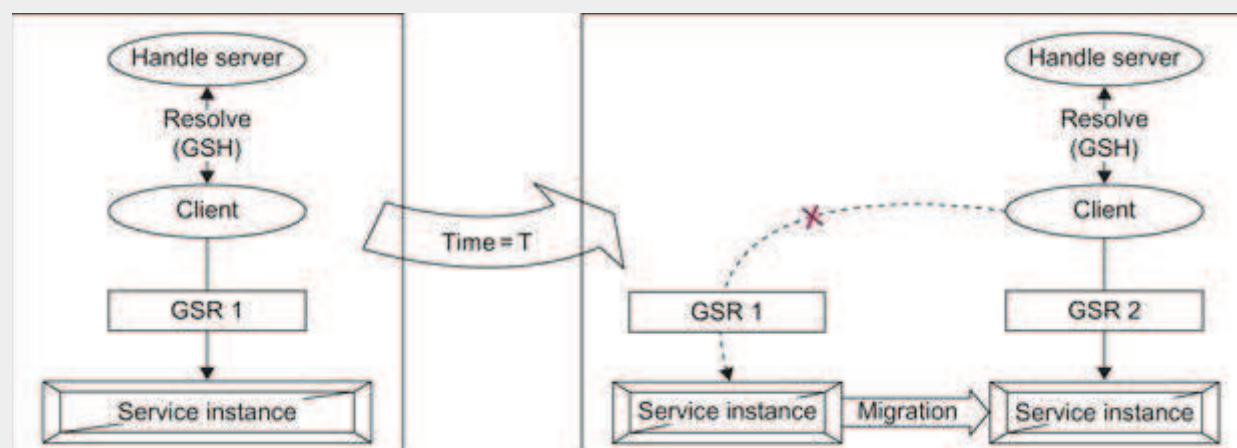
### 7.1.3.4 Grid Service Migration

This is a mechanism for creating new services and specifying assertions regarding the lifetime of a service. The OGSA model defines a standard interface, known as a *factor*, to implement this reference. Any service that is created must address the former services as the reference of later services. The factory interface is labeled as a *Create Service* operation in [Table 7.3](#). This creates a requested grid service with a specified interface and returns the GSH and initial GSR for the new service instance. It should also register the new service instance with a handle resolution service. Each dynamically created grid service instance is associated with a specified lifetime.

#### Example 7.3

#### Grid Service Migration Using GSH and GSR

[Figure 7.3](#) shows how a service instance may migrate from one location to another during execution. A GSH resolves to a different GSR for a migrated service instance before (on the left) and after (on the right) the migration at time T. The *handle resolver* simply returns different GSRs before and after the migration. The initial lifetime can be extended by a specified time period by explicitly requesting the client or another grid service acting on the client’s behalf.



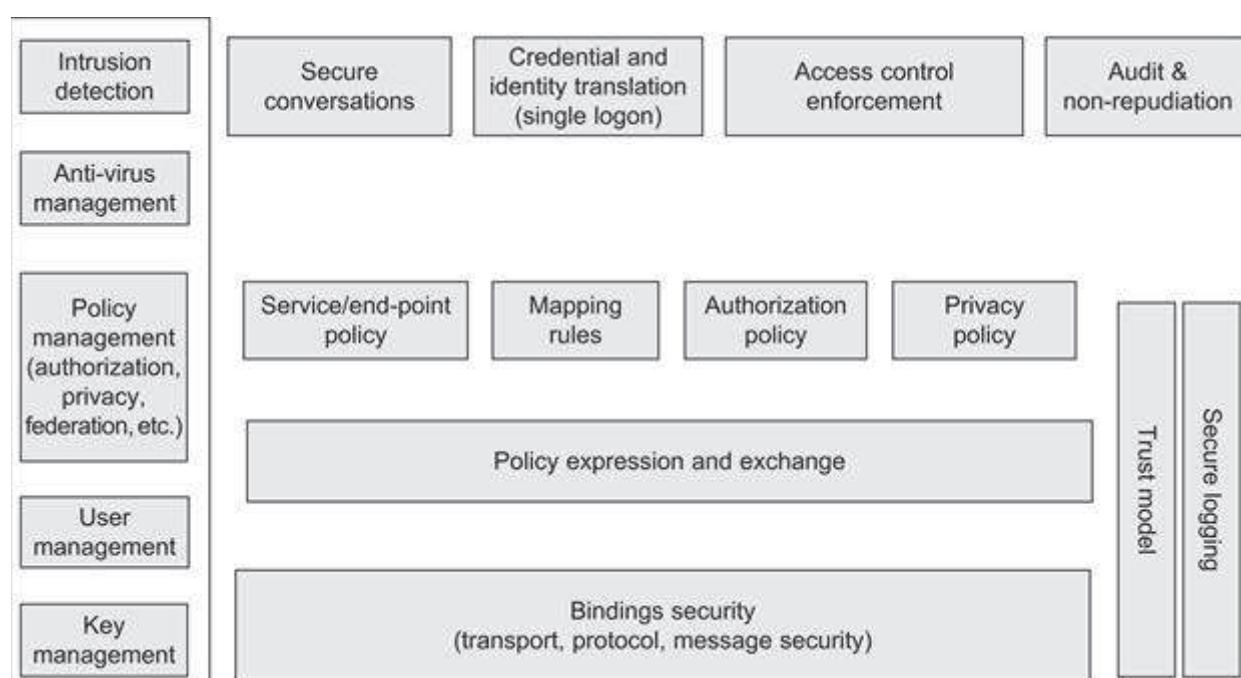
**FIGURE 7.3** A GSH resolving to a different GSR for a migrated service instance before (shown on

the left) and after (on the right) the migration at time T.

If the time period expires without having received a reaffirmed interest from a client, the service instance can be terminated on its own and release the associated resources accordingly. The lifetime management enables robust termination and failure detection. This is done by clearly defining the lifetime semantics of a service instance. Similarly, a hosting environment is guaranteed to consume bounded resources under some system failures. If the termination time of a service is reached, the hosting environment can reclaim all resources allocated.

### 7.1.3.5 OGSA Security Models

The OGSA supports security enforcement at various levels, as shown in Figure 7.4. The grid works in a heterogeneous distributed environment, which is essentially open to the general public. We must be able to detect intrusions or stop viruses from spreading by implementing secure conversations, single logon, access control, and auditing for nonrepudiation. At the security policy and user levels, we want to apply a service or endpoint policy, resource mapping rules, authorized access of critical resources, and privacy protection. At the *Public Key Infrastructure* (PKI) service level, the OGSA demands security binding with the security protocol stack and bridging of *certificate authorities* (CAs), use of multiple trusted intermediaries, and so on. Trust models and secure logging are often practiced in grid platforms.



**FIGURE 7.4** The OGSA security model implemented at various protection levels. Courtesy of I. Foster, et al., <http://www.ogf.org/documents/GFD.80.pdf>

## 7.1.4 Data-Intensive Grid Service Models

Applications in the grid are normally grouped into two categories: *computation-intensive* and *data-intensive*. For data-intensive applications, we may have to deal with massive amounts of data. For example, the data produced annually by a Large Hadron Collider may exceed several petabytes ( $10^{15}$  bytes). The grid system must be specially designed to discover, transfer, and manipulate these massive data sets. Transferring massive data sets is a time-consuming task. Efficient data management demands low-cost storage and high-speed data movement. Listed in the following paragraphs are several common methods for solving data movement problems.

### 7.1.4.1 Data Replication and Unified Namespace

This data access method is also known as caching, which is often applied to enhance data efficiency in a grid environment. By replicating the same data blocks and scattering them in multiple regions of a grid, users can access the same data with locality of references. Furthermore, the replicas of the same data set can be a backup for one another. Some key data will not be lost in case of failures. However, data replication may demand periodic consistency checks. The increase in storage requirements and network bandwidth may cause additional problems.

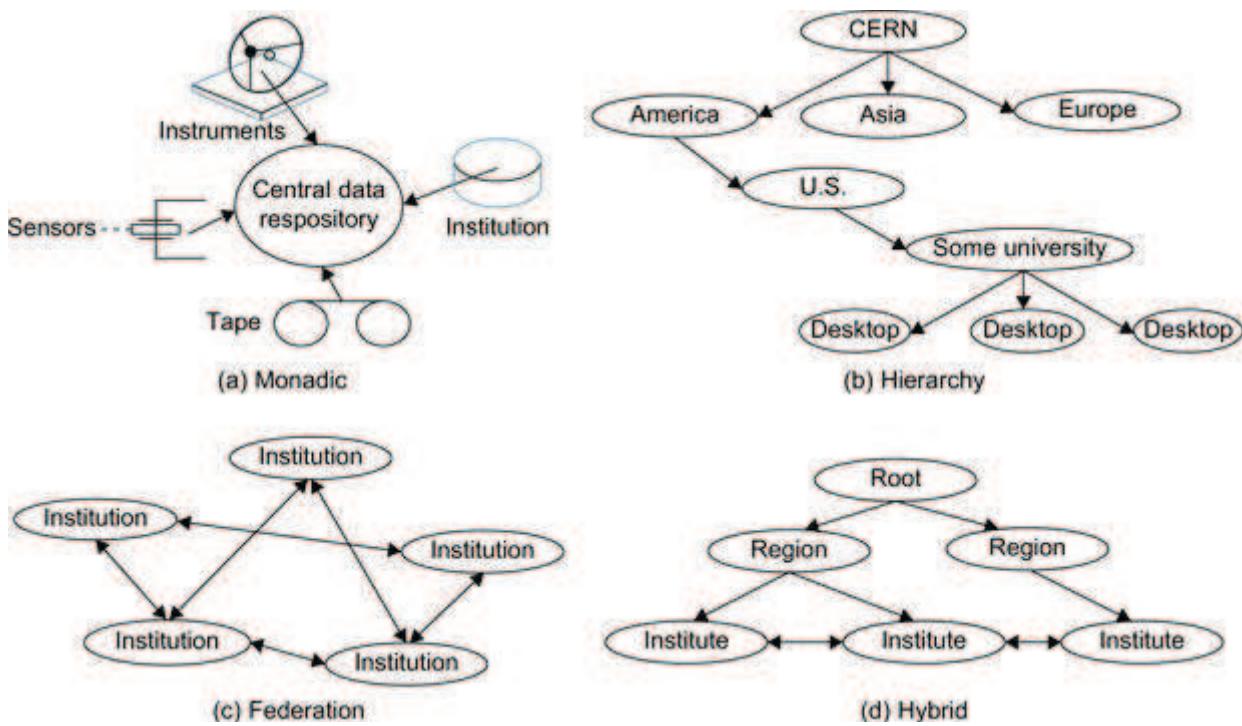
Replication strategies determine when and where to create a replica of the data. The factors to consider include data demand, network conditions, and transfer cost. The strategies of replication can be classified into method types: *dynamic* and *static*. For the static method, the locations and number of replicas are determined in advance and will not be modified. Although replication operations require little overhead, static strategies cannot adapt to changes in demand, bandwidth, and storage availability. Dynamic strategies can adjust locations and number of data replicas according to changes in conditions (e.g., user behavior).

However, frequent data-moving operations can result in much more overhead than in static strategies. The replication strategy must be optimized with respect to the status of data replicas. For static replication, optimization is required to determine the location and number of data replicas. For dynamic replication, optimization may be determined based on whether the data replica is being created, deleted, or moved. The most common replication strategies include preserving locality, minimizing update costs, and maximizing profits.

### 7.1.4.2 Grid Data Access Models

Multiple participants may want to share the same data collection. To retrieve any piece of data, we need a grid with a unique global namespace. Similarly, we desire to have unique file names. To achieve these, we have to resolve inconsistencies among multiple data objects bearing the same name. Access restrictions may be imposed to avoid confusion. Also, data needs to be protected to avoid leakage and damage. Users who want to access data have to be authenticated first and then authorized for access. In general, there are

four access models for organizing a data grid, as listed here and shown in Figure 7.5.



**FIGURE 7.5** Four architectural models for building a data grid.

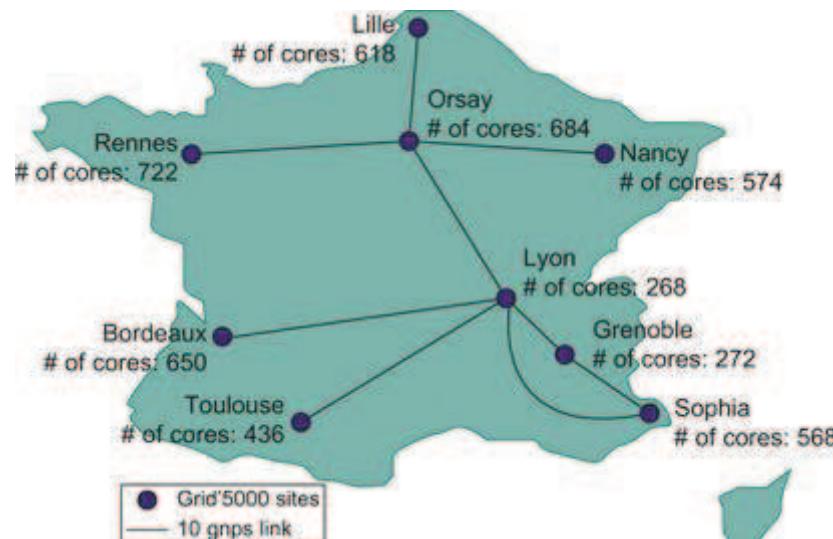
**Monadic model:** This is a centralized data repository model, shown in Figure 7.5(a). All the data is saved in a central data repository. When users want to access some data they have to submit requests directly to the central repository. No data is replicated for preserving data locality. This model is the simplest to implement for a small grid. For a large grid, this model is not efficient in terms of performance and reliability. Data replication is permitted in this model only when fault tolerance is demanded.

**Hierarchical model:** The hierarchical model, shown in Figure 7.5(b), is suitable for building a large data grid which has only one large data access directory. The data may be transferred from the source to a second-level center. Then some data in the regional center is transferred to the third-level center. After being forwarded several times, specific data objects are accessed directly by users. Generally speaking, a higher-level data center has a wider coverage area. It provides higher bandwidth for access than a lower-level data center. PKI security services are easier to implement in this hierarchical data access model. The *European Data Grid* (EDG) to be studied in Section 7.2.3 adopts this data access model.

**Federation model:** This data access model shown in Figure 7.5(c) is better suited for designing a data grid with multiple sources of data supplies. Sometimes this model is also known as a mesh model. The data sources are distributed to many different locations. Although the data is shared, the data items are still owned and controlled by their original owners. According to predefined access policies, only authenticated users are authorized to request data from any data source. This mesh model may cost the most when the number of grid institutions becomes very large.

**Hybrid model:** This data access model is shown in Figure 7.5(d). The model combines

the best features of the hierarchical and mesh models. Traditional data transfer technology, such as FTP, applies for networks with lower bandwidth. Network links in a data grid often have fairly high bandwidth, and other data transfer models are exploited by high-speed data transfer tools such as GridFTP developed with the Globus library. The cost of the hybrid model can be traded off between the two extreme models for hierarchical and mesh-connected grids.



**FIGURE 7.6** Overview of Grid'5000 located at nine resource sites in France. Courtesy of <https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home>

### 7.1.4.3 Parallel versus Striped Data Transfers

Compared with traditional FTP data transfer, *parallel data transfer* opens multiple data streams for passing subdivided segments of a file simultaneously. Although the speed of each stream is the same as in sequential streaming, the total time to move data in all streams can be significantly reduced compared to FTP transfer. In *striped data transfer*, a data object is partitioned into a number of sections, and each section is placed in an individual site in a data grid. When a user requests this piece of data, a data stream is created for each site, and all the sections of data objects are transferred simultaneously. Striped data transfer can utilize the bandwidths of multiple sites more efficiently to speed up data transfer.

## 7.2 Grid Projects and Grid Systems Built

Grid computing provides promising solutions to contemporary users who want to effectively share and collaborate with one another in distributed and self-governing environments. Apart from volunteer grids, most large-scale grids are national or international projects funded by public agencies. This section reviews the major grid systems developed in recent years. In particular, we describe three national grid projects that have been installed in the U.S., EU, and China.

## 7.2.1 National Grids and International Projects

Like supercomputers, national grids are mainly funded through government sources. These national grids are developed to promote research discovery, middleware products, and utility computing in grid-enabled applications.

### 7.2.1.1 National Grid Projects

Over the past decade, many data, information, or computational grids were built in various parts of the world. [Table 7.4](#) summarizes five representative grid computing systems built in the United States, European Union, United Kingdom, France, and China. We call these *national grids*, because they are essentially government-funded projects pushing for grand challenge applications that demand high-performance computing and high-bandwidth communication networks. Here treat the EU countries as a single entity. Most national grids are built by linking supercomputer centers and major computer ensembles together with Internet backbones and high-bandwidth WANs or LANs. These grids are briefly introduced in [Table 7.4](#). More details can be found in the cited subsequent sections.

**Table 7.4**

#### Some National Grid Projects in Five Countries

Project	Sponsor, Launch	Distributed Computing Capability and Applications
TeraGrid (USA)	NSF, 2002 ( <a href="#">Sec. 7.2.2</a> )	An open grid infrastructure in 11 resource sites, with 40 Gbps Internet2 backbone links with 2 Pflops computing power, 50 PB storage, and 100 specific databases.
DataGrid (EU)	EU and CERN, 2001 ( <a href="#">7.2.3</a> )	Claimed the largest grid on earth for data analysis in high-energy physics, environment, and bioinformatics, etc.
Grid'5000 (France)	French Gov., 2006, ( <a href="#">Sec. 7.2.1</a> )	An experimental grid comprising 5,000 processor cores in 9 French cities for HPC and research use ( <a href="#">Section 7.2.2</a> ).
ChinaGrid (China)	Education Ministry 2005, ( <a href="#">Sec. 7.2.4</a> )	An educational computing grid for research use, linking HPC systems in 100 Chinese universities.
NAS Grid (USA)	NASA Ames Lab, 2008	Running genetic algorithms with the Condor cycle scavenger on large number of Sun and SGI workstations.

### 7.2.1.2 International Grid Projects

Grid applications cannot be restricted to geographical boundaries. As summarized in [Table 7.5](#), several global-scale grid projects were launched or are still active in use today. These projects promote volunteer computing, utility computing, and specific software applications that utilizes grid infrastructure. International grids involve both government and industrial funding. The European Union has been a major player in grid computing. The most famous EU grid projects are the EGEE, DataGrid, and BEinGrid. In the industrial sector, we have seen grid providers including Sun Microsystems, IBM, HP, etc. International grids are built with fix-term projects. Some of them are no longer active to provide public services at the end of funding.

**Table 7.5****International Grid Projects (Past and Current)**

Project	Description and Operating Region	Status
EGEE	Enabling Grids for E-sciencE in European Union countries	2004–2010
D4Science	Grid-enabled technology for science in Europe, Asia and the Pacific	2008–2009
Nordic D-Grid	Nordic Data grid facility in Scandinavia and Finland	2006–2010
Future Grid	High-performance and grid research groups in U.S. universities	2010–present
WorldGrid	World community grid has 6.2 million machines running the BOINC in 2011	2004–present
SETI@Home	Volunteer grid with 2.5 machines running radioscope signals in 2001	2001–present
BOINC	Berkeley-led volunteer grid projects including the PrimeGrid, DNETC2Home, MilkyWay@Home, Collatz Conjective, etc.	2002–present
BEinGRID	Business Experiments in Grid funded by the European Commission	2006–2009

**Example 7.4****The EGEE Grid for Enabling Grids for E-science**

The EGEE grid infrastructure and projects were funded by the European Union in 3 stages from 2004 to 2006. EGEE stands for Enabling Grids for E-science. The project is no longer active. However, the distributed computing infrastructure built and nurtured by the projects DataGrid (2002-2004), EGEE-I, -II and -III (2004-2010) is now supported by the European Grid Infrastructure. This long-term organization coordinates National Grid Initiatives. Transitioning the care of the EGEE infrastructure to EGI is part of the process for ensuring a vibrant and sustained European research community.

The EGEE grid infrastructure consists of a set of middleware services deployed on a worldwide collection of computational and storage resources plus the services provided. For production services, a large multi-science grid infrastructure involves 250 resource centers world-wide, providing some 40,000 CPUs and several petabytes of storage. This infrastructure is used on a daily basis by several thousands of scientists over 200 virtual organizations on a daily basis. This is a stable, well-supported infrastructure, running the latest released versions of the gLite middleware. The EDEE also has a pre-production service, which provides access to grid services to test and evaluate features of the middleware.

In addition, the pre-production extends the middleware certification

activity, helping to evaluate deployment procedures, interoperability, and basic software functionality. Furthermore there is an *EGEE Network Operations Center* (ENOC), which caters for the network operational coordination between EGEE and other network providers such as GEANT2 and NRENs. In the long run, the primary goals of EGEE and EGI projects are to promote global collaboration for scientific research between academic and industrial applications. Reported EGEE application challenges include health and medicine, genomics and bioscience, particle physics and astronomy, environmental science, engineering design, chemistry and material science, and social sciences. Interested readers are referred to the web site: <http://www.eu-egee.org> for details.

### **Example 7.5**

#### **Grid'5000 Built over Nine French Cities**

Grid'5000 is an experimental grid testbed platform for providing large-scale computing and research instruments. As a national infrastructure, Grid'5000 connects 4,792 processor cores distributed over nine sites in France. Figure 7.6 shows the processor core distribution. Grid'5000 is a confined system. PKI and X.509 certificates are used to enforce security in Grid'5000. The system uses dedicated private networks to construct a trustable backbone network.

The Grid'5000 sites are connected via high-bandwidth links. Each user has a universal account. Each site manages its local user accounts, and each site runs a *Lightweight Directory Access Protocol* (LDAP) server, which maintains the same directory tree for all users. The user accounts at the local site are treated with higher priority to read and write shared files. The user accounts at remote sites are only allowed to read files locally. The account information of remote sites in the tree is synchronized periodically. From the user's point of view, a globally enabled account is installed in the entire grid system.

### **7.2.2 NSF TeraGrid in the United States**

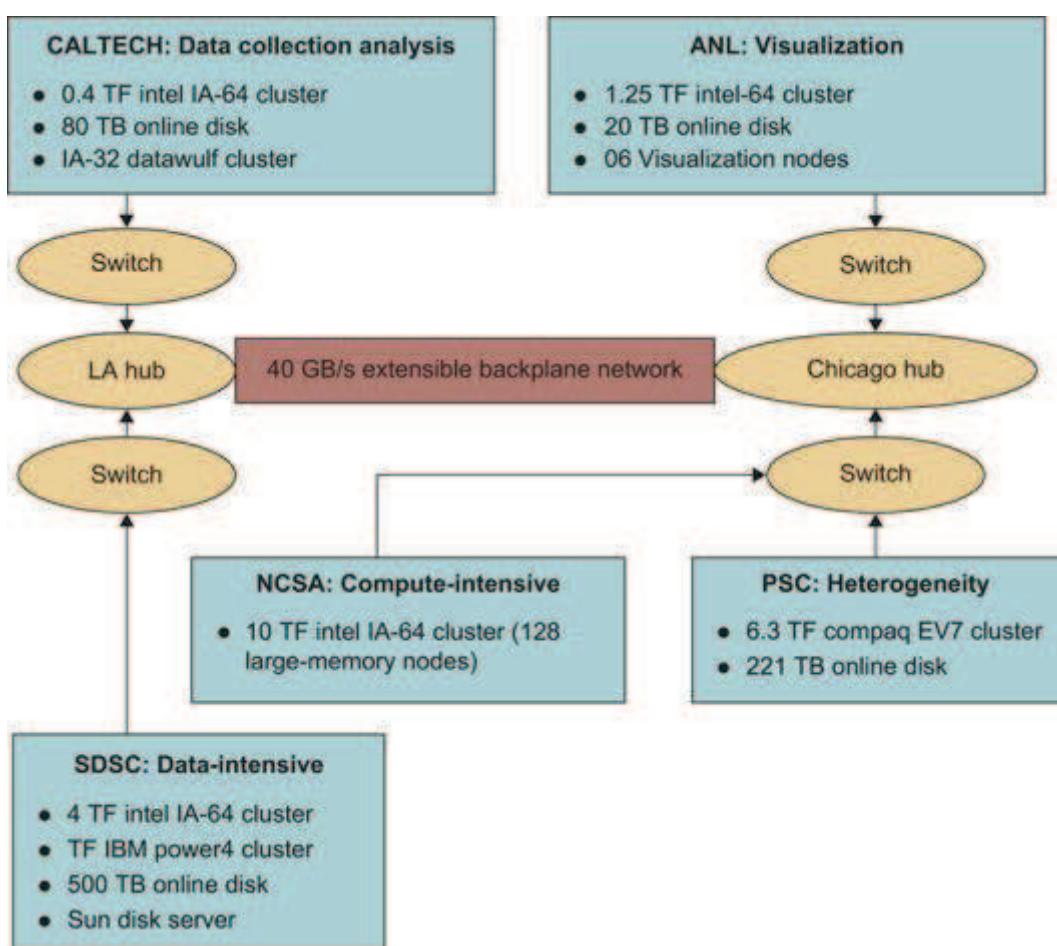
TeraGrid is an open scientific discovery infrastructure combining leadership-class resources at 11 partner sites to create an integrated, persistent computational resource in the USA. Using high-performance network connections, TeraGrid integrates high-performance computers, data resources and tools, and high-end experimental facilities around the country. At the time of this writing, TeraGrid resources include more than 2 Pflops of computing capability and more than 50 petabytes of online and archival data

storage, with rapid access and retrieval over high-performance networks. Researchers can also access more than 100 discipline-specific databases.

TeraGrid is coordinated through the Grid Infrastructure Group (GIG) at the University of Chicago, working in partnership with the 11 resource provider sites: Indiana University, the Louisiana Optical Network Initiative, the National Center for Supercomputing Applications (NCSA) at UIUC, the National Institute for Computational Sciences, Oak Ridge National Laboratory (ORNL), the Pittsburgh Supercomputing Center (PSC), Purdue University, the San Diego Supercomputer Center (SDSC), the Texas Advanced Computing Center (TACC), the University of Chicago/Argonne National Laboratory (ANL), and the National Center for Atmospheric Research (NCAR).

When the project started in 2001, TeraGrid had only four resource sites at ANL, SDSC, UIC, and Caltech. In 2003, NSF added four sites to TeraGrid as well as established a third network hub, in Atlanta. These new sites were at ORNL, Purdue University, Indiana University, and TACC. TeraGrid construction was also made possible through key corporate partnerships with Sun Microsystems, IBM, Intel Corporation, Qwest Communications, Juniper, Myricom, Hewlett-Packard Company, and Oracle Corporation. At the time of this writing, the TeraGrid facility is entering full production use. The next-generation petascale resources, called XD, may be added into one or two national grid facilities in the United States from 2011 to 2016.

TeraGrid is a *high-performance computing and communication (HPCC)* system using the Globus software/middleware tools to develop grids linking multiteraflop computers together. At the time of this writing, the Internet backbone networks used to build TeraGrid include Gbps links (four 10 Gigabit Ethernet bundled together). [Figure 7.7](#) shows five TeraGrid resource sites and their backbone networks. Major resource types at each grid site are identified. A 40 GB/second extensible backplane network connects two regional gateway hubs in Chicago and Los Angeles.



**FIGURE 7.7** Five major TeraGrid resource sites, connected by a 40 GB/s backplane network between the LA and Chicago hubs. Courtesy of <https://www.teragrid.org/web>

TeraGrid resources are integrated through SOA. Each resource provides a service in terms of specific interfaces and operational support. Computational resources run a set of software packages called *Coordinated TeraGrid Software and Services* (CTSS). CTSS provides a familiar user environment on all TeraGrid systems, allowing scientists to port code from one system to another. CTSS also provides integrative functions such as single sign-on, remote job submission, workflow support, and data movement tools. CTSS includes the Globus Toolkit, Condor, distributed accounting management software, verification and validation software, and a set of compilers, programming tools, and environment variables.

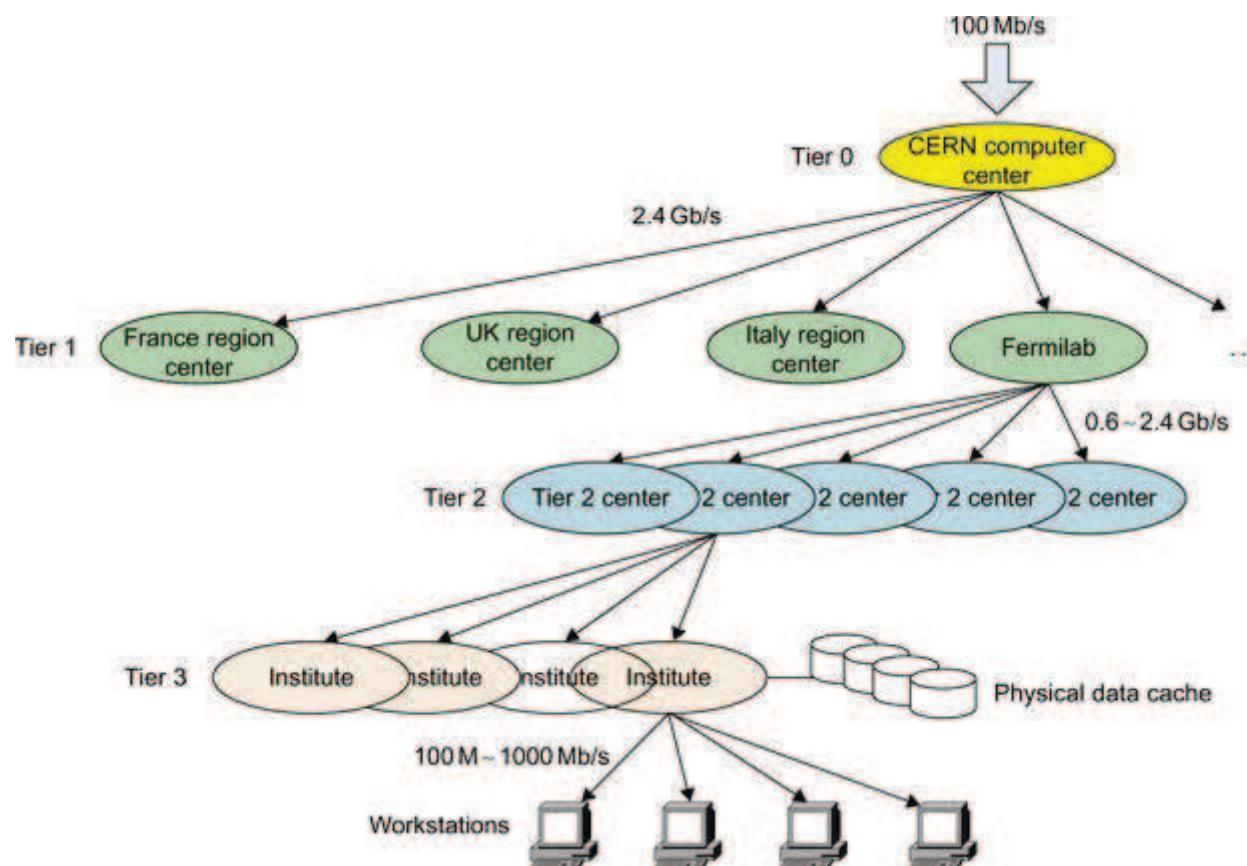
TeraGrid resources are interconnected by a dedicated optical network, with each resource provider site connecting at either 10 Gigabits per second or 30 Gigabits per second. TeraGrid users access the facility through national research networks such as the Internet2 Abilene backbone and National LambdaRail. TeraGrid users primarily come from U.S. universities. Academic researchers in the United States can obtain exploratory, or CPU hours. More extensive allocations involve proposal submissions and awards after a peer-review process. All allocation proposals are handled through the TeraGrid web site, <https://www.teragrid.org/web>.

## 7.2.3 DataGrid in the European Union

The European DataGrid Project (EDG) was funded by the European Union. This grid aims to build the next-generation, high-throughput, production-quality grid infrastructure to support I/O-intensive experiments in high-energy physics, earth observation, and bioinformatics. Specific grid middleware was developed to coherently manage and share petabyte-range information volumes. The goal is to secure access to massive amounts of data in a universal namespace. The system moves and replicates data at high speeds from one geographical site to another and keeps consistency checking and synchronization of multiple copies in a global scale.

### 7.2.3.1 The EDG Architecture

Figure 7.8 shows the overall architecture of the EDG based on information released by CERN. The DataGrid project has a multitier hierarchical organization. Data on the order of several petabytes will be distributed in a hierarchical fashion to multiple sites worldwide. For example, Tier-0 is the CERN computer center (European Organization for Nuclear Research) based in Geneva. This tier stores almost all the relevant data. Several Tier-1 regional centers are located in Italy, France, the United Kingdom, and the United States. The regional centers store smaller amounts of data. The EDG uses an extensible schema-based resource model with a hierarchical namespace to handle the creation of and access to all distributed and replicated data items.



**FIGURE 7.8** Hierarchical data distribution of the European Data Grid (EDG). <http://eu-datagrid.web.cern.ch/eu-datagrid> © CERN

Each Tier-1 center is further linked to a large number of Tier-2 data centers, which

further link to many Tier-3 institutes in a hierarchical manner. End users access the data grid at the workstations. The EDG does not guarantee QoS, and the resource information is based on an LDAP network directory. Resource dissemination is batched and periodically pushed to other parts of the grid. Resource discovery in the EDG is decentralized and query-based. The scheduler uses a hierarchical organization with an extensible scheduling policy. A special workload distribution facility balances the analysis of jobs submitted from hundreds of physicists to different entry points in the grid. Application monitoring and collection of user access patterns can be used to optimize global access and data distribution.

### **7.2.3.2 Data Distribution in the EDG**

The core middleware used is the Globus Toolkit with hooks for data grids. The physical grid fabric delivers all the necessary tools to manage a computing center, providing grid services on clusters of thousands of server nodes. The EDG uses European and national research network infrastructures to provide *virtual private networks* (VPNs) between the servers and the storage of resources. The EDG wide-area test bed interfaces with existing mass storage management systems based on the core Globus middleware. Grid monitoring enables end users and administrators to check status and error information in the EDG environment.

The data accessor and data locator are used to map location-independent identifiers to location-dependent identifiers. The data accessor is an interface encapsulating the details of the local file system and mass storage systems such as Castor and HPSS. Several implementations of this generic interface exist. The data locator makes use of the generic metadata manager, which is responsible for efficient publishing and management of a distributed and hierarchical set of objects. Query optimization ensures that for a given query an optimal migration and replication plan is produced. Such plans provide appropriate security mechanisms to cover connected resources, independent of organizational boundaries.

## **7.2.4 The ChinaGrid Design Experiences**

ChinaGrid aims to build the largest education and research platform of over 100 universities across China. The system is used for research, science, and education projects by aggregating existing dispersed resources throughout the *China Education and Research Network* (CERNET). In 2002, ChinaGrid was launched by the Ministry of Education of China among 100 key universities in China. The goal is to provide a nationwide grid computing platform. This grid service attempts to promote collaborative research, and scientific and education projects. ChinaGrid integrates heterogeneous massive resources distributed based on grid technology.

### **7.2.4.1 Connectivity of ChinaGrid**

As a long-term project, the first phase of ChinaGrid, from 2003 to 2006, focused on various platforms and applications (ranging from life science to computational physics)

on a computation grid (e-Science) and covered many top universities in China, among them Tsinghua University, Huazhong University of Science and Technology, Peking University, Beihang University, South China University of Technology, Shanghai Jiao Tong University, Southeast University, Xi'an Jiaotong University, National University of Defense Technology, etc. Details on the connectivity can be found in (<http://grid.hust.edu.cn/platform/project/chinagrid/index.html>).

ChinaGrid is built and deployed based on CERNET, which is the second largest nationwide network in China and covers 800 addition universities, colleges, and institutes in China. A high-bandwidth (2.5 Gbps for the backbone) and rich scientific and education resource provides the fabric environment for constructing ChinaGrid. At the core of ChinaGrid is its grid middleware, called *ChinaGrid Supporting Platform* (CGSP). CGSP integrates all kinds of resources in education and research environments, makes the heterogeneous and dynamic nature of these resources transparent to users, and provides a high-performance, highly reliable, secure, convenient, and transparent grid service for scientific computing and engineering research.

#### **7.2.4.2 Applications of ChinaGrid**

CGSP provides the ChinaGrid service portal, a set of core grid services including execution management, data services, information services, and monitoring services, and various utilities for deploying various grid applications. Based on CGSP, five application-specific grids are deployed: a bioinformatics grid, computational fluid dynamics grid, massive information processing grid, distance education grid for offering higher education courses online, and image processing grid. At the time of this writing, ChinaGrid's total computing power had reached 170 teraflops and its total storage capacity had reached 170 terabytes. ChinaGrid projects support a variety of applications, including weather forecasting, petroleum reservoir simulation, high-energy physics, numerical wind tunnel simulation, ship structure analysis, and traffic control and guidance.

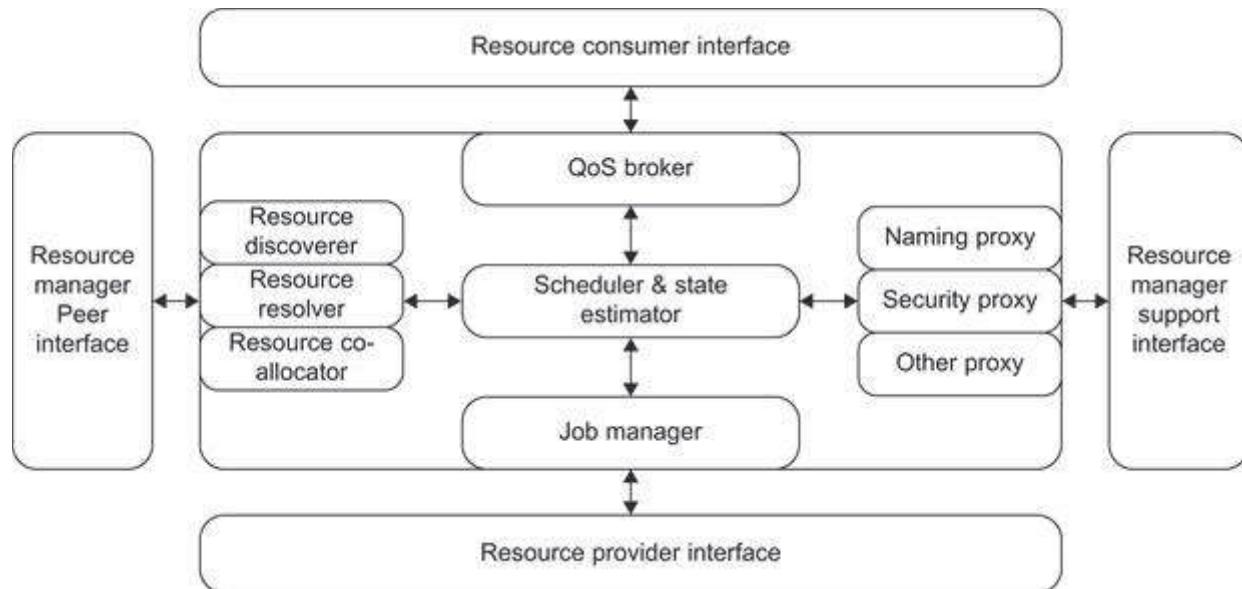
### **7.3 Grid Resource Management and Brokering**

In this section, we will discuss the grid resource management middleware, including scheduling, monitoring, and accounting. We will present two grid resource management technologies, the Globus Toolkit and the ChinaGrid Support Platform. Grid resource brokering and scheduling are two kernel functions of grid resource management. We will also cover grid operational flow and introduce the Gridbus resource broker as a brokering example.

#### **7.3.1 Resource Management and Job Scheduling**

In a grid system, resources are usually autonomous. Each organization may have its own resource management policies. It is more reasonable and adaptable to set an *individual resource management system* (RMS) for an organization—the RMSes in the upper level can

be considered the resource consumers, and the RMSes in the lower level can be considered the resource providers. To support such a multi-RMS structure, an abstract model for RMS is introduced. [Figure 7.9](#) shows the structure of this model. There are four published interfaces in the model. The resource consumer interface is for access from upper-level RMS or user applications. The resource discoverer actively searches qualified resources by this interface. The resource disseminator broadcasts information about local resources to other RMSes. The resource trader exchanges resources among RMSes for market-based grid systems. The resource resolver routes jobs to remote RMSes. The resource co-allocator simultaneously allocates multiple resources to a job.



**FIGURE 7.9** Major components of a grid resource management system (RMS) supported by Globus. Courtesy of I. Foster [17]

### 7.3.1.1 Hierarchy of RMSes

These RMSes are interconnected with one another. The RMSes are set in a hierarchical structure. The resource provider interface is for the lower-level RMS or actual resources. Resources and jobs in the lower-level RMS are managed and controlled through this interface. If it is supported, a resource reservation agent is also implemented here. A resource manager peer interface is used to interoperate with other RMSes in the same level. Job requests are accepted from this interface, and are forwarded to the local scheduler.

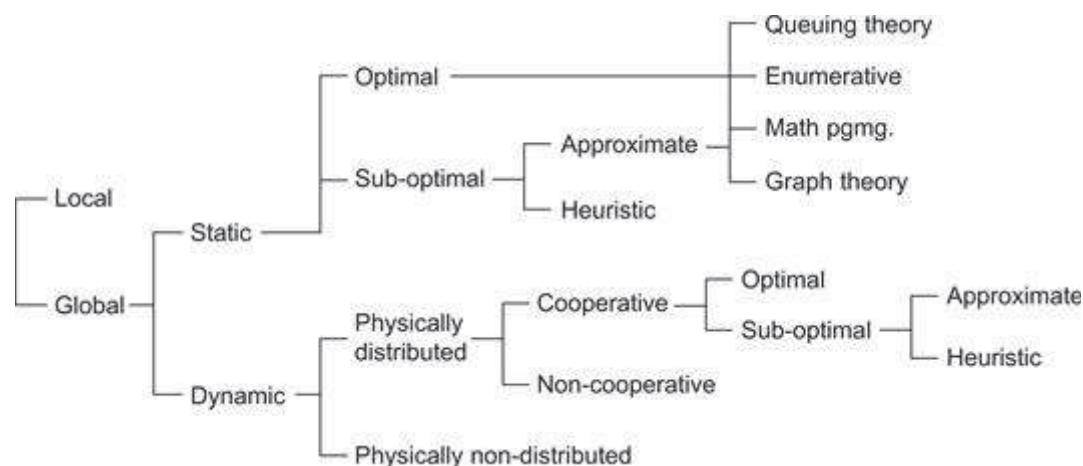
### 7.3.1.2 Grid Job Scheduling Methods

The grid is a new kind of distributed computing technology. Scheduling technology in the grid is not a completely new research topic. Traditional distributed systems (e.g., clusters) have similar issues in grid scheduling. These issues may have been explored for decades. Much research has been conducted on scheduling technology in traditional distributed systems. Two schemas are often used to classify them: *hierarchical*

*classification*, which classifies scheduling methods by a multilevel tree (Figure 7.11) and *flat classification* based on a single attribute. These include methods based on adaptive versus nonadaptive, load balancing, bidding, or probabilistic and one-time assignment versus dynamic reassignment.

The resources in a grid are often heterogeneous. The hardware capacity (e.g., the number of processors and the amount of memory) and the software configuration may be different for each site. It is reasonable for traditional scheduling to determine resource allocation without considering the differences among the sites. But this is unacceptable for a grid. The scheduler has to estimate the performance of a job for each candidate resource. We need a grid performance model to assess the cost in terms of time and efficiency in relation to grid applications.

A dynamic performance model is more accurate in estimating real grid applications. Some skills may be exploited to improve the precision of simulation modeling. The inner structure of the application or source code of the program needs to be analyzed. This modeling approach may cut the makespan of an individual job into several sections, including the *queuing time*, *data transfer time*, and *computation time*. Some sample data sets and benchmark programs are needed to collect the information for training the performance model.



**FIGURE 7.10** Hierarchical classification of grid job scheduling methods for grids and computer clusters.

The resources in a grid are dynamically allocated for different applications. For traditional distributed systems, the system administrator has full control of the resources. (P2P is not considered here, because job scheduling is not a key issue of it.) There are only two players (the system and the user). The scheduling technologies can completely focus on performance. The optimization objective of the scheduling can be system-centric or application-centric. However, the owner of the resources has the right to determine the number and quality of resources to be shared in the grid. The grid has three players: the grid system, resource provider, and resource customer.

Without any incentives, the resource providers will be reluctant to share more and better resources. So the optimization objectives of scheduling in the grid have to take the benefit of resource providers into account. The most common incentive is to allocate

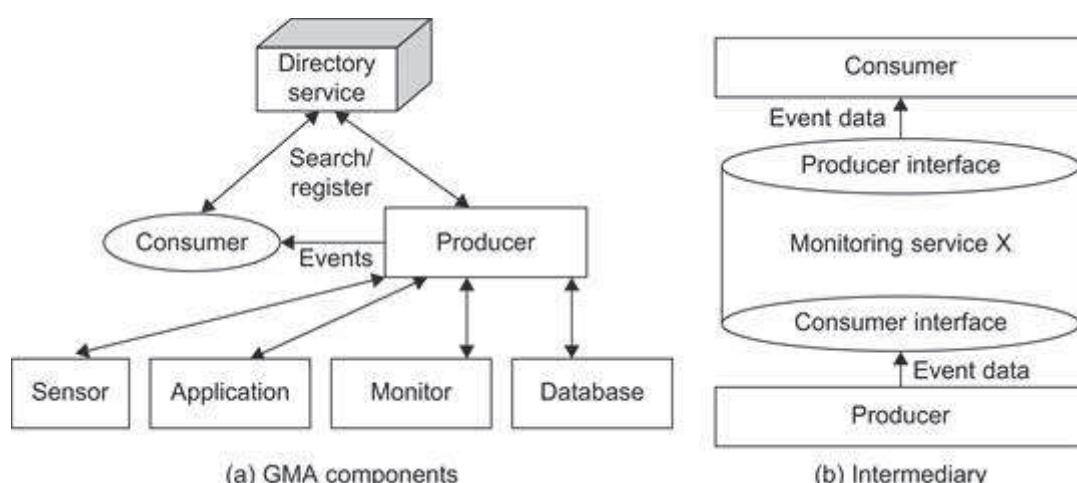
resources for jobs based on economics principles. In an economic-based grid computing environment, users express their requirements for QoS, and their preferred cost for different levels QoS. The grid service providers need to work out a cost model for various services.

## 7.3.2 Grid Resource Monitoring with CGSP

Generally, when we talk about monitoring in a grid environment to solve a particular problem, many elements are involved, including target grid resources as well as relevant grid middleware and application components on top of them.

### 7.3.2.1 Producer/Consumer Model for Grid Monitoring

The producer/consumer model is shown in Figure 7.11. This model provides a basic framework for grid resource monitoring to support functionalities required by users, and to improve interoperability among different grid resources. Grid monitoring data includes short lifetimes, frequent changes, randomness, and so on. The model should satisfy the requirements of low latency, high transfer rates, low overhead, security, and scalability. In particular, the discovery and transmission of grid monitoring data should be handled separately. Related metadata describing the profiles of monitoring data needs to be stored in a public location.



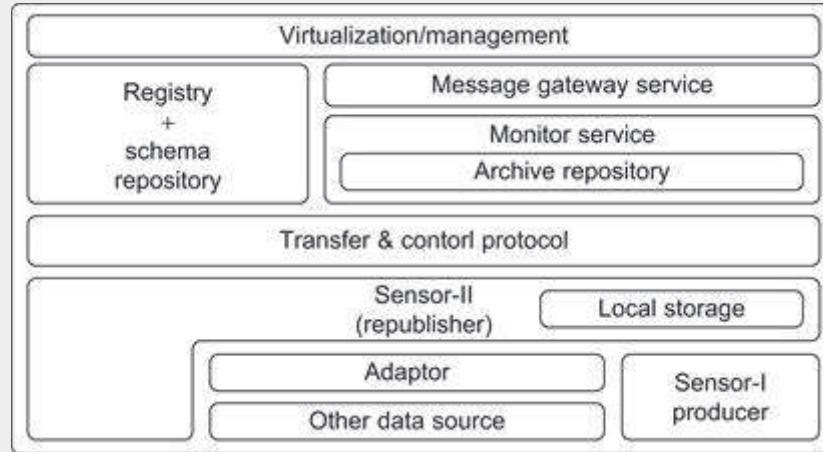
**FIGURE 7.11** Producer/consumer grid monitoring model implemented in the grid monitoring architecture (GMA).

### Example 7.6

#### ChinaGrid SuperVision (CGSV) for Resource Monitoring

In ChinaGrid, the monitoring system is essential to keep such a complex distributed system efficient. The CGSP components, including the Job

Manager, Storage Manager, and Information Center, provide system status information for different purposes. The *ChinaGrid SuperVision* (CGSV) system (Figure 7.12) was specifically designed for monitoring ChinaGrid performance. In addition to common inventory tracking tasks, ChinaGrid also requires CGSV to have scalable support to different types of data requests, and an efficient approach to data processing and transmission.



**FIGURE 7.12** Architecture of CGSV module deployment. Courtesy of Y. Wu, et al. [56]

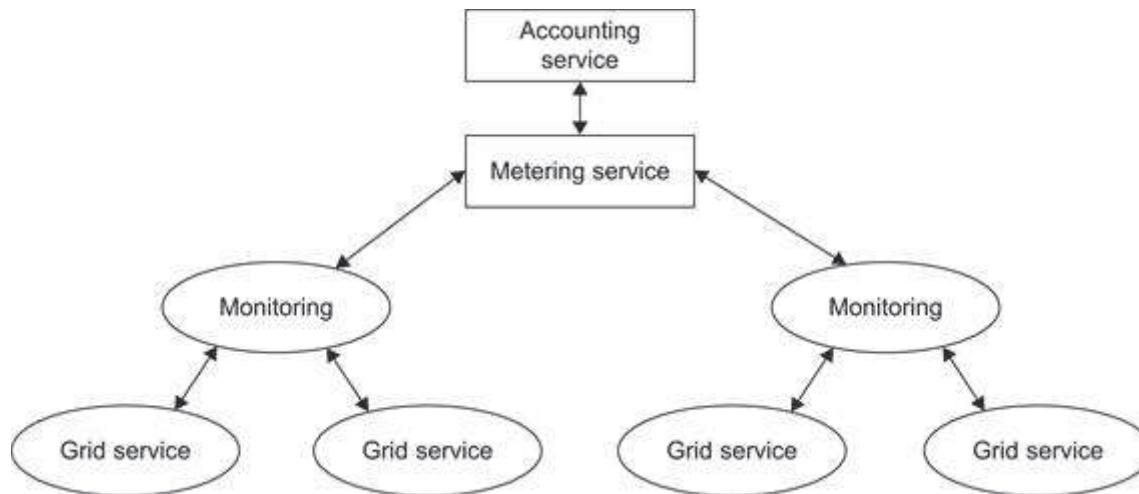
To cope with these requirements, CGSV is designed to be an adaptable, stream-integrated grid monitoring system. A transfer and control protocol was designed to efficiently transfer various types of measurement data and perform modifications over producer behavior in a unified way. Republishers are stream-oriented, in that they are designed to support predicate-based processing over measurement data streams and SQL-like stream query interfaces.

The CGSV is designed based on the GMA proposed by the Open Grid Forum (OGF). The CGSV is built with a collection layer, service layer, and presentation layer, as shown in Figure 7.12. In the collection layer, sensors are deployed to measure data collection. Sensor-I is responsible for collecting hardware resource information. Other sources of information including network condition and dynamic service information or data generated by other monitoring tools are wrapped by an adapter to unify the data format.

Above the data interface, we have the service layer. In each domain of ChinaGrid, this layer is presented as a logical domain monitoring center, where Domain Registry Service and Monitor Services are deployed. A Message Gateway Service is provided to alleviate the communication cost of monitoring services since too many notifications and subscriptions will decrease service performance tremendously. The top layer is the presentation layer, where data analysis and visualization are performed.

## 7.3.3 Service Accounting and Economy Model

Economics-based resource trading is an effective way to allocate grid resources. First, resource providers register their shared resources together with resource information in a grid market. The resource information may include hardware capacity, software configuration, and price policy. Then resource customers submit their jobs with special requirements to a resource broker. These requirements may include hardware capacity, software configuration, QoS, and budget. Next, the broker searches and allocates qualified resources in the grid market for each job. When the jobs are successfully completed, the customers are charged for the resources allocated. [Figure 7.13](#) shows the framework of an accounting system for distributed grid services. Two accounting services are introduced in the following paragraphs.



**FIGURE 7.13** A generic framework of a grid accounting system for billing services.

### 7.3.3.1 Monitoring and Metering Services

This component directly collects raw usage information (e.g., CPU time) from grid services or resources. Generally speaking, each grid service container or computing node should have a monitoring component deployed. The collected raw data is forwarded to the upper-level component, the metering component. The raw data can be gathered by querying the grid service, analyzing the log data of the application, or obtaining it directly from the underlying system. The metering service obtains the raw usage information from the monitoring component, and applies charge policies based on service usage. The charge policies are the mappings between raw usage events and monetary charge events. The charge policies specify the formulas used to obtain the cost from the application, user, and result.

To maintain good grid economy, the accounting system demands a grid economy model. The usage data collected by the accounting system will be considered as the foundation of charges to customers. If the usage data is faked or modified by malicious services, incorrect actions will be operated on customers' and providers' accounts. As a

result, the order of the grid economy will be seriously disturbed. Some accounting systems have taken measures to ensure security. GridBank [8], the accounting system of Gridbus, uses SOAP over Globus Toolkit sockets. Access from the remote service is authenticated and authorized with this grid security infrastructure. Grids normally consist of multiple real organizations. The accounting has to be performed across multiple domains, so the accounting system must provide the access interface of standard grid services. Globus Toolkit is the most popular and de facto standard for grid services.

### 7.3.3.2 Accounting Component

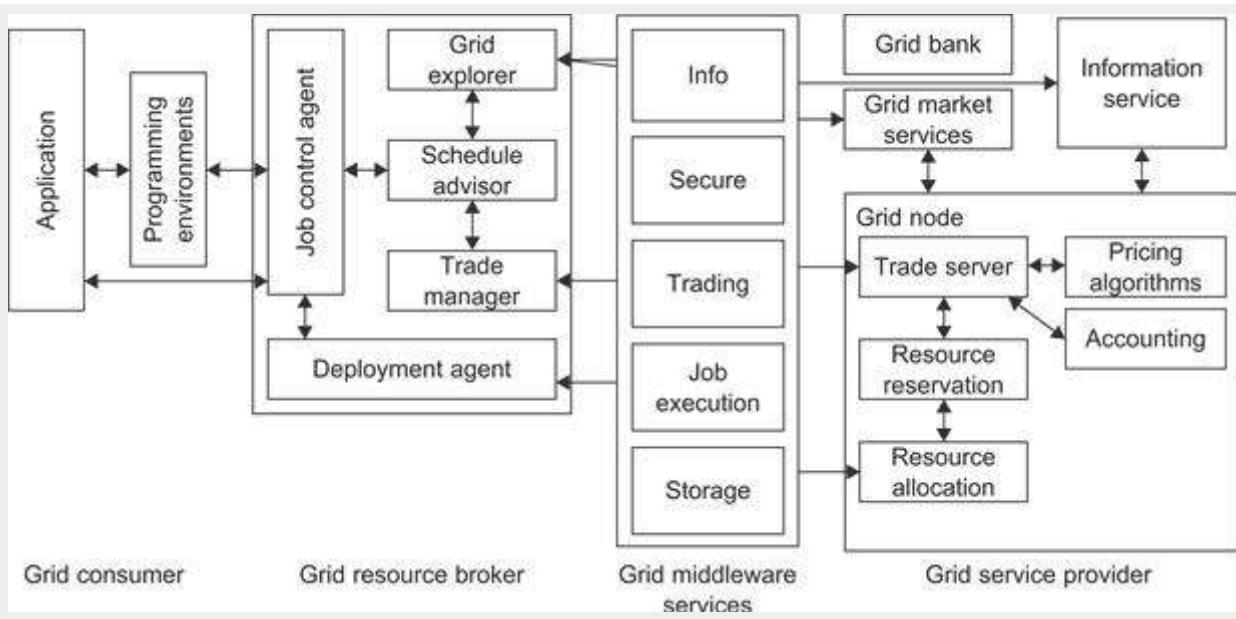
The primary function of an accounting module is to maintain the economic relationship between resource customers and providers. By receiving and converting charge events from the metering module, the accounting module records billing events. An accounting event is the action which will be operated on the account of each grid user. The account of the grid user holds the information on the balance amount. Some other features are also demanded for accounting in a grid, such as the security of the accounting system.

Over the past several years, realizing the full promise of the grid has required solutions to fundamental issues of authentication, authorization, resource discovery, resource access, and most notably, incompatibility of resources and policies for managing them. A wide variety of projects and products offer services intended to address these issues. Several grid middleware libraries have been developed and deployed, including the popular Globus Toolkit, glite, and CGSP. In this section, we will focus on Globus Toolkit and CGSP, two of the most important grid middleware libraries that underlie various systems and that provide the basis for most grid technology deployments.

#### Example 7.7

#### An Economy Model for Grid Computing

Figure 7.14 shows a generic economy model for assessing grid computing services in terms of cost-effectiveness. Besides the users (consumer) and providers, the computational economy needs some other components. The Trade Manager identifies resource access costs; the Grid Market Directory allows resource owners to publish their services; the Grid Trade Server negotiates with resource users and maximizes benefits; Pricing Policies define the price of the resource; and the Grid Bank records resource usage and bills the user. Co-allocation of resources is a special demand of grid scheduling. Some kinds of applications (e.g., workflow, multisite applications) in grids usually demand multiple resources from multiple sites simultaneously.

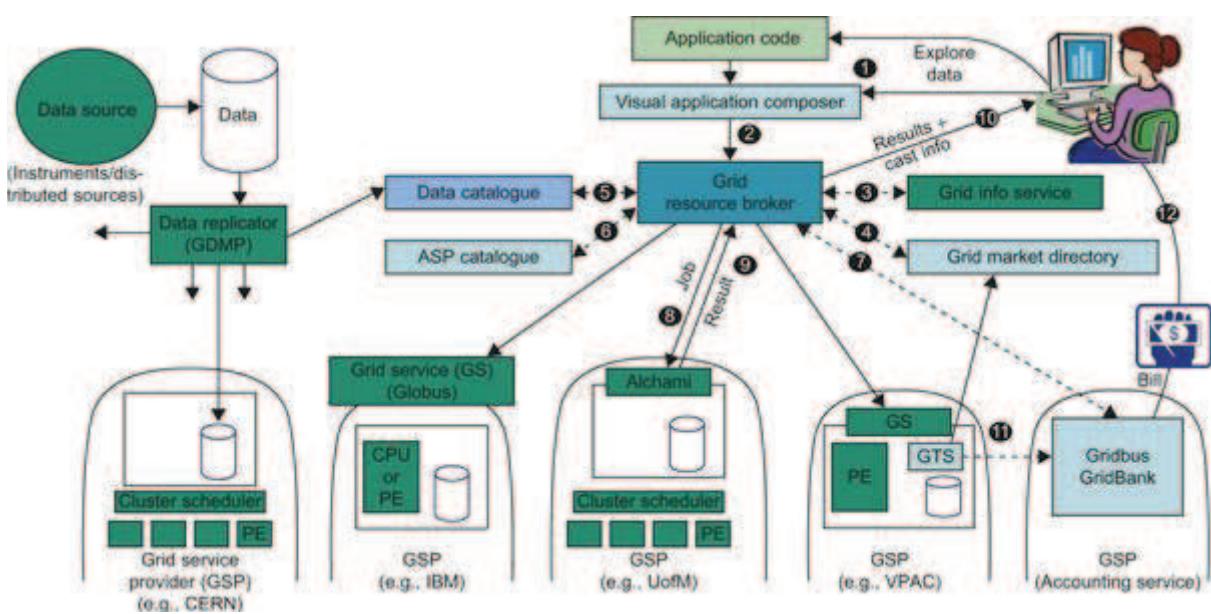


**FIGURE 7.14** An economy model for assessing grid computing services, developed at the University of Melbourne. Courtesy of Buyya, et al. [8]

For example, interactive data analysis may require simultaneous access to several resources, such as a storage system holding a copy of the data, a supercomputer for analysis, network elements for data transfer, and a display device for interaction. Two-phase transaction is often used to solve the problem of co-allocation. To request multiple resources, the first step is to acquire qualified resources. The second step is to bind the resource for specific applications. If the commission of any resource fails, the commissions of all resources will be cancelled, and the system will start to find a new solution for the request.

### 7.3.4 Resource Brokering with Gridbus

Figure 7.15 shows the grid environment from an operations perspective with the components organized according to their deployment and functionality. To make resources constituents of the grid, they need to be accessible from different management domains. This can be achieved by installing core grid middleware such as Globus in UNIX/Linux environments. Multinode clusters need to be presented as a single resource to the grid, and this can be achieved by deploying job management systems such as the Sun Grid Engine on them.



**FIGURE 7.15** A grid resource broker managing job scheduling and execution. Courtesy of Buyya and Bubendorfer (editors) [9]

In a grid environment where data needs to be federated for sharing among various interested parties, data grid technologies such as SRB, Globus RLS, and EU DataGrid need to be deployed. The user-level middleware needs to be deployed on resources responsible for providing resource brokering and application execution management services. Users may even access these services via web portals. Several grid resource brokers have been developed. Some of the more prominent include Nimrod-G, Condor-G, GridWay, and Gridbus Resource Broker. The following 11 steps are followed to aggregate the Grid resources:

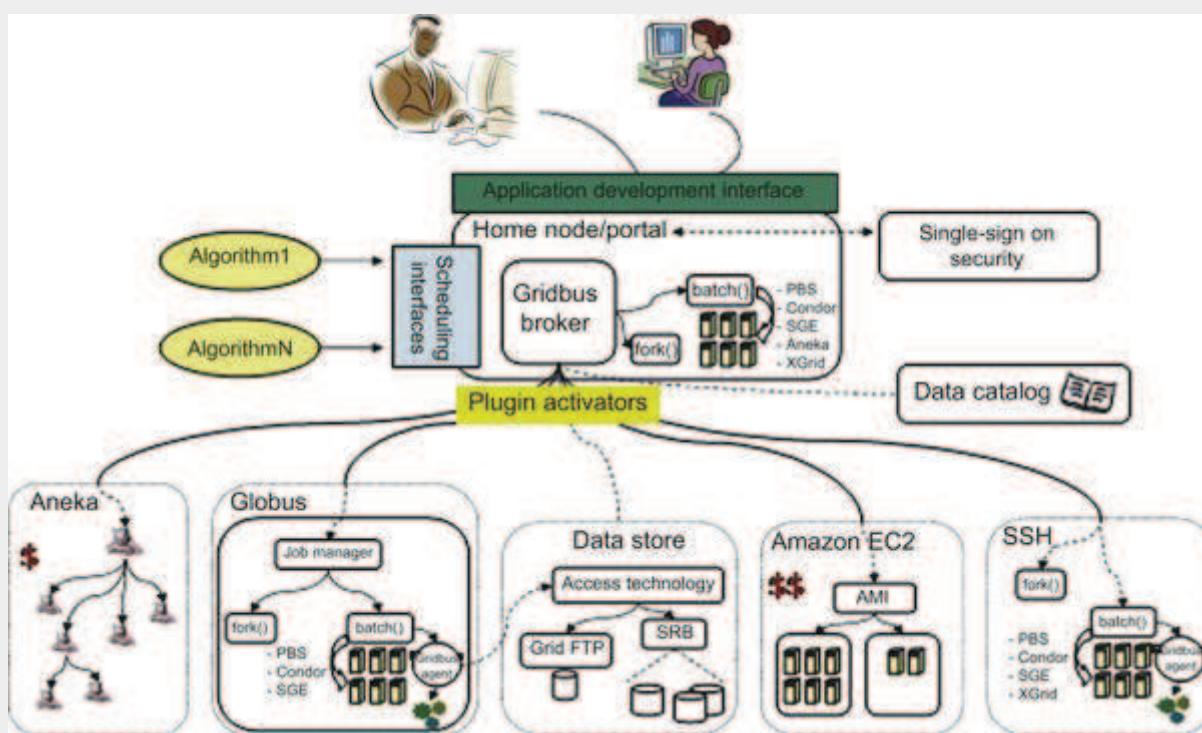
1. The user composes his application as a distributed application (e.g., parameter sweep) using visual application development tools.
2. The user specifies his analysis and QoS requirements and submits them to the grid resource broker.
3. The grid resource broker performs resource discovery using the grid information service.
4. The broker identifies resource service prices by querying the grid market directory.
5. The broker identifies the list of data sources or replicas and selects the optimal ones.
6. The broker identifies computational resources that provide the required services.
7. The broker ensures that the user has necessary credit or an authorized share to utilize the resources.
8. The broker scheduler analyzes the resources to meet the user's QoS requirements.
9. The broker agent on a resource executes the job and returns the results.
10. The broker collates the results and passes them to the user.
11. The meter charges the user by passing the resource usage information to the accountant.

## Example 7.8

### The Gridbus Architecture at the University of Melbourne

The accountant reports resource share allocation or credit utilization to the user. The Gridbus Resource Broker [9,48] is a market-oriented meta-scheduler for computational and data grids, with support for a wide range of grid middleware and services. It takes care of many functions that grid applications require, including discovering the right resources for a particular user application, scheduling jobs to meet deadlines, and handling faults that may occur during execution.

In particular, the broker provides capabilities such as resource selection, job scheduling, job management, and data access to any application that requires distributed grid resources for execution. The broker handles communication with the resources running different grid middleware, job failures, varying resource availability, and different user objectives such as meeting a deadline for execution or limiting execution within a certain budget. The design of the Gridbus broker follows a layered architecture consisting of interface, core, and execution layers that together provide the capabilities shown in Figure 7.16 for the market-oriented broker.



**FIGURE 7.16** Architecture of the Gridbus resource broker and its interaction with many public or private grids/clouds. Courtesy of Buyya and Bubendorfer (editors) [9]

The interface layer consists of APIs and parsers for the input files through which external programs and users communicate with the broker. Resource

discovery and negotiation, scheduling, and job monitoring are carried out in the core layer. Job execution is carried out through the execution layers in which middleware-specific adapters communicate with the target resources. In the Gridbus broker, middleware-specific adapters are implemented as plug-in actuators that support deployment of jobs on resources accessible via grid middleware such as Globus and enterprise cloud application platforms such as Aneka.

The broker also works with public clouds such as Amazon EC2 and supports remote job execution via SSH on clusters managed by queuing systems such as PBS, Condor, and SGE. The broker enables different types of scheduling depending on the objectives of the user and type of resources. At present, the broker can accommodate compute, storage, network, and information resources with prices based on time (e.g., 1 Grid Dollar/s) or capacity (e.g., 1 Grid Dollar for 1 MB). It can also accommodate user objectives such as the fastest computation within the budget (time optimization), or the cheapest computation within the deadline (cost optimization) for both compute and data-intensive applications. The compute-intensive algorithms are based on those developed previously in Nimrod/G [10].

## 7.4 Software and Middleware for Grid Computing

In this section, we will study the middleware libraries developed for grid computing over the years. First, we give an overview of open source middleware packages. Then we introduce the Globus Toolkit based on version GT4. The concept of Globus containers and the grid resource allocation manager are studied in more detail. Finally, we study the CGSP library specifically developed out of the ChinaGrid Project.

### 7.4.1 Open Source Grid Middleware Packages

As reviewed in Berman, Fox, and Hey [5], many software, middleware, and programming environments have been developed for grid computing over past 15 years. Below we assess their relative strength and limitations based on recently reported applications. We first introduce some grid standards and popular APIs. Then we present the desired software support and middleware developed for grid computing. Table 7.6 summarizes four grid middleware packages.

**Table 7.6**

**Grid Software Support and Middleware Packages**

Package	Brief Description, References and Coverage in This Book
---------	---

BOINC	Berkeley Open Infrastructure for Network Computing.
UNICORE	Middleware developed by the German grid computing community.
Globus (GT4)	A middleware library jointly developed by Argonne National Lab., Univ. of Chicago, and USC Information Science Institute, funded by DARPA, NSF, and NIH. ( <a href="#">Sections 7.4.2, 7.4.3, and 7.5.5</a> ).
CGSP in ChinaGrid	The CGSP (ChinaGrid Support Platform) is a middleware library developed by 20 top universities in China as part of the ChinaGrid Project ( <a href="#">Section 7.4.4</a> ).
Condor-G	Originally developed at the Univ. of Wisconsin for general distributed computing, and later extended to Condor-G for grid job management. ( <a href="#">Example 7.9</a> ).
Sun Grid Engine (SGE)	Developed by Sun Microsystems for business grid applications. Applied to private grids and local clusters within enterprises or campuses. ( <a href="#">Example 7.10</a> ).

### 7.4.1.1 Grid Standards and APIs

Grid standards have been developed over the years. The Open Grid Forum (formally Global Grid Forum) and Object Management Group are two well-formed organizations behind those standards. We have already introduced the OGSA (*Open Grid Services Architecture*) in [Chapter 5](#) and [Section 7.1.3](#). In [Chapter 5](#), we have also reported some grid standards including the GLUE for resource representation, SAGA (*Simple API for Grid Applications*), GSI (*Grid Security Infrastructure*), OGSI (*Open Grid Service Infrastructure*), and WSRE (*Web Service Resource Framework*).

GSI will be studied in [Section 7.5.5](#). The grid standards have guided the development of several middleware libraries and API tools for grid computing. They are applied in both research grids and production grids today. Research grids tested include the EGEE, France Grilles, D-Grid (German), CNGrid (China), TeraGrid (USA), etc. Production grids built with the standards include the EGEE, INFN grid (Italian), NorduGrid, Sun Grid, Techila, and Xgrid [52]. We review next the software environments and middleware implementations based on these standards.

### 7.4.1.2 Software Support and Middleware

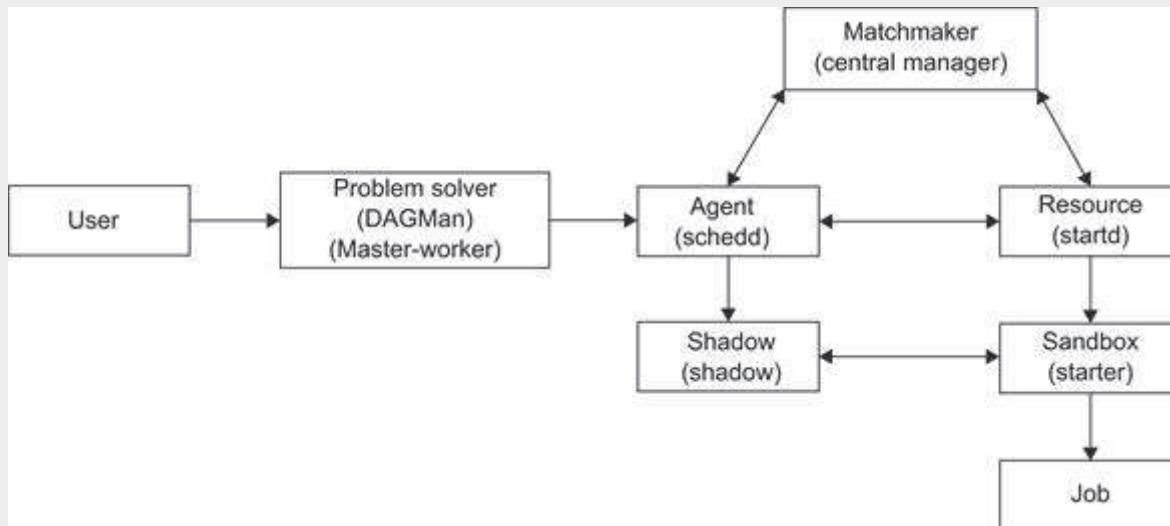
Grid middleware is specifically designed a layer between hardware and the software. The middleware products enable the sharing of heterogeneous resources and managing virtual organizations created around the grid. Middleware glues the allocated resources with specific user applications. Popular grid middleware tools include the Globus Toolkits (USA), gLight, UNICORE (German), BOINC (Berkeley), CGSP (China), Condor-G, and Sun Grid Engine, etc. [52]. [Table 7.6](#) summarizes the grid software support and middleware packages developed for grid systems since 1995. In subsequent sections, we will describe the features in Condor-G, SGE, GT4, and CGSP.

[Example 7.9](#) introduces the matchmaking capability of Condor and Condor-G developed at University of Wisconsin. [Example 7.10](#) discusses the features of Sun's Grid Engine developed at Sun Microsystems for business grid computing.

## Example 7.9

### Features in Condor Kernel and Condor-G for Grid Computing

Condor is a software tool for high-throughput distributed batch computing. It was designed to explore the idle cycles of a network of distributed computers. The major components of Condor are the user agent, resources, and matchmaker, as shown in Figure 7.17. The ClassAds (classified advertisements) language was exploited in Condor to express user requests against available resources in a distributed system. Agents and resources advertise their status and requirements in ClassAds to a central matchmaker. The matchmaker scans the ClassAds and creates pairs of (resources, agents) that match each other's requirements.



**FIGURE 7.17** Major functional components in a Condor system (Condor-specific names are in parentheses).

Subsequently, the matched agent negotiates with the available resource to execute the job. Two problem solvers are provided in Condor: the master-worker and the DAG manager. For specific jobs, Condor records checkpoints and subsequently resumes program execution from the checkpoints. When running a job on remote machines without a shared file system, Condor accesses the local execution environment via remote system calls. Condor-G was designed to match the GRAM protocol (to be covered in Section 7.4.4) for coupling jobs with resources in the Globus Project. Condor-G adds durability and two-phase commitment to prevent the loss and repetition of jobs in GRAM. However, Condor-G does not support all the features in GRAM.

## **Example 7.10**

### **The Sun Grid Engine (SGE) Middleware Package**

SGE was developed by Sun Microsystems in response to increasing demands of business grid applications. SGE features are applied to private grids and local clusters within an enterprise or campus for intranet-based cluster or grid applications. The system supports batch processing with dynamic allocation of grid resources. Fault tolerance and failover capability are implemented. Users can specify resources independent of submission locations. Job status and resource utilization can be monitored periodically.

Sun offers free and enterprise editions of the SGE package. The system offers centralized management of all resources allocated to individual jobs. This was meant to enhance efficiency and performance. Suspend and resume tools are provided to allow users to halt a job and restart it later without losing the work already completed. The SGE workflow follows the following sequence of events:

- Accepts the jobs from users
- Places jobs in a computer area until execution
- Sends the jobs from the holding area to a host where they can be executed
- Manages the jobs during execution
- Logs a record of their execution when they are finished

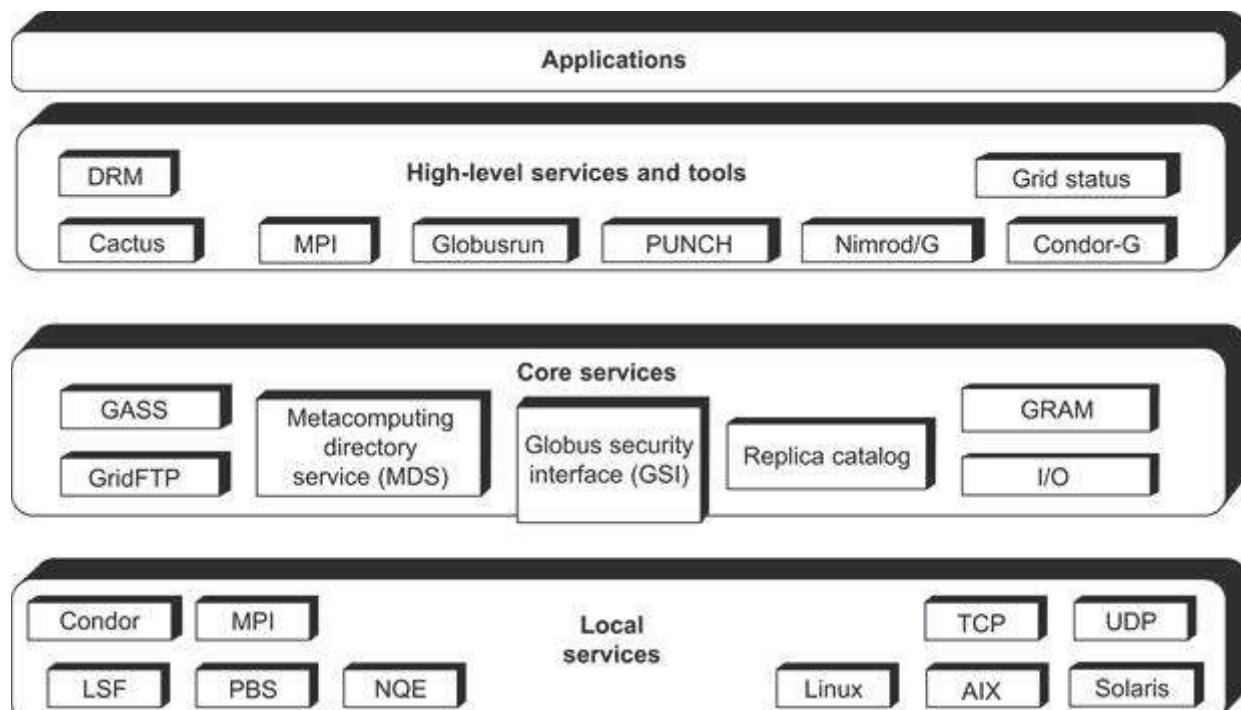
The SGE system uses reserved ports, Kerberos, DCE, SSL, and authentication in classified hosts to enforce security at different trust levels and resource access restrictions.

## **7.4.2 The Globus Toolkit Architecture (GT4)**

The Globus Toolkit, started in 1995 with funding from DARPA, is an open middleware library for the grid computing communities. These open source software libraries support many operational grids and their applications on an international basis. The toolkit addresses common problems and issues related to grid resource discovery, management, communication, security, fault detection, and portability. The software itself provides a variety of components and capabilities. The library includes a rich set of service implementations.

The implemented software supports grid infrastructure management, provides tools for building new web services in Java, C, and Python, builds a powerful standard-based security infrastructure and client APIs (in different languages), and offers comprehensive command-line programs for accessing various grid services. The Globus Toolkit was initially motivated by a desire to remove obstacles that prevent seamless

collaboration, and thus sharing of resources and services, in scientific and engineering applications. The shared resources can be computers, storage, data, services, networks, science instruments (e.g., sensors), and so on. The Globus library version GT4, is conceptually shown in [Figure 7.18](#).



**FIGURE 7.18** Globus Toolkit GT4 supports distributed and cluster computing services. Courtesy of I. Foster [17]

### 7.4.2.1 The GT4 Library

GT4 offers the middle-level core services in grid applications. The high-level services and tools, such as MPI, Condor-G, and Nimrod/G, are developed by third parties for general-purpose distributed computing applications. The local services, such as LSF, TCP, Linux, and Condor, are at the bottom level and are fundamental tools supplied by other developers. [Table 7.7](#) summarizes GT4's core grid services by module name. Essentially, these functional modules help users to discover available resources, move data between sites, manage user credentials, and so on. As a de facto standard in grid middleware, GT4 is based on industry-standard web service technologies.

**Table 7.7**

#### Functional Modules in Globus GT4 Library

Service Functionality	Module Name	Functional Description
Global Resource Allocation Manager	GRAM	Grid Resource Access and Management (HTTP-based)
Communication	Nexus	Unicast and multicast communication
Grid Security Infrastructure	GSI	Authentication and related security services
Monitoring and Discovery Service	MDS	Distributed access to structure and state information

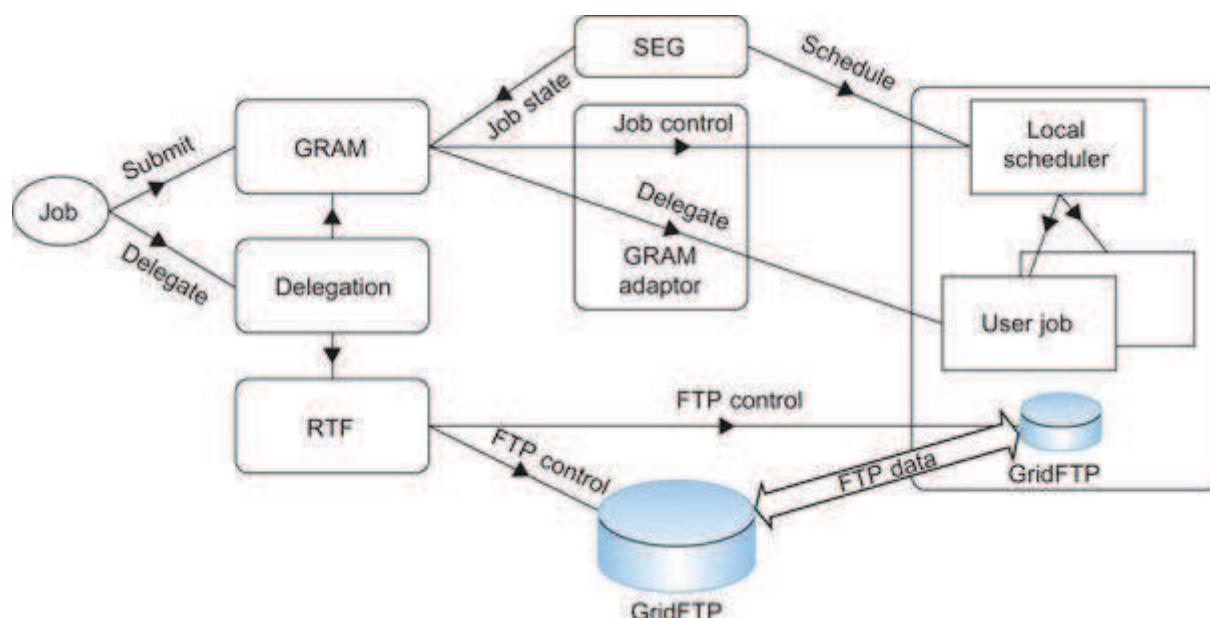
Health and Status	HBM	Heartbeat monitoring of system components
Global Access of Secondary Storage	GASS	Grid access of data in remote secondary storage
Grid File Transfer	GridFTP	Inter-node fast file transfer

Courtesy of Ian Foster [17].

In Section 7.4.3, we will examine the HTTP-based GRAM and MDS modules. Nexus is used for collective communications and HBM for heartbeat monitoring of resource nodes. GridFTP is for speeding up internode file transfers. The module GASS is used for *global access of secondary storage*. GSI (grid security infrastructure) will be covered in Section 7.5. More details of the functional modules of Globus GT4 and their applications are available at [www.globus.org/toolkit/](http://www.globus.org/toolkit/).

### 7.4.2.2 Globus Job Workflow

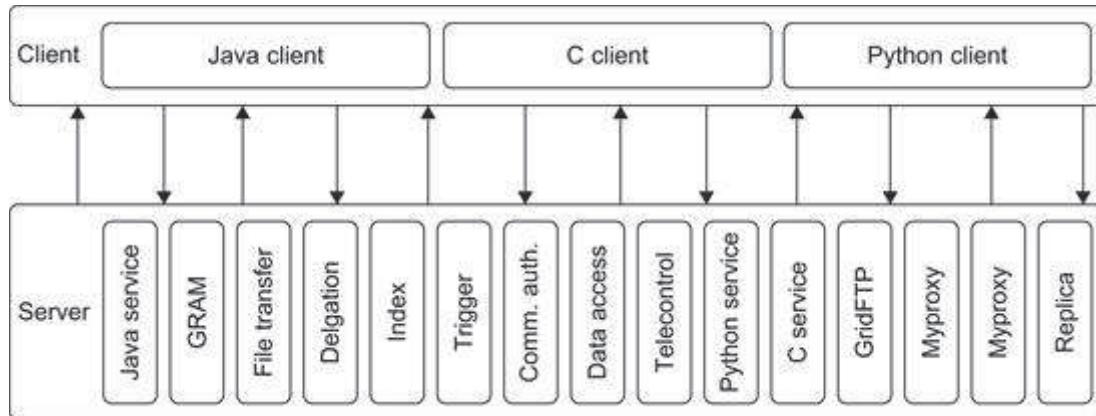
Figure 7.19 shows the typical job workflow when using the Globus tools. A typical job execution sequence proceeds as follows: The user delegates his credentials to a *delegation service*. The user submits a job request to GRAM with the delegation identifier as a parameter. GRAM parses the request, retrieves the user proxy certificate from the delegation service, and then acts on behalf of the user. GRAM sends a transfer request to the RFT (Reliable File Transfer), which applies GridFTP to bring in the necessary files. GRAM invokes a *local scheduler* via a GRAM adapter and the SEG (Scheduler Event Generator) initiates a set of user jobs. The local scheduler reports the job state to the SEG. Once the job is complete, GRAM uses RFT and GridFTP to stage out the resultant files. The grid monitors the progress of these operations and sends the user a notification when they succeed, fail, or are delayed.



**FIGURE 7.19** Globus job workflow among interactive functional modules. Courtesy of Foster and Kesselman [15]

### 7.4.2.3 Client-Globus Interactions

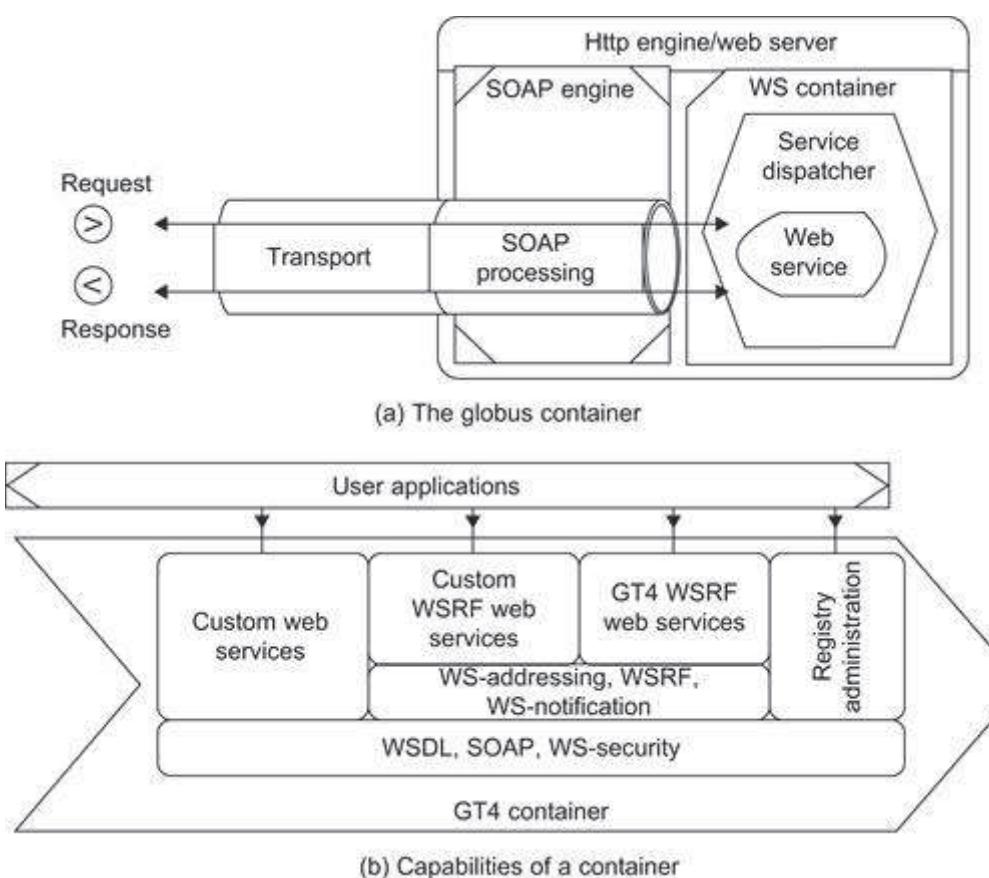
GT4 service programs are designed to support user applications as illustrated in [Figure 7.20](#). There are strong interactions between provider programs and user code. GT4 makes heavy use of industry-standard web service protocols and mechanisms in service description, discovery, access, authentication, authorization, and the like. GT4 makes extensive use of Java, C, and Python to write user code. Web service mechanisms define specific interfaces for grid computing. Web services provide flexible, extensible, and widely adopted XML-based interfaces.



**FIGURE 7.20** Client and GT4 server interactions; vertical boxes correspond to service programs and horizontal boxes represent the user codes. Courtesy of Foster and Kesselman [15]

GT4 components do not, in general, address end-user needs directly. Instead, GT4 provides a set of infrastructure services for accessing, monitoring, managing, and controlling access to infrastructure elements. The server code in the vertical boxes in [Figure 7.22](#) corresponds to 15 grid services that are in heavy use in the GT4 library. These demand computational, communication, data, and storage resources. We must enable a range of end-user tools that provide the higher-level capabilities needed in specific user applications. Wherever possible, GT4 implements standards to facilitate construction of operable and reusable user code. Developers can use these services and libraries to build simple and complex systems quickly.

A high-security subsystem addresses message protection, authentication, delegation, and authorization. Comprising both a set of service implementations (server programs at the bottom of [Figure 7.21](#)) and associated client libraries at the top, GT4 provides both web services and non-WS applications. The horizontal boxes in the client domain denote custom applications and/or third-party tools that access GT4 services. The toolkit programs provide a set of useful infrastructure services.



**FIGURE 7.21** Globus container serving as a runtime environment for implementing web services in a grid platform. Courtesy of Foster and Kesselman [16]

Three containers are used to host user-developed services written in Java, Python, and C, respectively. These containers provide implementations of security, management, discovery, state management, and other mechanisms frequently required when building services. They extend open source service hosting environments with support for a range of useful web service specifications, including WSRF, WS-Notification, and WS-Security.

A set of client libraries allow client programs in Java, C, and Python to invoke operations on both GT4 and user-developed services. In many cases, multiple interfaces provide different levels of control: For example, in the case of GridFTP, there is not only a simple command-line client (`globus-url-copy`) but also control and data channel libraries for use in programs—and the XIO library allowing for the integration of alternative transports. The use of uniform abstractions and mechanisms means clients can interact with different services in similar ways, which facilitates construction of complex, interoperable systems and encourages code reuse.

### 7.4.3 Containers and Resources/Data Management

In this section, we introduce the container concept introduced with the GRAM module in the GT4 Toolkit. Then we explain how to use GRAM and MDS in resource and data management in any open grid. GRAM supports dynamic job execution with coordinated file staging. MDS is used for monitoring and discovery of available grid resources in a grid execution environment.

### 7.4.3.1 Resource Management Using GRAM

The GRAM module supports web services for initiating, monitoring, and managing execution of computational jobs on remote computers in an open grid. GRAM is built atop various local resource allocation services. A standardized GRAM interface enables access to a variety of local resource management tools. Some of them are commercially available from cluster and grid computing vendors, such as the Load Sharing Facility introduced in [Chapter 2](#), the Network Queuing Environment, IBM's LoadLeveler, and Condor. This interface allows a client to express the resource type and quantity, data to be staged to and from the execution site, executable code and its arguments, security credentials to be used, and job persistence requirements. Other operations enable clients to monitor the status of both allocated resources and running tasks and notify users about their status, as well as guide job execution on the grid.

The heart of GRAM contains a set of web services designed to run the *Globus Web Services Resource Framework* (WSRF) core hosting environment. Each submitted job is exposed with a distinct resource qualifying the requested service. The service provides an interface to monitor the status of the job or to terminate the job. Each compute element is accessed through a local scheduler. The service provides an interface to create the managed job resources to perform a job in that local scheduler, as illustrated in the job workflow diagram.

### 7.4.3.2 Globus Container: A Runtime Environment

The Globus Container (also called a *web service core* or WS container) provides a basic runtime environment for hosting the web services needed to execute grid jobs. [Figure 7.21\(a\)](#) shows the container concept. The container is built with heavy use of SOAP engines. The main SOAP functions performed include the transport of incoming job requests and corresponding responses. It is also common to partition the hosting environment logic for transporting the SOAP message via an HTTP engine or web server.

[Figure 7.21\(b\)](#) summarizes the capabilities of the containers. All WS containers implement the SOAP commands over the HTTP engine as a message transport protocol. Both transport-level and message-level security is enforced in all communications. The WS-addressing, WSRF, and WS-Notification functions are implemented. The container supports logging using the Jakarta logging API. The container defines WSRF WS-Resources for grid services deployed in the container. Thus, any GT4 container can host many services to multiple jobs simultaneously.

The container supports custom web services and host services whose client interfaces make use of WSRF and related mechanisms. The container also hosts advanced services provided by GT4, such as GRAM, MDS, and RFT. The application clients use the Globus container registry interfaces to determine which services are hosted in a particular container. The container administration interfaces are used to perform routine management functions. At the time of this writing, there are web service containers written in Java, C, and Python in the GT4 suite.

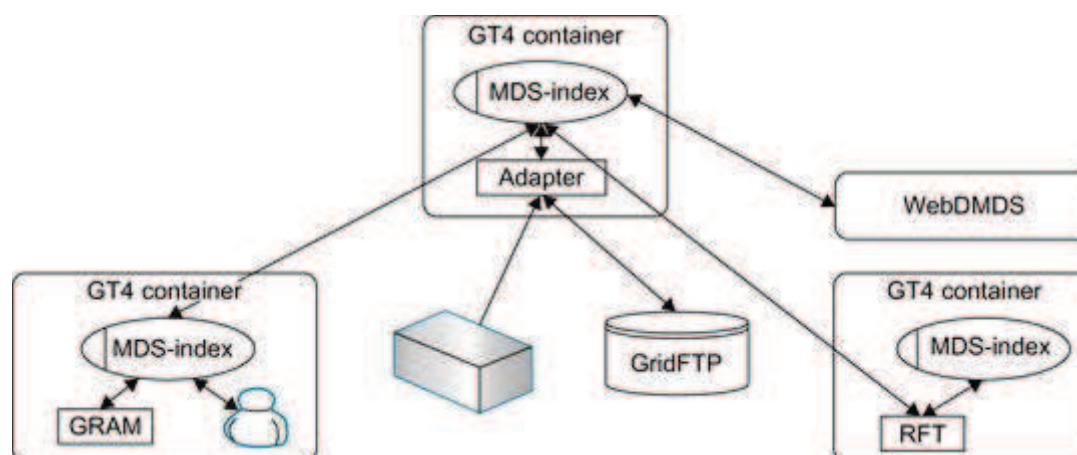
### 7.4.3.3 Data Management Using GT4

Grid applications often need to provide access to and/or integrate large quantities of data at multiple sites. The GT4 tools can be used individually or in conjunction with other tools to develop interesting solutions to efficient data access. The following list briefly introduces these GT4 tools:

1. **GridFTP** supports reliable, secure, and fast memory-to-memory and disk-to-disk data movement over high-bandwidth WANs. Based on the popular FTP protocol for Internet file transfer, GridFTP adds additional features such as parallel data transfer, third-party data transfer, and striped data transfer. In addition, GridFTP benefits from using the strong *Globus Security Infrastructure* (GSI to be studied in [Section 7.5.5](#)) for securing data channels with authentication and reusability. It has been reported that the grid has achieved 27 Gbit/second end-to-end transfer speeds over some WANs.
2. **RFT** provides reliable management of multiple GridFTP transfers. It has been used to orchestrate the transfer of millions of files among many sites simultaneously.
3. **RLS** (Replica Location Service) is a scalable system for maintaining and providing access to information about the location of replicated files and data sets.
4. **OGSA-DAI** (Globus Data Access and Integration) tools were developed by the UK e-Science program and provide access to relational and XML databases.

### 7.4.3.4 The MDS Services

Monitoring and discovery are two vital functions in any distributed system. Both tasks require the ability to collect information from multiple distributed information sources. GT4 provides monitoring and discovery support at a fundamental level. [Figure 7.22](#) shows the Globus MDS infrastructure. By implementing WSRF and WS-Notification specifications, GT4 enables the association of XML-based resource properties with network entities. Multiple containers are built to carry out MDS indexing, GridFTP, and other functions needed to conduct monitoring and discovery of available resources. Grid services can be registered with distributed containers.



**FIGURE 7.22** GT4 system monitoring and resource discovery infrastructure. Courtesy of Ian Foster [[17](#)]

Furthermore, GT4 provides two aggregator services that collect data from any

information source, whether XML-based or otherwise. In addition, GT4 provides a range of browser-based interfaces, command-line tools, and web service interfaces that allow users to query and access the collected information. In particular, the WebMDS service is configured via XSLT transformations to create specialized views of the index data. These mechanisms provide a powerful framework for monitoring diverse collections of distributed components. Details of GRAM, MDS, and other interesting tools are available at the GT4 web site: <http://www.globus.org/>.

## 7.4.4 The ChinaGrid Support Platform (CGSP)

CGSP was developed by 20 universities in China for building ChinaGrid. The package offers the first step toward multidomain web services. Unlike GT4, CGSP offers a software platform rather than a toolkit library. CGSP covers everything from user interfaces to the integration of heterogeneous resources for a given application. It brings the user closer to both grid developers and field experts. CGSP has been tested in image processing, bioinformatics, distance education, computational fluid dynamics, and large-scale information processing on ChinaGrid.

Table 7.8 shows the functional blocks built in CGSP. ChinaGrid aims at constructing a public grid service system for research and higher education in China. CGSP integrates all sorts of heterogeneous resources, especially educational and research resources distributed over CERNET in China. CGSP provides a grid platform for grid construction from the top portal level to the integration of underlying resources. This system supports uniform management of heterogeneous resources, portal building, job definitions, service packaging, and grid monitoring. It provides an extensible grid framework to serve primarily the top 100 universities in China.

---

**Table 7.8**  
**Functional Components of the CGSP Library**

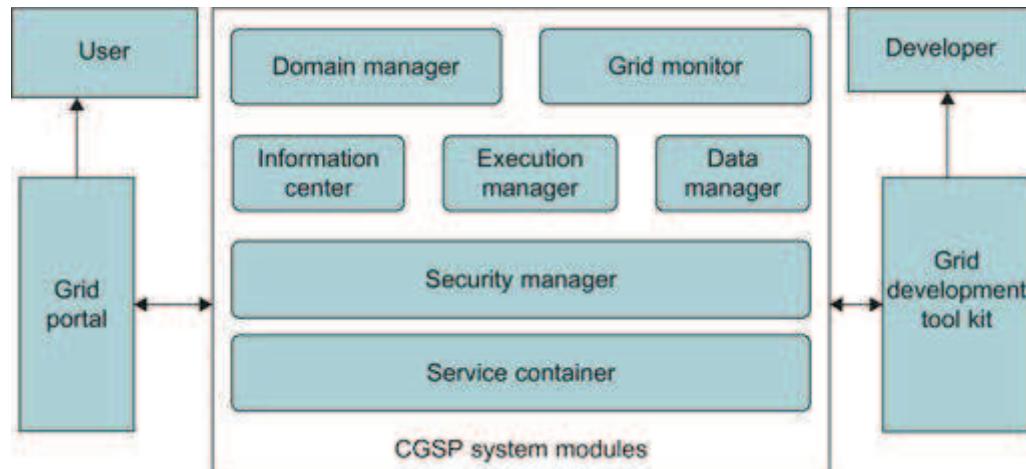
---

Component	Brief Description
Service container	Produces a basic environment for the installation, deployment, running, and monitoring of services, especially CGSP kernel services
Security manager	Focuses on user identity authentication, identity mapping, service and resource authorization, and secure message passing between CGSP nodes
Information center	Provides service registration, publishing, metadata management, service querying, service matching, and resource status collection for the CGSP services in a uniform way
Data manager	Shields users from the heterogeneousness of underlying storage resources to provide a uniform storage resource access mode
Execution manager	Accepts users' job execution requests and invokes relevant services according to the job description, and manages the job during its lifetime
Domain manager	In charge of user management, logging and accounting, and user identity mapping between different CGSP domains
Grid monitor	Named ChinaGrid Super Vision (CGSV), mainly focuses on the monitoring of CGSP resource load, QoS, user actions, job status, and network, to ensure that the system is running normally and to enhance grid performance
Portal	A web page for learning about CGSP services and providing support for new grid applications

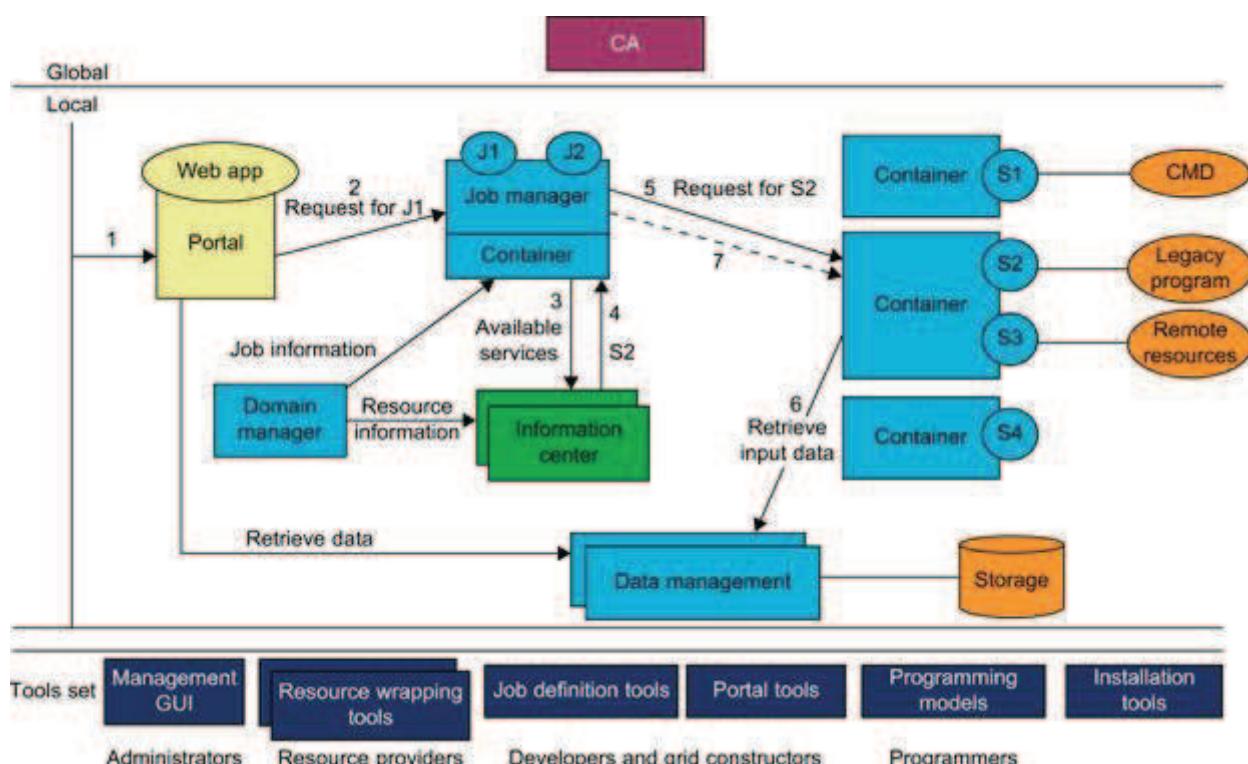
The CGSP software platform meets the OGSA standard and the WSRF criteria. All software, hardware, storage, and network technology becomes virtual resources to the user. Like Globus, CGSP emphasizes a high degree of resource sharing and cooperation among resources. CGSP comprises nine major components, as briefly introduced in [Table 7.8](#). In addition, CGSP provides a set of APIs and toolkits that enable end users and developers to write grid-enabled programs, and query and deploy their applications.

#### 7.4.4.1 CGSP Job Execution Flow

[Figure 7.24](#) shows the flow of jobs in CGSP on a single domain. First, the grid service containers should include the allocated resources. The resources are extracted through a service packaging tool and are included in the service containers. The three symbols, S1, S2, and S3, represent the command-line software, legacy program, and service requested, respectively. We need to use a job definition tool to model the workflow of a job. Because job requests are submitted via SOAP messages, the job manager is deployed in the service container.



**FIGURE 7.23** Functional building blocks in the CGSP library for ChinaGrid, developed by Tsinghua University, China [53].



**FIGURE 7.24** The job execution sequence in using the CGSP on ChinaGrid [53].

The job manager is responsible for job request analysis, service selection, and job scheduling and execution management. Each domain needs to deploy a service manager which collects service information from containers to support other software modules. In addition, each domain has to deploy a domain manager to implement the functions of user management and other issues. Lastly, one or more grid portals can be deployed to satisfy specific needs of different specialties. A portal development tool can help users to develop web applications that satisfy interaction requirements and can be deployed in the portal.

After an application is deployed, the following steps are taking place, as illustrated in Figure 7.24 in 8 steps. More details can be found in [53].

1. The user should register and acquire the certificate signed by ChinaGrid. Then the user logs in to the grid portal through a certain security protocol (such as HTTPS) and interacts with the grid system through web applications deployed in the portal.
2. If necessary, the user uploads the input data required by the computing job/request to the personal data space in the data manager. Then the user can define a job (such as J1) by using the job defining tool. The job definition is then sent to the job manager.
3. According to the job process rule in J1, which is stored locally, the job manager queries available services from the information center and helps to complete job J1.
4. The information center collects the service information using a “push/pull” model. It maintains the resource view so that it can return an available service list sorted by a certain rule. In principle, the information center will return local available services. If necessary, it will query other information centers for more available services if it cannot find a service satisfying the requirement in its own domain.
5. The job manager selects service S2 and sends it to the allocated computing node.
6. After service containers receive the requests, they need to invoke the access interfaces

provided by the data manager in order to get the data by a logical file path in the unified data space. Then the service containers execute the services.

7. The job manager continuously collects the services' status in the execution processes to report the information to the user through the portal.
8. When the job is completed, the service container moves the computing results to the personal data space in the data manager.

## 7.5 Grid Application Trends and Security Measures

In this final section, we assess the application trends of grid computing, in line with the progress of related theories and technologies such as game theory, fuzzy trust model, virtualized data centers, and the growth of the clouds. Users and resources from multiple administrative organizations are involved in a grid. We need to prevent not only network attacks to data and resources, but also combat the selfishness of grid users and remove the distrust among users.

More importantly, we should be aware of the security challenges in grid applications. We need to establish a trust mode for grid computing. The principles of some grid security enforcement schemes and mechanisms are presented, based on the lessons learned from the GridSec Project at the University of Southern California [29,39,40]. Specifically, we describe methods to cope with the problems of user authentication and resource-access authorization.

### 7.5.1 Grid Applications and Technology Fusion

Grid computing has been applied to solve computation-intensive problems in scientific discovery, engineering design, environment protection, and government management areas. The virtual grids have been applied in solving difficult academic and mathematical problems through volunteer computing. In commercial enterprise, many companies are using grids for drug discovery, economic forecasting, seismic analysis, e-commerce, and cloud services. We see a trend that calls for the fusion of computing, communication, and future Internet technologies. In what follows, we first review the theories and technologies that are relevant to grid computing. Then we present an example to use game theory to cope with the selfishness problem in P2P and grid users.

#### 7.5.1.1 Relevant Theories and Technologies

Grid computing is related to many other computing concepts and technologies. Just to name a few: clouds, clusters, P2P, parallel processing, and distributed computing are closely relevant to the development of grid technology. Other areas like edge computing, grid file systems, e-science, metacomputing, scientific workflow systems, and semantic grids are all related. In [Table 7.9](#), we identify a few related theories and technologies that have impacted the development of efficient grids in the future. The relevant chapters or sections in this book are also identified in the table. In [Example 7.11](#), we discuss how game theory can be applied to upgrade the grid performance in real-life practice.

**Table 7.9**

## Grid Computing and Relevant Theories and Technologies

Theory/Technology	Fusion Trend and Relevant Chapters
Cloud Computing	Cloud and grid technologies share some common resources and infrastructures. They are gradually merging in high-tech and service industries. ( <a href="#">Chapters 4–6</a> )
P2P Computing	Volunteer grid computing can benefit from P2P computing technology ( <a href="#">Chapter 8</a> )
HP and HA Cluster Systems	Supercomputers appear in various network platforms, not restricted to centralized MPP or high-performance or high-availability clusters ( <a href="#">Chapter 2</a> )
Data Center Automation	Through virtualization, the boundaries among data centers, clouds and supercomputers are blurred. ( <a href="#">Chapters 3, 4, 9</a> )
Fuzzy Theory and Reputation Systems	Fuzzy theory and reputation systems can be applied for trust management in grids, clouds, and any distributed systems ( <a href="#">Sec. 7.5.2</a> , <a href="#">Sec. 7.5.3</a> and <a href="#">Sec. 8.4 [42]</a> )
Game-Theoretic Optimization	Game theory can be applied to achieve win-win optimization and avoid selfish behaviors in degrading grid and cloud applications ( <a href="#">Example 7.11</a> and <a href="#">[29]</a> )

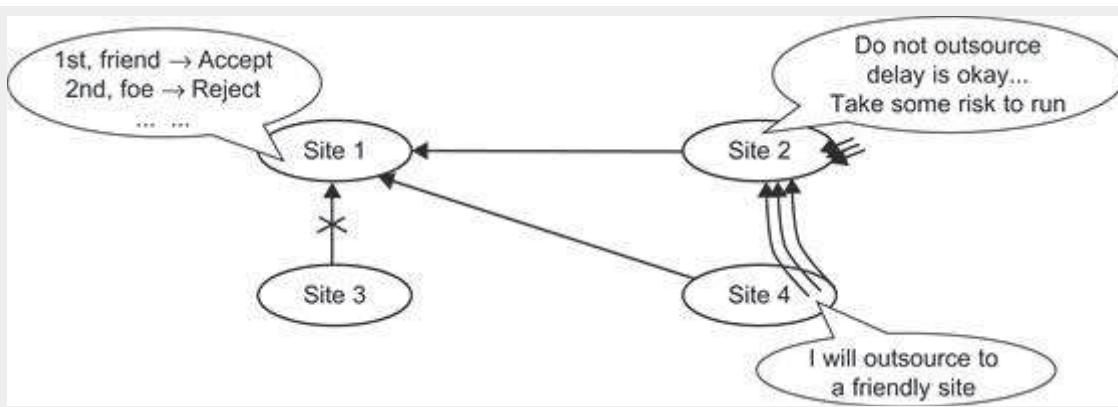
### 7.5.1.2 Fusion of Technologies with Theories

Many theories and mathematics such as graph theory, fuzzy sets, game theory, etc. can be applied to upgrade the computing technology, if proper fusion among them is performed. For example, fuzzy theory and reputation systems can be applied for trust management in grids and P2P systems [41]. The performance of grids, clouds, and distributed systems can be improved with optimization and game theory [30,42]. In [Example 7.11](#), we show how to apply game theory to achieve a win-win optimization situation and to avoid selfish behaviors in degrading the grid and cloud applications.

#### Example 7.11

#### Selfish Grids and A Game-Theoretic Solution

Grid resource owners have the tendency to give higher priority to schedule jobs submitted by local or trusted users, against the jobs from remote or unknown users. This behavior is known as *selfish grids*. A grid with a selfish job scheduling policy will downgrade the overall grid performance or quality of service to unknown users, especially in an untrusted grid user community. The selfish situation is illustrated in [Figure 7.25](#). For example, resource site 1 acts in favor of jobs from friends over jobs submitted by foes at site 3. Site 2 may refuse to outsource grid jobs at all or delay them randomly. Site 4 will encourage to outsource as much as possible. These behaviors will destroy the load balancing policy or the fairness policy by the grid schedulers.



**FIGURE 7.25** A conceptual example to illustrate the selfish grid problem and a game-theoretical solution. Courtesy of Kwok, Hwang, and Song [30], 2007

That is why the overall grid performance may suffer. In 2007, Kwok, Hwang, and Song [30] have proposed a game-theoretic approach to solving this selfish problem. The idea is to model the grid resource allocation as a non-cooperative game among multiple players. We need to provide a strategic policy to achieve an all-win situation by all parties (users) involved. All user jobs competing for the same resources pool must interact with each other to reach a *Nash Equilibrium* [35], by which the grid system can achieve a “semi-optimal” throughput among all jobs processed. The authors also verified the game-theoretic scheduling strategy by NAS benchmark experiments to show its effectiveness. The proof can be found in [30].

## 7.5.2 Grid Workload and Performance Prediction

Grid performance is directly related to the collective workload to be executed on a large number of processors in participating grid sites. Predicting the collective grid workload is a very challenging task, because heterogeneous resources are highly distributed under the control of different organizations. The grid workload is represented by a collective *load index* among all processors. The load index  $X(t)$  is the percentage of processors utilized within a unit time interval  $[t - 1, t]$ . All discrete time instances  $t$  are denoted by non-negative integers. For simplicity, assume five minutes per time step. Load index reflects the CPU utilization rate among all processors in a grid. For example,  $X(t) = 0.45$  implies that 45 percent of the processors are busy during the observation period.

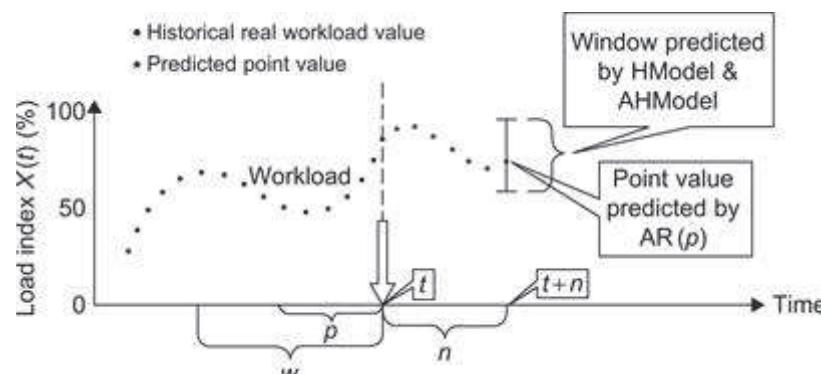
Traditional point-value prediction methods apply a very short prediction window. Although they may work well to predict CPU load in centralized computer systems, they do not work well on large-scale production grids due to the long execution time expected. In fact, point-value prediction can hardly cover workload fluctuation in a long time frame. In general, the load index is used to estimate the percentage of peak performance achievable on a given computational grid. The management console of such a grid can monitor CPU utilization.

Workload managers in large grid infrastructures are notoriously weak in determining

correct scheduling scenarios, which affects the application execution time on the grid. This section discusses predicting future workload within a reasonable confidence range. The narrower the prediction range, the higher the accuracy of prediction. Long-range workload prediction is bound to have some errors. But we try to minimize these prediction errors by using look-ahead filtering techniques within a trainable confidence window.

### 7.5.2.1 Adaptive Workload Prediction

We introduce in [Figure 7.26](#) an adaptive workload prediction process by a time series of events. The current time is marked at  $t$ . The real workload is shown by a series of dots up to time  $t$ . The future load indices can only be predicted beyond time  $t$ . The conventional prediction scheme, like the  $AR(p)$  method, predicts the future load indices using load information from the past  $p$  point values. The predicted point values are marked by stars in [Figure 7.28](#). An  $n$ -step look-ahead process predicts the load index  $X(t + n)$  using all load data in a *historical interval*  $w$  plus predicted load values in a look-ahead span of  $n$  time steps beyond  $t$ . Instead of predicting a point value, which may not be accurate due to load fluctuation, we use a *confidence window* to predict  $X(t + n)$  within a range  $\{X_l(t + n), X_u(t + n)\}$ .



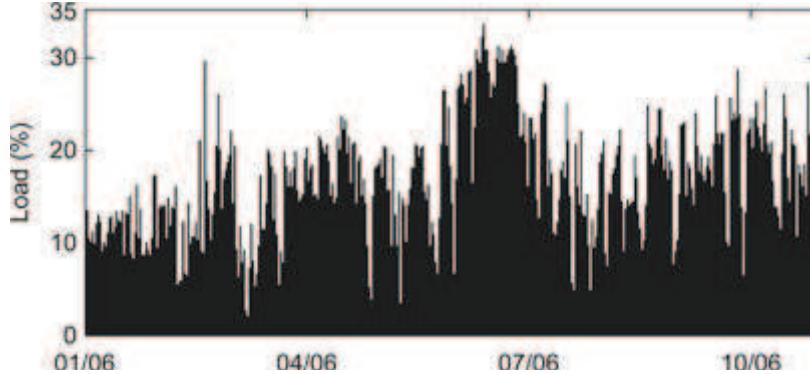
**FIGURE 7.26** Adaptive prediction of grid workload using confidence windows derived from historical load information and projected load in lookahead window. Courtesy of Wu, Hwang, Yuan, and Zheng [55], 2010

Workload traces on two French grids are introduced in this section. Sampling errors are analyzed. Two grid workload traces come from the AuverGrid and Grid'5000 in France. AuverGrid is a production grid, built over five clusters totaling 475 processors, which are geographically scattered around the Auvergne area of France. Grid'5000 is an experimental grid platform, built over 5,000 processors in nine sites throughout France. The *Grid Workloads Archive* (GWA) collected the workload traces.

### 7.5.2.2 Workload Traces on Grid'5000

The grid workload traces on Grid'5000 are shown in [Figure 7.27](#) based on traced experiments conducted in 2006. Job log entries of Grid'5000 consist of two types: those that ran successfully and those that failed. Measurement errors were possibly caused by

failed jobs or by erroneous job entries. The average workload on Grid'5000 is 10.6 percent. During an observation period of 10 months, Grid'5000 was lightly loaded and fluctuated around the average load. We collected three traces: Trace 1 for sudden local changes, Trace 2 for sudden level changes, and Trace 3 for gradual level changes in these performance prediction experiments.



**FIGURE 7.27** Workload variation in the Grid'5000 from Jan. to Oct. 2006.

### 7.5.2.3 Auto-Regression (AR) Method

An auto-regression prediction method was reported in [18] and [46]. The method applies a time sequence of load samples,  $X(t)$ . The coefficients used in the AR method are estimated from past load data sequences. These coefficients are used to predict future load sequence. The AR prediction method is specified by 
$$\hat{X}(t) = \phi_0 + \phi_1 X(t-1) + \phi_2 X(t-2) + \dots + \phi_p X(t-p) + \epsilon_t$$
 at time  $t$ , where  $p$  is the traceback span, called the *order* of the prediction method. The first term gives an estimated value of the current load index by a weighted sum of past  $p$  load values. The  $p$  coefficients  $\{\phi_i\}$  are trained from past workload. The  $\epsilon_t$  term covers the noise at time  $t$ , where  $p$  is the traceback span, called the *order* of the prediction method.

The first term gives an estimated value of the current load index by a weighted sum of past  $p$  load values. The  $p$  coefficients  $\{\phi_i\}$  are trained from past workload. The  $\epsilon_t$  term covers the noise at time  $t$ . In essence, the  $AR(p)$  model predicts the load index  $X(t+1)$  at time  $t$  based on the past  $p$  load values  $\{X(t), X(t-1), \dots, X(t-p+1)\}$  using the coefficients

The larger the order  $p$  of the  $AR(p)$  method, the higher the prediction accuracy expected. This AR method needs to find the accurate coefficients  $\phi_i$ , based on past load indices  $X(t-i)$ . The method assumes that the series  $X(t-i)$  is linear and stationary. The load series  $X(t-i)$  should be maintained to have a zero mean value.

### 7.5.2.4 H-Model for Workload Prediction

The hybrid model (H-Model) is an  $n$ -step look-ahead workload prediction method specified in [Algorithm 7.1](#). Two linear arrays  $A_l(i)$  and  $A_u(i)$  are used to represent the lower bound and upper bound of confidence window values. The confidence range is specified by a pair of point values  $\{X_l(t), X_u(t)\}$ . After the  $n$ -step look-ahead confidence

window is computed, the Savitzky-Golay filter is used to smooth the confidence windows. The evaluation checks different trace time span  $p$  parameters in the kernel AR( $p$ ) computation against the historical interval  $w$  used. These two parameters  $\{p, w\}$  have a major impact on prediction accuracy. The optimal choice of the window size  $w$  varies with the workload.

The track span  $p$  has less impact on the prediction error. Fixed parameter values in the H-Model may not predict accurately. The confidence window  $C(t + n)$  is characterized as

, where  $L = |X_u(t + n) - X_l(t + n)|$  is the length of the confidence window predicted at time  $t + n$ . A default value of  $\varepsilon = 5\%$  is applied. In general, we should use a smaller historical interval  $w$ , when  $n$  has a smaller or moderate value (say, fewer than 30 time steps). The smaller the confidence window used, the higher the chance of aiding task scheduling or load balancing in a production grid environment.

### **7.5.2.5 Adaptive Prediction Scheme (AH-Model)**

Extending the H-Model, an *adaptive hybrid model* (AH-Model) can be implemented using the historical interval, which is dynamically trained and validated from past workload series. Two adaptations are integrated to capture the workload changes dynamically. The proper choice of historical interval  $w$  can reduce the prediction error. The purpose is to minimize the prediction errors (*MSE* or *mean square error*) by training the historical window.

#### **Algorithm 7.1**

#### **Hybrid Model for Workload Prediction within Confidence Interval**

**Input:** AR model parameter  $p$ , workload index  $X(t)$ , look-ahead span  $n$ , historical interval  $w$

**Output:** Confidence window  $\{X_l(t + n), X_u(t + n)\}$  for  $n$ -step look-ahead prediction

**Procedure:**

1. **Produce** the refined workload index  $X(t)$ ; // using Kalman filter on workload index  $X(t)$  //
2. **Generate** smoothed workload index  $X(t)$ ;  
// using Savitzky-Golay filter on refined workload index  $X(t)$ //
3. **Predict** workload index series  $[X(t + 1), \dots, X(t + n)]$   
//using Algorithm 9.1 on smoothed loads  $[X(t - p), \dots, X(t)]$  //
4. **forall**  $i = 1$  to  $n$
5. **Form**  $A(i) =$  smoothed load index  $[X(t - w + 1), \dots, X(t)]$   
followed by predicted load indices  $[X(t + 1), \dots, X(t + i)]$
6. **Compute** confidence window  $\{X_l(t + i), X_u(t + i)\}$  of  $A(i)$  to yield

$A_l(i) = X_l(t + i)$  and  $A_u(i) = X_u(t + i)$

7. endforall

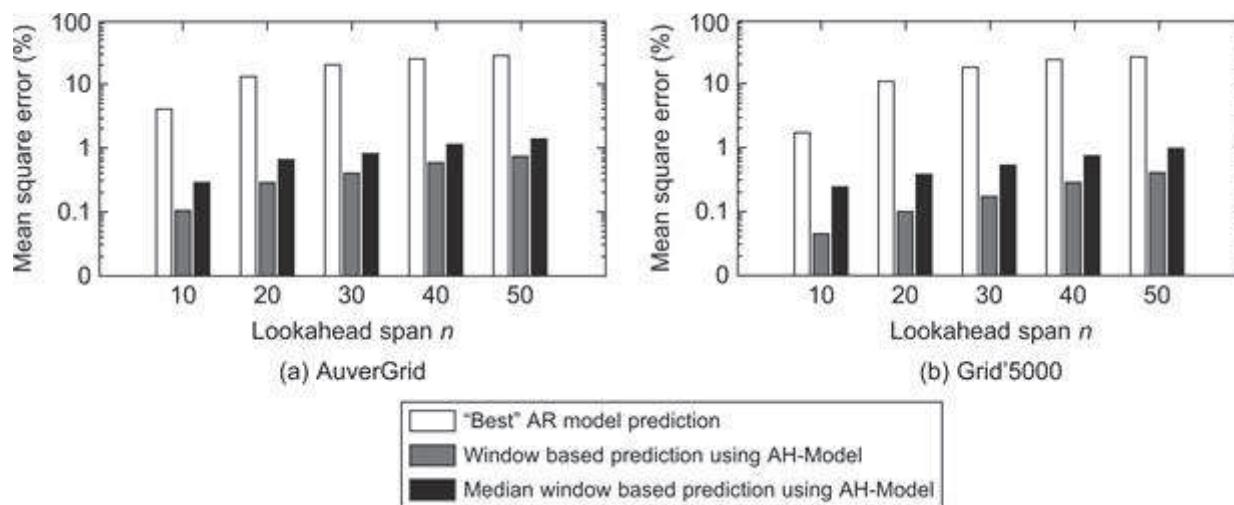
8. Generate smoothed  $A_{ls}$ ,  $A_{us}$  //using Savitzky-Golay filter//

9. Generate  $X_l(t + n) = A_{ls}(n)$ , the lower end of confidence window

Generate  $X_u(t + n) = A_{us}(n)$ , the upper end of confidence window

### 7.5.2.6 Benchmark Results on the Grid'5000 System

We report experimental results on implementing the AH-Model in both AuverGrid and Grid'5000 (Figure 7.28). The average MSEs of the AH-Model in two implementations are compared with the best AR model. The AR method leads to an MSE in the range [2% ~ 30%] on both machines. The window-based AH-Model has the lowest MSEs, between [0.1% ~ 0.7%]. We reveal the relative performance of the adaptive prediction scheme AH-Model, compared with the hybrid prediction scheme H-Model. To simplify the comparison, both schemes apply the median of the confidence window in making predictions. The results are shown in Figure 7.30 based on trace experiments on the AuverGrid and Grid'5000, respectively.



**FIGURE 7.28** Mean square errors of two implementations of AH-Model, compared with the best MSE by using the AR model. Courtesy of Wu, Hwang, Yuan, and Zheng [55], 2010

The point-value AR method is inadequate to predict workload in computational grids, because they cannot cover load variations in a long execution environment. We developed two new look-ahead workload prediction schemes for assessing long-term grid performance under fluctuating loads. We obtained encouraging results to demonstrate the effectiveness of our new prediction schemes. The main idea is to extend the point value to a confidence window, which is dynamically adjusted against load variations. The adaptation is done by adjusting the historical interval.

Many potential security issues may occur in a grid environment if qualified security mechanisms are not in place. These issues include network sniffers, out-of-control access, faulty operation, malicious operation, integration of local security mechanisms, delegation, dynamic resources and services, attack provenance, and so on. Computational grids are motivated by the desire to share processing resources among many organizations to solve large-scale problems. Indeed, grid sites may exhibit unacceptable security conditions and system vulnerabilities.

On the one hand, a user job demands the resource site to provide security assurance by issuing a *security demand* (*SD*). On the other hand, the site needs to reveal its trustworthiness, called its *trust index* (*TI*). These two parameters must satisfy a security-assurance condition:  $TI \geq SD$  during the job mapping process. When determining its security demand, users usually care about some typical attributes. These attributes and their values are dynamically changing and depend heavily on the trust model, security policy, accumulated reputation, self-defense capability, attack history, and site vulnerability. Three challenges are outlined below to establish the trust among grid sites [39].

The first challenge is integration with existing systems and technologies. The resources sites in a grid are usually heterogeneous and autonomous. It is unrealistic to expect that a single type of security can be compatible with and adopted by every hosting environment. At the same time, existing security infrastructure on the sites cannot be replaced overnight. Thus, to be successful, grid security architecture needs to step up to the challenge of integrating with existing security architecture and models across platforms and hosting environments.

The second challenge is interoperability with different “hosting environments.” Services are often invoked across multiple domains, and need to be able to interact with one another. The interoperation is demanded at the protocol, policy, and identity levels. For all these levels, interoperation must be protected securely. The third challenge is to construct trust relationships among interacting hosting environments. Grid service requests can be handled by combining resources on multiple security domains. Trust relationships are required by these domains during the end-to-end traversals. A service needs to be open to friendly and interested entities so that they can submit requests and access securely.

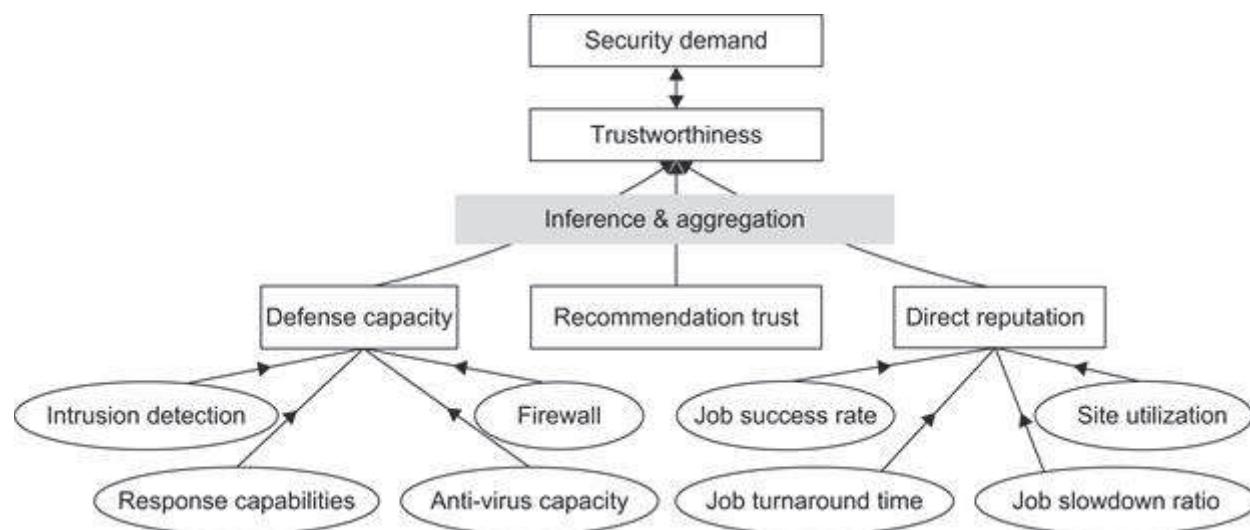
Resource sharing among entities is one of the major goals of grid computing. A trust relationship must be established before the entities in the grid interoperate with one another. The entities have to choose other entities that can meet the requirements of trust to coordinate with. The entities that submit requests should believe the resource providers will try to process their requests and return the results with a specified QoS. To create the proper trust relationship between grid entities, two kinds of trust models are often used. One is the PKI-based model, which mainly exploits the PKI to authenticate and authorize entities; we will discuss this in the next section. The other is the reputation-based model.

The grid aims to construct a large-scale network computing system by integrating

distributed, heterogeneous, and autonomous resources. The security challenges faced by the grid are much greater than other computing systems. Before any effective sharing and cooperation occurs, a trust relationship has to be established among participants. Otherwise, not only will participants be reluctant to share their resources and services, but also the grid may cause a lot of damage [63].

### 7.5.3.1 A Generalized Trust Model

Figure 7.29 shows a general trust model. At the bottom, we identify three major factors which influence the trustworthiness of a resource site. An inference module is required to aggregate these factors. Followings are some existing inference or aggregation methods. An intra-site fuzzy inference procedure is called to assess defense capability and direct reputation. Defense capability is decided by the firewall, *intrusion detection system* (IDS), intrusion response capability, and anti-virus capacity of the individual resource site. Direct reputation is decided based on the job success rate, site utilization, job turnaround time, and job slowdown ratio measured. Recommended trust is also known as secondary trust and is obtained indirectly over the grid network.



**FIGURE 7.29** A general trust model for grid computing. Courtesy of Song, Hwang, and Kwok [43], 2005

### 7.5.3.2 Reputation-Based Trust Model

In a reputation-based model, jobs are sent to a resource site only when the site is trustworthy to meet users' demands. The site trustworthiness is usually calculated from the following information: the defense capability, direct reputation, and recommendation trust. The defense capability refers to the site's ability to protect itself from danger. It is assessed according to such factors as intrusion detection, firewall, response capabilities, anti-virus capacity, and so on. Direct reputation is based on experiences of prior jobs previously submitted to the site. The reputation is measured by many factors such as prior job execution success rate, cumulative site utilization, job turnaround time, job slowdown ratio, and so on. A positive experience associated with a site will improve its

reputation. On the contrary, a negative experience with a site will decrease its reputation.

### 7.5.3.3 A Fuzzy-Trust Model

In this model [42], the job *security demand* (SD) is supplied by the user programs. The *trust index* (TI) of a resource site is aggregated through the fuzzy-logic inference process over all related parameters. Specifically, one can use a two-level fuzzy logic to estimate the aggregation of numerous trust parameters and security attributes into scalar quantities that are easy to use in the job scheduling and resource mapping process.

The TI is normalized as a single real number with 0 representing the condition with the highest risk at a site and 1 representing the condition which is totally risk-free or fully trusted. The fuzzy inference is accomplished through four steps: *fuzzification*, *inference*, *aggregation*, and *defuzzification*. The second salient feature of the trust model is that if a site's trust index cannot match the job security demand (i.e.,  $SD > TI$ ), the trust model could deduce detailed security features to guide the site security upgrade as a result of tuning the fuzzy system.

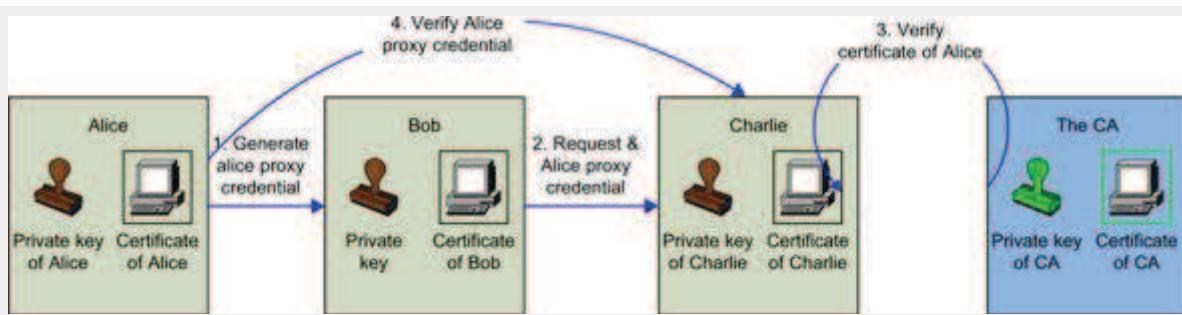
### 7.5.4 Authentication and Authorization Methods

The major authentication methods in the grid include passwords, PKI, and Kerberos. The password is the simplest method to identify users, but the most vulnerable one to use. The PKI is the most popular method supported by GSI. To implement PKI, we use a trusted third party, called the *certificate authority* (CA). Each user applies a unique pair of public and private keys. The *public keys* are issued by the CA by issuing a certificate, after recognizing a legitimate user. The *private key* is exclusive for each user to use, and is unknown to any other users. A digital certificate in IEEE X.509 format consists of the user name, user public key, CA name, and a secrete signature of the user. The following example illustrates the use of a PKI service in a grid environment.

#### Example 7.12

##### Trust Delegation Using the Proxy Credential in GSI

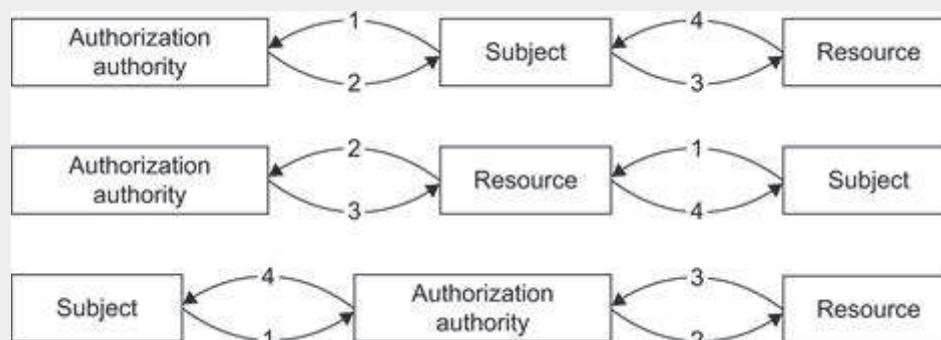
The PKI is not strong enough for user authentication in a grid. Figure 7.30 shows a scenario where a sequence of trust delegation is necessary. Bob and Charlie both trust Alice, but Charlie does not trust Bob. Now, Alice submits a task Z to Bob. The task Z demands many resources for Bob to use, independently. Bob forwards a subtask Y of Z to Charlie. Because Charlie does not trust Bob and is not sure whether Y is really originally requested by Alice, the subtask Y from Bob is rejected for resources by Charlie.



**FIGURE 7.30** Interactions among multiple parties in a sequence of trust delegation operations using the PKI services in a GT4-enabled grid environment.

For Charlie to accept the subtask Y, Bob needs to show Charlie some proof of entrust from Alice. A *proxy credential* is the solution proposed by GSI. A proxy credential is a temporary certificate generated by a user. Two benefits are seen by using proxy credentials. First, the proxy credential is used by its holder to act on behalf of the original user or the delegating party. A user can temporarily delegate his right to a proxy. Second, single sign-on can be achieved with a sequence of credentials passed along the trust chain. The delegating party (Alice) need not verify the remote intermediate parties in a trust chain.

The only difference between the proxy credential and a digital certificate is that the proxy credential is not signed by a CA. We need to know the relationship among the certificates of the CA and Alice, and proxy credential of Alice. The CA certificate is signed first with its own private key. Second, the certificate Alice holds is signed with the private key of the CA. Finally, the proxy credential sent to her proxy (Bob) is signed with her private key. The procedure delegates the rights of Alice to Bob by using the proxy credential.



**FIGURE 7.31** Three authorization models: the subject-push model, resource-pulling model, and the authorization agent model.

First, the generation of the proxy credential is similar to the procedure of generating a user certificate in the traditional PKI. Second, when Bob acts on behalf of Alice, he sends the request together with Alice's proxy credential and the Alice certificate to Charlie. Third, after obtaining the proxy

credential, Charlie finds out that the proxy credential is signed by Alice. So he tries to verify the identity of Alice and finds Alice trustable. Finally, Charlie accepts Bob's requests on behalf of Alice. This is called a *trust delegation chain*.

#### 7.5.4.1 Authorization for Access Control

The authorization is a process to exercise access control of shared resources. Decisions can be made either at the access point of service or at a centralized place. Typically, the resource is a host that provides processors and storage for services deployed on it. Based on a set predefined policies or rules, the resource may enforce access for local services. The central authority is a special entity which is capable of issuing and revoking policies of access rights granted to remote accesses. The authority can be classified into three categories: *attribute authorities*, *policy authorities*, and *identity authorities*. Attribute authorities issue attribute assertions; policy authorities issue authorization policies; identity authorities issue certificates. The authorization server makes the final authorization decision.

#### 7.5.4.2 Three Authorization Models

Figure 7.31 shows three authorization models. The subject is the user and the resource refers to the machine side. The *subject-push model* is shown at the top diagram. The user conducts handshake with the authority first and then with the resource site in a sequence. The *resource-pulling model* puts the resource in the middle. The user checks the resource first. Then the resource contacts its authority to verify the request, and the authority authorizes at step 3. Finally the resource accepts or rejects the request from the subject at step 4. The *authorization agent model* puts the authority in the middle. The subject check with the authority at step 1 and the authority makes decisions on the access of the requested resources. The authorization process is complete at steps 3 and 4 in the reverse direction.

### 7.5.5 Grid Security Infrastructure (GSI)

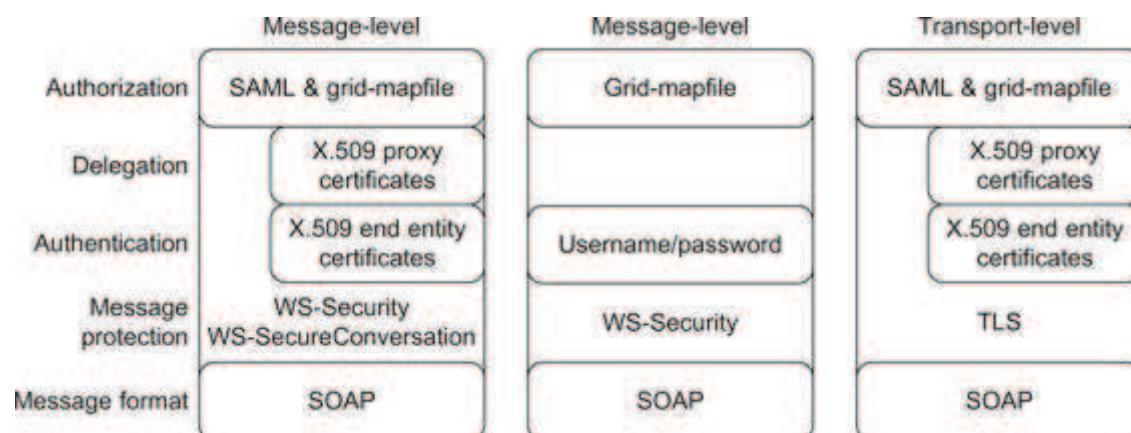
Although the grid is increasingly deployed as a common approach to constructing dynamic, interdomain, distributed computing and data collaborations, "lack of security/trust between different services" is still an important challenge of the grid. The grid requires a security infrastructure with the following properties: easy to use; conforms with the VO's security needs while working well with site policies of each resource provider site; and provides appropriate authentication and encryption of all interactions.

The GSI is an important step toward satisfying these requirements. As a well-known security solution in the grid environment, GSI is a portion of the Globus Toolkit and

provides fundamental security services needed to support grids, including supporting for message protection, authentication and delegation, and authorization. GSI enables secure authentication and communication over an open network, and permits mutual authentication across and among distributed sites with single sign-on capability. No centrally managed security system is required, and the grid maintains the integrity of its members' local policies. GSI supports both message-level security, which supports the WS-Security standard and the WS-SecureConversation specification to provide message protection for SOAP messages, and transport-level security, which means authentication via TLS with support for X.509 proxy certificates.

### 7.5.5.1 GSI Functional Layers

GT4 provides distinct WS and pre-WS authentication and authorization capabilities. Both build on the same base, namely the X.509 standard and entity certificates and proxy certificates, which are used to identify persistent entities such as users and servers and to support the temporary delegation of privileges to other entities, respectively. As shown in Figure 7.32, GSI may be thought of as being composed of four distinct functions: message protection, authentication, delegation, and authorization.



**FIGURE 7.32** GSI functional layers at the message and transport levels. See Berman, Fox and Hey [5], Foster and Kesselman [15], and Foster [17] for more details

TLS (*transport-level security*) or WS-Security and WS-Secure Conversation (message-level) are used as message protection mechanisms in combination with SOAP. X.509 End Entity Certificates or Username and Password are used as authentication credentials. X.509 Proxy Certificates and WS-Trust are used for delegation. An Authorization Framework allows for a variety of authorization schemes, including a "grid-mapfile" ACL, an ACL defined by a service, a custom authorization handler, and access to an authorization service via the SAML protocol. In addition, associated security tools provide for the storage of X.509 credentials (MyProxy and Delegation services), the mapping between GSI and other authentication mechanisms (e.g., KX509 and PKINIT for Kerberos, MyProxy for one-time passwords), and maintenance of information used for authorization (VOMS, GUMS, PERMIS).

The remainder of this section reviews both the GT implementations of each of these

functions and the standards that are used in these implementations. The web services portions of GT4 use SOAP as their message protocol for communication. Message protection can be provided either by transport-level security, which transports SOAP messages over TLS, or by message-level security, which is signing and/or encrypting portions of the SOAP message using the WS-Security standard. Here we describe these two methods.

### **7.5.5.2 Transport-Level Security**

Transport-level security entails SOAP messages conveyed over a network connection protected by TLS. TLS provides for both integrity protection and privacy (via encryption). Transport-level security is normally used in conjunction with X.509 credentials for authentication, but can also be used without such credentials to provide message protection without authentication, often referred to as “anonymous transport-level security.” In this mode of operation, authentication may be done by username and password in a SOAP message.

### **7.5.5.3 Message-Level Security**

GSI also provides message-level security for message protection for SOAP messages by implementing the WS-Security standard and the WS-Secure Conversation specification. The WS-Security standard from OASIS defines a framework for applying security to individual SOAP messages; WS-Secure Conversation is a proposed standard from IBM and Microsoft that allows for an initial exchange of messages to establish a security context which can then be used to protect subsequent messages in a manner that requires less computational overhead (i.e., it allows the trade-off of initial overhead for setting up the session for lower overhead for messages).

GSI conforms to this standard. GSI uses these mechanisms to provide security on a per-message basis, that is, to an individual message without any preexisting context between the sender and receiver (outside of sharing some set of trust roots). GSI, as described further in the subsequent section on authentication, allows for both X.509 public key credentials and the combination of username and password for authentication; however, differences still exist. With username/password, only the WS-Security standard can be used to allow for authentication; that is, a receiver can verify the identity of the communication initiator.

GSI allows three additional protection mechanisms. The first is *integrity protection*, by which a receiver can verify that messages were not altered in transit from the sender. The second is *encryption*, by which messages can be protected to provide confidentiality. The third is *replay prevention*, by which a receiver can verify that it has not received the same message previously. These protections are provided between WS-Security and WS-Secure Conversation. The former applies the keys associated with the sender and receiver’s X.509 credentials. The X.509 credentials are used to establish a session key that is used to provide the message protection.

## 7.5.5.4 Authentication and Delegation

GSI has traditionally supported authentication and delegation through the use of X.509 certificates and public keys. As a new feature in GT4, GSI also supports authentication through plain usernames and passwords as a deployment option. We discuss both methods in this section. GSI uses X.509 *certificates* to identify persistent users and services.

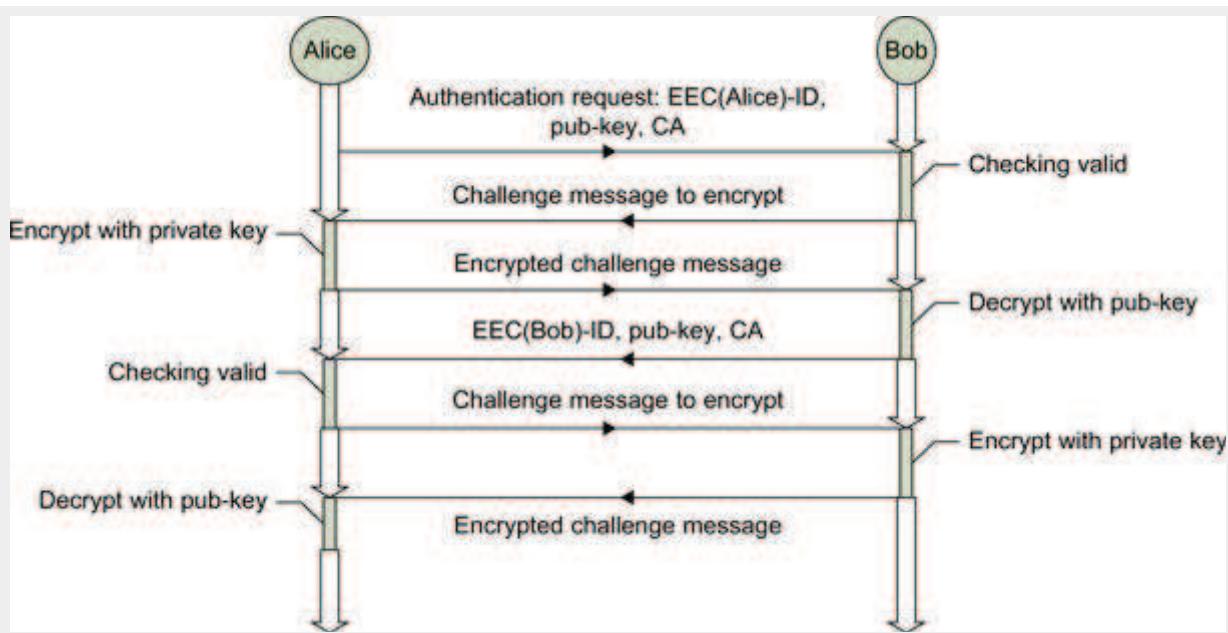
As a central concept in GSI authentication, a certificate includes four primary pieces of information: (1) a subject name, which identifies the person or object that the certificate represents; (2) the public key belonging to the subject; (3) the identity of a CA that has signed the certificate to certify that the public key and the identity both belong to the subject; and (4) the digital signature of the named CA. X.509 provides each entity with a unique identifier (i.e., a distinguished name) and a method to assert that identifier to another party through the use of an asymmetric key pair bound to the identifier by the certificate.

The X.509 certificate used by GSI are conformant to the relevant standards and conventions. Grid deployments around the world have established their own CAs based on third-party software to issue the X.509 certificate for use with GSI and the Globus Toolkit. GSI also supports delegation and single sign-on through the use of standard X.509 proxy certificates. Proxy certificates allow bearers of X.509 to delegate their privileges temporarily to another entity. For the purposes of authentication and authorization, GSI treats certificates and proxy certificates equivalently. Authentication with X.509 credentials can be accomplished either via TLS, in the case of transport-level security, or via signature as specified by WS-Security, in the case of message-level security.

### Example 7.13

#### Mutual Authentication between Two Parties

Mutual authentication is a process by which two parties with certificates signed by the CA prove to each other that they are who they say they are based on the certificate and the trust of the CAs that signed each other's certificates. GSI uses the Secure Sockets Layer (SSL) for its mutual authentication protocol, which is described in [Figure 7.33](#). To mutually authenticate, the first person (Alice) establishes a connection to the second person (Bob) to start the authentication process.



**FIGURE 7.33** Multiple handshaking in a mutual authentication scheme. Courtesy of Foster and Kesselman [15], 2002

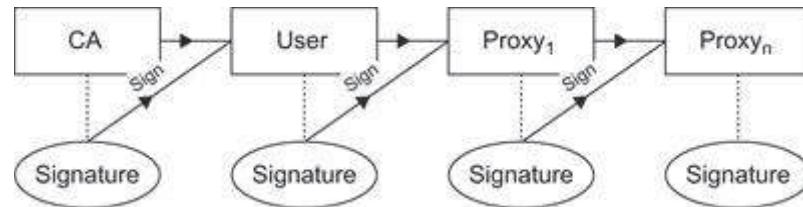
Alice gives Bob her certificate. The certificate tells Bob who Alice is claiming to be (the identity), what Alice's public key is, and what CA is being used to certify the certificate. Bob will first make sure the certificate is valid by checking the CA's digital signature to make sure the CA actually signed the certificate and the certificate hasn't been tampered with. Once Bob has checked out Alice's certificate, Bob must make sure Alice really is the person identified in the certificate.

Bob generates a random message and sends it to Alice, asking Alice to encrypt it. Alice encrypts the message using her private key, and sends it back to Bob. Bob decrypts the message using Alice's public key. If this results in the original random message, Bob knows Alice is who she says she is. Now that Bob trusts Alice's identity, the same operation must happen in reverse. Bob sends Alice his certificate, and Alice validates the certificate and sends a challenge message to be encrypted. Bob encrypts the message and sends it back to Alice, and Alice decrypts it and compares it with the original. If it matches, Alice knows Bob is who he says he is.

### 7.5.5.5 Trust Delegation

To reduce or even avoid the number of times the user must enter his passphrase when several grids are used or have agents (local or remote) requesting services on behalf of a user, GSI provides a delegation capability and a delegation service that provides an interface to allow clients to delegate (and renew) X.509 proxy certificates to a service. The interface to this service is based on the WS-Trust specification. A proxy consists of a new certificate and a private key. The key pair that is used for the proxy, that is, the public key

embedded in the certificate and the private key, may either be regenerated for each proxy or be obtained by other means. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA (see [Figure 7.34](#)).



**FIGURE 7.34** A sequence of trust delegations in which new certificates are signed by the owners rather by the CA.

The certificate also includes a time notation after which the proxy should no longer be accepted by others. Proxies have limited lifetimes. Because the proxy isn't valid for very long, it doesn't have to stay quite as secure as the owner's private key, and thus it is possible to store the proxy's private key in a local storage system without being encrypted, as long as the permissions on the file prevent anyone else from looking at them easily. Once a proxy is created and stored, the user can use the proxy certificate and private key for mutual authentication without entering a password. When proxies are used, the mutual authentication process differs slightly. The remote party receives not only the proxy's certificate (signed by the owner), but also the owner's certificate.

During mutual authentication, the owner's public key (obtained from her certificate) is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA to the last proxy through the successive owners of resources. The GSI uses WS-Security with textual usernames and passwords. This mechanism supports more rudimentary web service applications. When using usernames and passwords as opposed to X.509 credentials, the GSI provides authentication, but no advanced security features such as delegation, confidentiality, integrity, and replay prevention. However, one can use usernames and passwords with anonymous transport-level security such as unauthenticated TLS to ensure privacy.

## 7.6 Bibliographic Notes and Homework Problems

Introduction to grid computing can be found in [\[6,12,36,52\]](#). Comprehensive treatment of grid computing can be found in Foster and Kesselman [\[15\]](#) and Berman, Fox, and Hey [\[5\]](#). Foster, et al. [\[16\]](#) links grids to virtual organizations. Grid core technologies are treated in [\[18,41\]](#). The OGSA standards are specified in [\[5\]](#). The Globus Project was reported in [\[1,15,17\]](#). Grid resource management systems were surveyed in [\[29,49\]](#). Grid job scheduling and resource brokering can be found in [\[4,25,30,44,49\]](#). Resource discovery in grids are reported in Shen and Hwang [\[40\]](#) and Zhang and Schopf [\[59\]](#). The Nash Equilibrium was pioneered by Nash [\[35\]](#).

Grid monitoring and accounting systems can be found in [2,10]. The grid economy model was introduced by Buyya, et al. [8]. BOINC is treated in [7], Folding@Home in [14], and SETI in [39]. Ferreira, et al. reviewed grid computing in research and education [13]. Dacenter on Grid is studied in [37]. TeraGrid is treated in Chapter 1 in [5], where the DataGrid [21] is also covered in Chapters 15 and 16. The e-Science grid is discussed in [21] and in Chapter 36 in [5]. Grid'5000 was evaluated in [55]. Grid performance was treated in [32,34,55]. Grid performance prediction was reported by Wu, et al. [55]. ChinaGrid was reported by Jin, Wu, et al. in [20,26,54]. China's Vega grid was reported in Wang, et al. [50] and Xu, et al. [57]. The CGSP was reported by Wu, et al. [53,56].

Trust management and grid security are treated in [3,23,34,44,51]. Condor was introduced in [47] and in Chapter 11 of [5] for distributed computing. The Gridbus broker system was described by Buyya, et al. [9]. The Sun Grid Engine was treated in [45]. ChinaGrid's CCSV middleware for grid monitoring was reported by Zheng, et al. [60]. Grid security and trust management are studied in [30,34,42–44]. Additional coverage of grid software tools and grid OS support can be found in [1,10,17,28,50]. Earlier work on related scalable computing systems can be found in Dongarra, et al. [11], Taylor [46], and Xu and Hwang [22]. Various grid applications are reported in [4,5,7,15,18,48].

## Acknowledgments

This chapter was authored by Yongwei Wu of Tsinghua University (THU) China and Kai Hwang of University of Southern California, jointly. Rajkumar Buyya of the University of Melbourne has contributed Section 7.3.4. The authors appreciate the technical support from Professor Weimin Zheng of THU and Dr. Zhiwei Xu of Chinese Academy of Sciences during the course of developing this chapter.

## References

1. Allcock W, Bresnahan J, Kettimuthu R, et al. The globus striped gridFTP framework and server. In: *Proceedings of the ACM/IEEE Conference on Supercomputing*. 2005.
2. R. Aydt, D. Gunter, W. Smith, et al., A Grid Monitoring Architecture, Global Grid Forum Performance Working Group, 2002.
3. Azzedin F, Maheswaran M. A trust brokering system and its application to resource management in public-resource grids. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*. Santa Fe, NM, April 26. 2004;22a.
4. Berman F, Wolski R, Figueira S, Schopf J, Shao G. Application-level scheduling on distributed heterogeneous networks. In: *Proceedings of the ACM/IEEE Conference on Supercomputing*. Pittsburgh. 1996.
5. Wiley Series in Communications Networking and Distributed Systems . Berman F, Fox G, Hey T, eds. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley 2003.

6. V. Berstis, Fundamentals of grid computing. IBM Publication, <http://www.redbooks.ibm.com/abstracts/redp3613.html>, 2011 (accessed 26.04.11).
7. BOINCstats - Boinc combined credit overview, [http://www.boincstats.com/stats/project\\_graph.php?pr=bo](http://www.boincstats.com/stats/project_graph.php?pr=bo), 2011 (accessed 26.04.11).
8. Buyya R, Abramson D, Venugopal S. The Grid Economy. In: *Proceedings of the IEEE*. 2005;698–714.
9. Buyya R, Bubendorfer K, eds. *Market Oriented Grid and Utility Computing*. John Wiley & Sons 2009.
10. Chervenak A, et al. Giggle: a framework for constructing scalable replica location services. In: *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. Baltimore. 16–22 November 2002.
11. Dongarra, I. Fister, G. Fox, et al. *Sourcebook of Parallel Computing* Morgan Kaufman Publishers 2002.
12. L. Ferreira, et al., Introduction to Grid Computing with Globus, (<http://www.redbooks.ibm.com/abstracts/sg246895.html?OPen>).
13. L. Ferreira, et al., Grid Computing in Research and Education, (<http://www.redbooks.ibm.com/abstracts/sg246649.html?OPen>).
14. Folding@Home. (<http://fah-web.stanford.edu/cgi-bin/main.py?qtype=oosrats>), 2011, (accessed 7.03.2011).
15. Foster I, Kesselman C. *Grid2: Blueprint for a New Computing Infrastructure* Morgan Kaufman Publishers 2002.
16. Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. *Int J High Perform Comput Appl.* 2001;15(3):200.
17. Foster I. Globus toolkit version 4: software for service-oriented systems. *J Comput Sci Technol.* 2006;21(4):513–520.
18. Francesco L, et al. The many faces of the integration of instruments and the grid. *Int J Web Grid Services.* 2007;3(3):239–266.
19. Gong L, Sun SH, Watson EF. Performance modeling and prediction of non-dedicated network computing. *IEEE Trans Comput.* 2002.
20. Yuan G, Jin H, Li M, et al. Grid computing in China. *J Grid Comput.* 2004;2.
21. Hoschek W, Jaen-Martinez J, Samar A, Stockinger H, Stockinger K. Data management in an international data grid project. In: *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000)*. Bangalore, India. 17 December 2000;77–90.
22. Hwang K, Xu Z. *Scalable Parallel Computing* McGraw-Hill 1998.
23. Hwang K, Kwok Y, Song S, Cai M, Chen Yu, Chen Y. DHT-based security infrastructure for trusted internet and grid computing. *Int J Crit Infrastructures.* 2006;2(4):412–433.
24. Iverson MA, Ozguner F, Potter L. Statistical prediction of task execution time through analytical benchmarking for scheduling in a heterogeneous environment. *IEEE Trans Comput.* 1999;1374–1379.
25. Jiadao L, Yahyapour R. Negotiation model supporting co-allocation for grid

- scheduling. In: *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*. 2006;8.
- 26. H. Jin, Challenges of grid computing, *Advances in Web-Age Information Management*. Lecture Notes in Computer Science, 3739 (2005) 25–31.
  - 27. Kalantari M, Akbari M. Fault-aware grid scheduling using performance prediction by workload modelling. *J Supercomput*. 2008;46.
  - 28. Karnow C. The grid: blueprint for a new computing infrastructure. *Leonardo*. 1999;32(4):331–332.
  - 29. Krauter K, Buyya R, Maheswaran M. A taxonomy and survey of grid resource management systems for distributed computing. *Softw Pract Exper*. 2002;32(2):135–164.
  - 30. Kwok R, Hwang K, Song S. Selfish grids: game-theoretic modeling and NAS/PSA benchmark evaluation. *IEEE Trans Parallel Distrib Syst*. 2007.
  - 31. Li M, Baker M. *The Grid: Core Technologies* Wiley 2005;  
<http://coregridtechnologies.org/>.
  - 32. Li H. Performance evaluation in grid computing: a modeling and prediction perspective. In: *Seventh IEEE International Symposium on Cluster Computing and The Grid*, (CCGrid 2007). May 2007;869–874.
  - 33. Lijuan X, Yanmin Z, Ni LM, Xu Z. GridIS: an incentive-based grid scheduling. In: *Proceedings of the 19th IEEE Int'l Parallel and Distributed Processing Symposium*. April 2005;4–8.
  - 34. Lin C, Varadharajan V, Wang Y, Pruthi V. Enhancing grid security with trust management. In: *Proc of the 2004 IEEE International Conference on Services Computing*. Washington, DC. 14 September 2004;303–310.
  - 35. Nash J. Non-Cooperative Games. *Ann Math Second Series*. 1951;54(2):286–295.
  - 36. Plaszczak P, Wellner R. *Grid Computing: The Savvy Manager's Guide* Kaufmann 2006.
  - 37. Poess M, Raghunath N. *Larege-scale data warehouses on grid* 2005;  
<http://www.vldb2005.org/program/papertue/p1055-poess.pdf>.
  - 38. Schopf J, Berman F. Performance prediction in production environments. In: *12th International Parallel Processing Symposium*. Orlando, FL. April 1998;647–653.
  - 39. SETI@Home credir overview,  
[\(http://www.boincstats.com/stats/project\\_graph.php?pr=sah\)](http://www.boincstats.com/stats/project_graph.php?pr=sah), (accessed 21.04.11).
  - 40. H. Shen, K. Hwang, Locality-preserving clustering and discovery of resources in wide-area computational grids, *IEEE Trans. Comput.* (accepted to appear 2011).
  - 41. R. Smith, Grid computing: a brief technology analysis, CTO Network Library.  
[\(http://www.ctonet.org/documents/GridComputing\\_analysis.pdf\)](http://www.ctonet.org/documents/GridComputing_analysis.pdf).
  - 42. Song S, Hwang K, Zhou R, Kwok YK. Trusted P2P transactions with fuzzy reputation aggregation. *IEEE Internet Comput*. 2005;November–December:18–28.
  - 43. Song S, Hwang K, Kwok YK. Trusted grid computing with security binding and trust integration. *J Grid Comput*. 2005;3.
  - 44. Song S, Hwang K, Kwok Y. Risk-tolerant heuristics and genetic algorithms for security-assured grid job scheduling. *IEEE Trans Comput*. 2006;703–719.

45. Sun Microsystems, How Sun Grid Engine, enterprise edition works. White paper, [www.sun.com/software/gridware/sgeee53/wp-sgeee.pdf](http://www.sun.com/software/gridware/sgeee53/wp-sgeee.pdf), 2001.
46. Taylor I. *From P2P to Web Services and Grids* London: Springer-Verlag; 2005.
47. Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. *Concurrency Comput Pract Exp*. 2005;323–356.
48. Venugopal S, Buyya R, Ramamohanarao K. A taxonomy of data grids for distributed data sharing. *management, and processing ACM Comput Surv*. 2006;38(1):1–53.
49. Venugopal S, Buyya R, Winton L. A grid service broker for scheduling e-science applications on global data grids. *Concurrency Comput Pract Exp*. 2006;18(6):685–699.
50. Wang H, Xu Z, Gong Y, Li W. Agora: grid community in Vega grid. In: *Proceedings of the 2nd International Workshop on Grid and Cooperative Computing*. Shanghai, China. 7 December, 2003;685–691.
51. Welch V, et al. Security for grid services. *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing* June 22–24, 2003;48–57.
52. Wikipedia, Grid Computing. [http://en.wikipedia.org/wiki/Grid\\_computing](http://en.wikipedia.org/wiki/Grid_computing), (accessed 26.04.11).
53. Wu Y, Wu S, Yu H, Hu C. CGSP: an extensible and reconfigurable grid framework. *Lect Notes Comput Sci*. 2005;3756:292–300.
54. Wu Y, Hu C, Zha L, Wu S. Grid middleware in China. *Int J Web Grid Serv*. 2007;3(4):371–402.
55. Wu Y, Hwang K, Yuan Y, Zheng W. Adaptive workload prediction of grid performance in confidence windows. *IEEE Trans Parallel Distrib Syst*. 2010.
56. Wu YW, Wu S, Yu HS, Hu CM. Introduction to ChinaGrid support platform. In: *Proceedings of Parallel and Distributed Processing and Applications*. 2005;232–240.
57. Xu Z, Li W, Zha L, Yu H, Liu D. Vega: a computer systems approach to grid computing. *J Grid Comput*. 2004;2(2):109–120.
58. Yang L, Foster I, Schopf JM. Homeostatic and tendency-based CPU load predictions. In: *International Parallel and Distributed Processing Symposium*. 2003;42–50.
59. Zhang X, Schopf J. Performance analysis of the globus toolkit monitoring and discovery service, MDS2. In: *Proceedings of the Int'l Workshop on Middleware Performance*. 2004;843–849.
60. Zheng W, Liu L, Hu M, et al. CGSV: an adaptable stream-integrated grid monitoring system. *Network Parallel Comput*. 2005;3779:22–31 Lecture Notes in Computer Science, Springer.

## Homework Problems

### Problem 7.1

Consider the following list of resources allocation, code files, data transfers, and

computation tasks performed in typical grid experiments. You are asked to execute a typical job on a computational grid that is available to you. Discuss the resource items that are controlled by the grid resource management system on your platform. Either the Globus GT4 or the CGSP are candidate systems to use.

- a. Starting, monitoring, and controlling the execution of the resultant processes
- b. Putting and getting files
- c. Relevant load information: current load and queue state
- d. Resources allocated to network transfers
- e. Resources with advance reservation
- f. Network characteristics and load
- g. Relevant load information: available space and bandwidth utilization
- h. Versioned source and object codes

## Problem 7.2

Briefly specify the functionality of the software modules GRAM, Nexus, GSI, MDS, and GASS implemented in the Globus GT-4 middleware library. Try to illustrate an example application in using each of the five functional modules.

## Problem 7.3

Circle only one correct answer in the following question related to the use of GT4 or CGSP middleware tools for grid computing. Justify why you chose your answer. Repeat the question for those features that are not supported by CGSP.

- a. Grid workflow with WSRF support
- b. JSDL job submission and legacy job execution using General Running Service
- c. Heterogeneous integration using virtual database and virtual table.
- d. Multi-Local Scheduler support including PBS, LSF, and Condor

## Problem 7.4

Four steps are required to verify proxy credentials of two parties (Bob and Alice) in Figure 7.30. Extend the trust deletion scheme to three parties (say adding John). Draw a diagram similar to Figure 7.30 to show all the credential checking operations in a e GT4-enable grid.

## Problem 7.5

In Example 7.11, you have learned the basic concept of a game-theoretic approach to solving the grid selfishness problem. Now, you are asked to the original paper by Kwok, Hwang, and Song [30] in details on the game theory and NAS verification experiments. Discuss the advantage of the game-theoretical solution and its implementation requirements in a real-life grid environment.

## Problem 7.6

Compare three performance prediction methods: AR( $p$ ), H-Model, and AH-Model presented in [Section 7.5.2](#). Discuss their relative merits and limitations in grid workload characterization and performance prediction. Comment on the advantage use of Savitzky-Golay filter and Kalman filter in the prediction process.

## Problem 7.7

Study the Gridbus broker system in [Section 7.3.4](#). Try to use the system to schedule an application job with many tasks on an available grid or cloud platform. Report on the experimental settings and performance results obtained. Discuss the strength and weakness of the Gridbus system.

## Problem 7.8

Visit the BOINC web site: <http://boinc@berkeley.edu> and register your desktop or personal computer as a contributor machine. Join one of the listed grid application projects to gain some concrete experience in volunteer computing. Report on your research findings and discuss the experimental results obtained. Comment on the advantages of using a virtual grid platform against traditional HPC systems and identify the shortcomings or limitations encountered.

## Problem 7.9

Match the seven abbreviated terms and system models on the left with the best-match descriptions on the right. Just enter the description label (**a**, **b**, **c**, ..., **g**) in the underlined blanks in front of the specific terms. This is a one-to-one correspondence mapping.

<u>OGSA</u>	(a) A web service that provides a set of well-defined interfaces
<u>OGSI</u>	(b) A globally unique resource identifier that distinguishes a specific grid service instance from all other grid service instances, but carries no protocol- or instance-specific information such as network address or supported protocol bindings
<u>WSRF</u>	(c) A true community standard built based on two basic technologies, Globus Toolkit and web services, that supports, via standard interfaces and conventions, the creation, termination, management, and invocation of stateful, transient services as named, managed entities with a dynamic, managed lifetime
<u>Grid Service</u>	(d) A single abstraction for a grid service instance that encapsulates all the instance-specific information required to interact with a specific service instance, and can change over that service's lifetime
<u>GSH</u>	(e) The first implementation specification of a service-oriented grid, which defines a set of conventions and extensions on the use of Web Services Description Language (WSDL) and XML Schema to enable stateful web services
<u>GSR</u>	(f) The composition of a web service and a stateful resource that is (i) expressed as an association of an XML document with defined type with a web services portType, and (ii) is addressed and accessed according to the implied resource pattern, a conventional use of WS-Addressing endpoint references
<u>WS-Resource</u>	(g) A family of web service specifications and conventions designed to standardize representation of, and access to, stateful resources in a distributed environment

## Problem 7.10

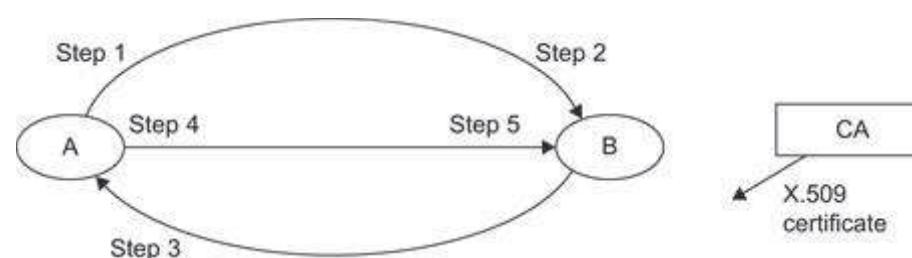
Study the grid systems: EGEE, Grid'500, DataGrid, and ChinaGrid in the examples in [Section 7.2](#). Choose one example system to dig deeper by finding more details from its Web site and published papers. Conduct an original research on its updated system development. Identify some open problems discovered and report on your research findings.

## Problem 7.11

This problem requires you to run a distributed program (MPI code) on any GT\$-enable grid platform that you have access to or you can set up with a research team. Write a report to explain the grid experimental setting using the Globus tools. Report the lessons learned and performance results obtained.

## Problem 7.12

Describe how mutual authentication is implemented between two user parties A and B in [Figure 7.35](#) across a grid using the Globus Grid Security Infrastructure and PKI. You need to describe five handshaking steps for how B verifies the identity of A. Assume that a trusted certificate authority (CA) is available to issue X.509 certificates containing the needed public keys to any grid user. Each user can apply his private key to encrypt a message. The receiving end can apply the sender's public key to decrypt the message. User A can verify the identity of B in the exact same operation, except reversing the roles. You do not need to repeat that reversed process.



**FIGURE 7.35** Mutual authentication between two parties using the GSI and X.509 certificate issued from the certificate authority (CA).

## Problem 7.13

Distinguish P2P networks from grids and P2P grids by filling in the missing entries in [Table 7.10](#). Some entries are already given. You may want to review material in [Table 1.3](#), [Table 1.5](#), and [Table 1.9](#). More material on P2P and P2P grid systems can be found in [Chapter 8](#).

## Comparison among P2P Networks, Grids, and P2P Grids

Features	P2P Networks	Grid Systems	P2P Grids
Applications and peer or node roles	Distributed file sharing, content distribution, peers acting as clients and servers		
System control and service model			Policy-based control in a P2P grid, all services from client machines
System connectivity		Static connections with high-speed links over grid resources	
Resource discovery and job management	Autonomous peers without discovery, no central job scheduler		
Representative systems			NSF TeraGrid, UK EGGE Grid, ChinaGrid

## Problem 7.14

This is a survey report about all the P2P grids or volunteer computing grids that can manage to join as contributing hosts. Compare the relative performance of all the volunteer grids you have used. Discuss their strengths and weakness in hardware capabilities, software support, and middleware services provided.

## Problem 7.15

Try to virtualize some of the resources in Problems 7.8, 7.13, or 7.14. Repeat one of the three problems in using virtual machines. Discuss the gains and losses in using a virtualized production grid. You can also use the EC 2 or other virtualized data centers to conduct the needed experiments.

# CHAPTER 8

# Peer-to-Peer Computing and Overlay Networks

## CHAPTER OUTLINE

### Summary

#### 8.1 Peer-to-Peer Computing Systems

- 8.1.1 Basic Concepts of P2P Computing Systems
- 8.1.2 Fundamental Challenges in P2P Computing
- 8.1.3 Taxonomy of P2P Network Systems

#### 8.2 P2P Overlay Networks and Properties

- 8.2.1 Unstructured P2P Overlay Networks
- 8.2.2 Distributed Hash Tables (DHTs)
- 8.2.3 Structured P2P Overlay Networks
- 8.2.4 Hierarchically Structured Overlay Networks

#### 8.3 Routing, Proximity, and Fault Tolerance

- 8.3.1 Routing in P2P Overlay Networks
- 8.3.2 Network Proximity in P2P Overlays
- 8.3.3 Fault Tolerance and Failure Recovery
- 8.3.4 Churn Resilience against Failures

#### 8.4 Trust, Reputation, and Security Management

- 8.4.1 Peer Trust and Reputation Systems
- 8.4.2 Trust Overlay and DHT Implementation
- 8.4.3 PowerTrust: A Scalable Reputation System
- 8.4.4 Securing Overlays to Prevent DDoS Attacks

#### 8.5 P2P File Sharing and Copyright Protection

- 8.5.1 Fast Search, Replica, and Consistency
- 8.5.2 P2P Content Delivery Networks
- 8.5.3 Copyright Protection Issues and Solutions
- 8.5.4 Collusive Piracy Prevention in P2P Networks

#### 8.6 Bibliographic Notes and Homework Problems

### Acknowledgements

In this chapter, we study *peer-to-peer* (P2P) networks and their applications. P2P overlay networks are built over a large number of client computers through virtual networking over the Internet. We present unstructured, structured, and hybrid overlay networks. *Distributed hash tables* (DHTs) are studied for fast routing and locality preservation operations. Fault tolerance and churn resilience are studied to improve dependability. On the application side, P2P networks are used for distributed file sharing and content distribution applications. Trust management and copyright protection are studied with reputation systems and content poisoning techniques. Examples are given to illustrate the design principles of P2P networks and the protocols applied in real-life applications. We will cover P2P social networks in [Chapter 9](#).

peer-to-peer networks, overlay networks, distributed hash tables (DHTs), routing, file sharing, copyright protection, reputation management

## Summary

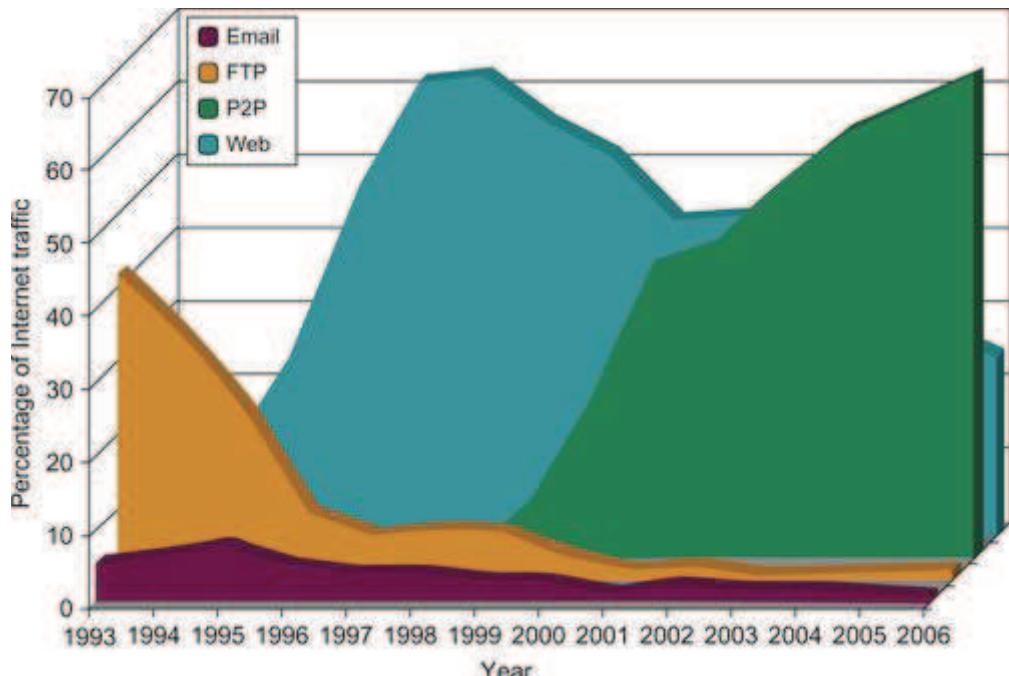
In this chapter, we study *peer-to-peer* (P2P) networks and their applications. P2P overlay networks are built over a large number of client computers through virtual networking over the Internet. We present unstructured, structured, and hybrid overlay networks. *Distributed hash tables* (DHTs) are studied for fast routing and locality preservation operations. Fault tolerance and churn resilience are studied to improve dependability. On the application side, P2P networks are used for distributed file sharing and content distribution applications. Trust management and copyright protection are studied with reputation systems and content poisoning techniques. Examples are given to illustrate the design principles of P2P networks and the protocols applied in real-life applications. We will cover P2P social networks in [Chapter 9](#).

## 8.1 Peer-to-Peer Computing Systems

P2P computing systems are widely used in distributed file sharing, messaging, online chatting, video streaming, and social networking. It has been estimated that P2P applications on the Internet accounted for about 40 percent to 70 percent of the Internet traffic worldwide from 2008 through 2010 and the trend is increasing in light of the popularity of online applications such as Facebook, Twitter, and YouTube. Unlike a traditional distributed system, P2P networks are autonomous and self-organized with free participation from the peers or clients. In a P2P network, peers share computing and data resources with others. All peers offer a mixture of online services on a voluntary basis. There are economical advantages as well as legal problems in today's P2P networks and their open applications.

As shown in [Figure 8.1](#), Internet traffic due to e-mail is quite flat and getting lower,

compared with other traffic components. FTP traffic has also declined over the past 15 years. The major traffic surge on the Internet has really been in P2P requests and responses. Even web traffic is declining with the rise of social networking and digital content downloads. By 2007, 70 percent of Internet traffic was attributed to P2P applications. This is our motivation for studying P2P systems in this chapter.



**FIGURE 8.1** The distribution of Internet traffic based on application trends. Courtesy of CacheLogic Research, 2007

## 8.1.1 Basic Concepts of P2P Computing Systems

Communication involving two equal role users can be treated as P2P communication; as such, it can be said that earlier distributed systems also followed a P2P model. Basically, P2P computing is a technique that takes advantage of computational resources (such as storage, CPU cycles, and bandwidth) and content resources (such as content files) available at the edges of the Internet to complete huge tasks, such as distribution of content to a large audience and distributed search engine and CPU-bound computing tasks.

The resources at an edge network have intermittent availability, as they are added and removed unexpectedly all the time. P2P computing eliminates the coordination by a central server, and no peer has a global view of the entire system. Instead, peers act as both servers and clients, sharing resources and services directly with one another. P2P networks share a number of common characteristics, which are summarized in the following list.

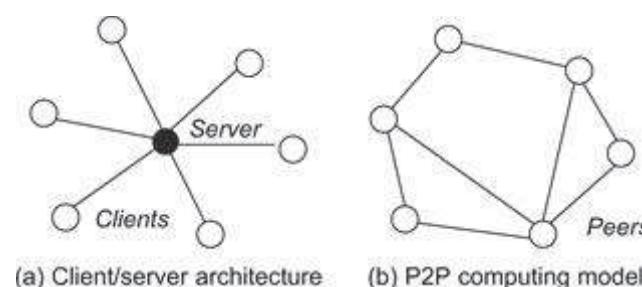
- **Decentralization** In a pure P2P computing system, peers are equal and there is no central server to coordinate the system. Each peer has a partial view of the whole system to construct the overlay network, and controls its data and resources.
- **Self-organization** Self-organization means no central management point is required to

organize the peer computing resources and the data resources spreading over the peers. The resources in P2P computing systems are dynamic or fluctuate in the sense that they come and go at any time, at will. It would be very costly to dedicate a server to manage these systems.

- **Ad hoc connectivity and dynamics** The availability of peers is unpredictable. A peer may join or leave the system at any time. The overlay topologies and the system scales will vary enormously and rapidly. P2P computing systems should provide stable services even in extra dynamic environments.
- **Anonymity** In a decentralized P2P network, peers resort to detour paths to send/receive requests (i.e., two nodes communicate through some middle nodes). This ensures senders' anonymity. Anonymity can be also achieved through hashing.
- **Scalability** The P2P model eliminates the single point of failure problem related to the traditional centralized client/server model. Each peer only maintains a limited number of system states and shares resources directly with others. These features enable the high scalability of P2P computing systems.
- **Fault tolerance** In a P2P network, all peers are created equal and no peer can dominate the system or become a single point of failure to pull down the entire system. Resources could be stored in multiple peers. These features facilitate fault tolerance.

### 8.1.1.1 Distinction between Client/Server and P2P Architectures

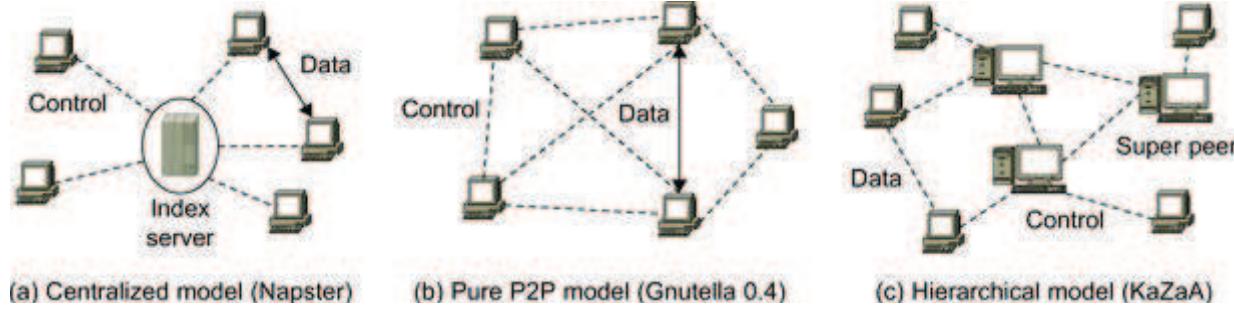
The traditional client/server architecture is formed with many client hosts surrounding a server node; see [Figure 8.2\(a\)](#). A P2P computing system is formed with peer hosts that are fully distributed without using a centralized server; see [Figure 8.2\(b\)](#). In other words, the client/server architecture is server-oriented. The server splits a task into many parts and dispatches the parts to the clients. The clients complete the task assigned to them independently and seldom communicate with one another. The clients require resources from the server and the server delivers the required resources to the clients. On the contrary, in a P2P network, participating clients (peers) all play equal roles. They exchange resources with one another directly. Peers are autonomous, self-organizing, and thus less structured, less secure, and less controllable than those of client/server systems.



**FIGURE 8.2** P2P network model versus client/server architecture.

### 8.1.1.2 Three P2P Network Models

Servers are also used in P2P networks, but they have a different role. Napster, a representative P2P file-sharing service, uses an index server to manage all file links shared by all peers. [Figure 8.3\(a\)](#) shows the Napster model. Peers register shared data information on the index server. The address information of file owners is given to the requester from the index server. Then the file is transferred between the requester and the owners directly. The Gnutella 0.4 file-sharing system adopts a pure P2P model, which is illustrated in [Figure 8.3\(b\)](#). Peers only index their own shared files.



**FIGURE 8.3** Evolution of P2P network systems from centralized Napster to flat Gnutella and hierarchically structured KaZaA using super nodes.

Since there is no central index server, file request messages are flooded to their neighbor peers. Gnutella 0.6 systems, KaZaA, and Skype leverage the third model, the hierarchical model; see [Figure 8.3\(c\)](#). A portion of powerful peers are selected as super nodes in the system. The requests are only flooded among powerful peers, hence increasing scalability. P2P distributed computing systems always maintain central servers for task management and communication to client peers. However, client peers do not need to communicate. Therefore, peers in this context refer to computing systems that make their resources available. P2P platforms act as a middleware foundation to enable easy development of P2P systems. The platforms provide security services, communication services, and standard services (indexing, searching, and file sharing).

### 8.1.1.3 P2P Applications

P2P networks are widely used in Internet applications. [Table 8.1](#) lists some typical P2P applications and examples. File sharing is the most popular P2P application. P2P content networks deliver data objects to all audiences. However, unlike client/server networks, the system uses peer resources to distribute content. Peers locate users and relay the messages directly.

**Table 8.1**

**Typical P2P Applications and Example Network Systems**

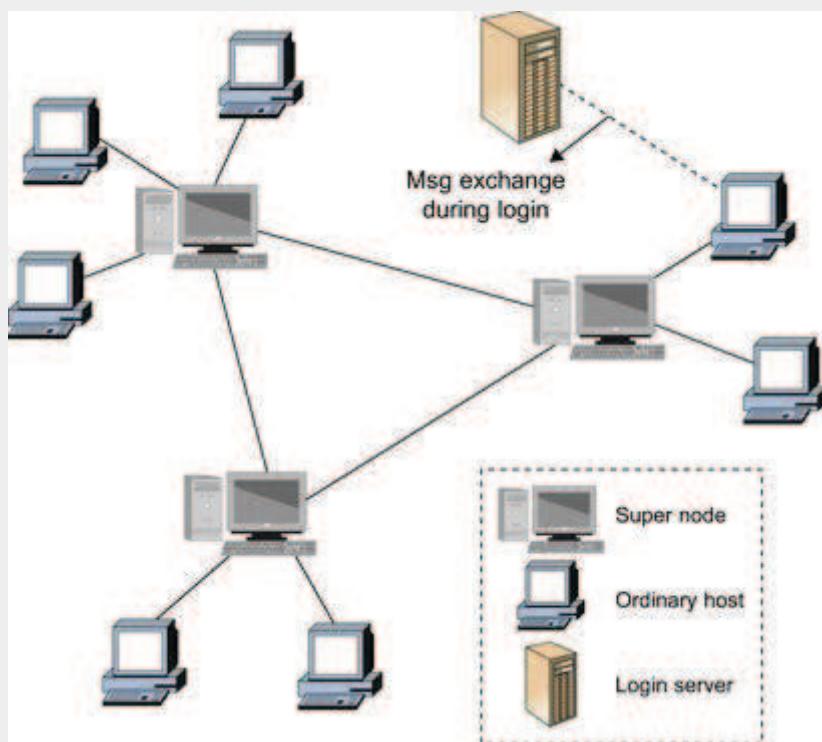
P2P Application Area	Brief Description	Examples of P2P Networks
File sharing	Queries are forwarded by peers collaboratively. File objects are transferred directly	Napster, Gnutella, KaZaA,

Content distribution	Nodes forward the content they have received to other peers until all nodes receive the content.	eDonkey BitTorrent, PP live
P2P instant messaging	Text messages or video and audio data are relayed by peers in the systems, instead of central servers.	Skype, MSN
Distributed computing	Computationally intensive tasks are split into subtasks. Nodes process subtasks independently.	SETI@Home, Folding@Home
Collaboration support	Events and messages exchanged are relayed instantly to all other peer members in the group.	Groove, Magi, AIM
Platforms	Platforms are general frameworks that provide support for P2P computing or use P2P mechanisms.	JXTA, Microsoft .NET

## Example 8.1

### The Skype Network for Online Video Telephony

Skype is a popular P2P *Voice over IP (VoIP)* service which attracts more than 500 million registered users and more than 40 million active users daily. Skype leverages P2P techniques for user profile storage, user search, media transfer, and NAT traversal. It also supports instant messaging and file transfers. Instead of the standard *Session Initiation Protocol (SIP)*, Skype uses a proprietary VoIP network. Figure 8.4 shows the P2P architecture of the Skype online VoIP system.



**FIGURE 8.4** The Skype architecture and its major components.

Skype applies the hybrid P2P model, similar to KaZaA. In fact, these two

systems are developed by the same design group. Any node with a public IP address and having sufficient CPU, memory, and network bandwidth is a candidate to be a super node. However, users cannot manually control whether it is a super node or ordinary host. An ordinary host connects to a super node, in order to join the system for VoIP services.

A super node or ordinary host is also called a Skype client. Each client maintains a list of reachable nodes. This list is called the *host cache*, and it contains the IP addresses and ports of a number of super nodes (typically 200). A number of super nodes, which are run by the service provider, are hardcoded into the client host cache. After the first login, a client learns other super nodes' information to fill its cache. The login server is the only central component in Skype. It stores usernames and passwords for user authentication. During its login process, a client first chooses a valid entry from its cache to build a TCP connection. If there is no valid entry in the cache, the client will report a login failure. The client then contacts the login server for authentication.

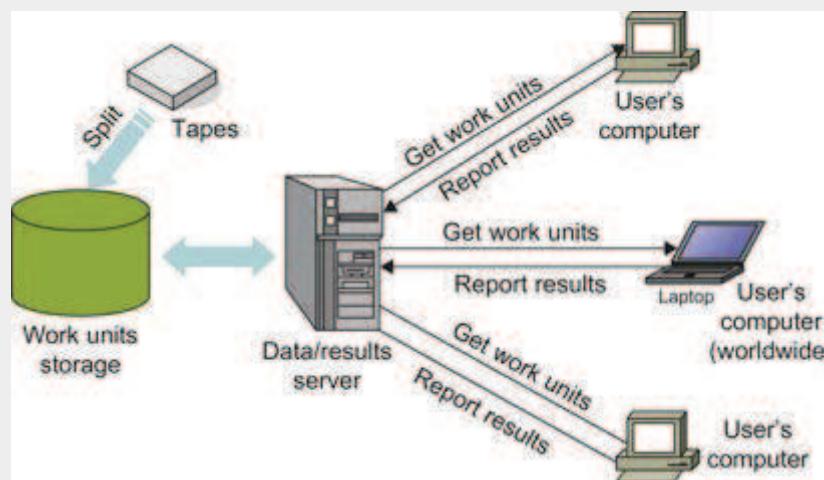
A client detects whether it is behind the network address translator (NAT)/firewall or not during its login process. Online and offline user information is stored and propagated in a decentralized fashion, and so are user search queries. Users' buddy lists are stored locally on the Skype clients. Skype claims it uses an advanced Globe Index technique for quick user search. However, it is unclear how it works since all the communications, including both control messages and media data, are encrypted. Skype clients prefer to use TCP for call signaling and UDP for media transfers. However, if it is impossible to communicate through UDP due to a NAT/firewall, TCP connections are used. In the case that the caller or callee is behind a NAT/firewall, a valid node which has a publicly accessible IP address is used for signaling and media data relay. The relay nodes may change during conservations.

## Example 8.2

### The SETI@Home Project for Distributed Supercomputing

SETI is an acronym for *Search for Extra-Terrestrial Intelligence*. The goal of SETI is to analyze radio signals from the outer space for detecting intelligent life outside Earth. The analysis work is a computationally intensive task and requires Tflops/second (*Tera-floating-point operations per second*) processing power. The SETI@Home project aims to leverage the available CPU cycles at the edge of the Internet. The project was released by the Space Sciences

Laboratory at the University of California, Berkeley in May 1999. SETI@Home has logged more than 5.2 million participants worldwide and has been considered the largest distributed computing project. Figure 8.5 illustrates the workload distribution process.



**FIGURE 8.5** Workload distribution process in SETI@Home for volunteer computing.

SETI@Home shares SERENDIP's (Search for Extraterrestrial Radio Emissions from Nearby Developed Intelligent Populations) data source, which was recorded on removable tapes and mailed to the laboratory at UCB. SERENDIP collects signal data sources from outer space at Arecibo using a telescope. The data sources are divided into small work units which are formed by dividing the 2.5 MHz signal into 256 frequency bands, each about 10 KHz wide. These work units are stored on the servers and are sent to participant computers. The programs on client sites analyze the work units during idle cycles. Finally, the results are reported back to the servers.

Users who would like to donate their computing resources download BOINC (Berkeley Open Infrastructure for Network Computing), which is a software that enables users to donate their idle computer time to science projects like SETI@Home. BOINC automatically downloads work units and manages analysis progress. The screensavers on users' computers display some analysis information when users are not using their computers. SETI@Home uses the concept of P2P in the sense that it takes advantage of available computing resources at the edge of the Internet. However, users' computers do not communicate directly.

## 8.1.2 Fundamental Challenges in P2P Computing

A P2P computing system eliminates the need for a central coordination point. This makes the system more scalable and flexible. In this section, we introduce 11

fundamental technical challenges in designing and using a P2P network in real-life applications. Solutions to these issues will be studied in subsequent sections.

### **8.1.2.1 Heterogeneity in Peer Resources**

Basically, P2P computing faces three sources of heterogeneity problems: namely *hardware*, *software*, and *network* requirements. Peers offer many different hardware hosts and platform architectures. Incompatibility does exist between the software and the OS at different hosts. Different network connections and protocols make it even more complex to apply in some real-life applications. A P2P computing system aggregates diverse resources at the peer nodes. The system should be designed to handle node heterogeneity transparently.

### **8.1.2.2 Scalability for Future Growth**

This is a fundamental goal of P2P computing systems. The system is able to support web-scale computers from hundreds to millions, and as the system size grows, the efficiency should not be affected negatively, or should just degrade gracefully. System scaling is directly related to performance and bandwidth. For example, a decentralized P2P file-sharing system uses message forwarding for information searching. If the forwarding path length grows linearly with the system size, the system cannot be scaled to a larger size.

With flooding, the messages consume a considerable network bandwidth. Gnutella version 0.4 took flooding as a query forwarding scheme, and hence was not scalable. Gnutella version 0.6 and KaZaA leverage some powerful and stable nodes as super peers. Each peer is either a super peer or assigned to a super peer. Queries are only flooded among super peers, which saves bandwidth consumption. The super-node model is more scalable in this sense.

### **8.1.2.3 Effective Search of Desired Peers**

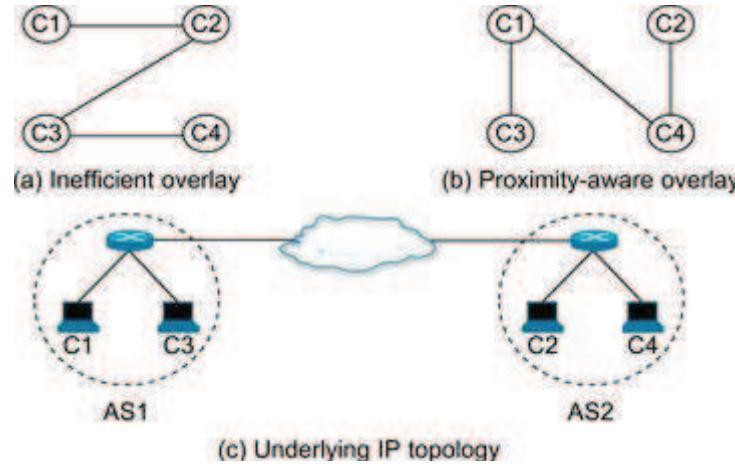
Data resources spread over the peer nodes in P2P systems. To use or to access a data object, a data locating operation (or search operation) among the peer nodes is performed first. Apparently, the data location is important in terms of affecting collective performance. However, there is no central index server for the shared resources, and each peer only has a partial view of the whole system. It is difficult to find the right peers for queries. The design of efficient data location algorithms is a unique challenge in P2P systems.

Search can be classified into two categories: blind searches and informed searches. In a blind search, peers only have the indices of their own data. When receiving a query, a peer forwards it to either all its neighbors or a subset of its neighbors without considering from which path it can receive the query response. In an informed search, peers maintain indices of the data residing on other peers. For example, a peer keeps track of the data indices of the peers that  $r$  hops away from it. Peers forward queries to the neighbors that can respond with high probability. The design challenge in a blind

search is to find the right peers quickly with low-bandwidth consumption, while the challenge in an informed search is to track peers' data indices with low control overhead.

### 8.1.2.4 Data Locality and Network Proximity

These are two design objectives in modern P2P applications. Data locality means data with similar attribute values is kept on close peers. It is an efficient way to support complex query operations and enable fast data location. Network proximity is measured by the closeness of two peers in the underlying physical IP network. By taking network proximity into account, one can build a P2P overlay network in such a way that physically close peers are connected with one another. Thus, query messages or data items are likely to be exchanged among physically close peers, which not only improves *Quality of Service* (QoS) but also saves network bandwidth. Figure 8.6 illustrates the advantage of network proximity awareness.



**FIGURE 8.6** Building a proximity-aware P2P overlay network.

Figure 6.5(c) shows a part of the IP network. C1 and C3 belong to one *autonomous system* (AS), while C2 and C4 belong to another. If the nodes form an overlay as shown in Figure 8.6(a), a message from C3 to C1 would traverse the link between two ASes two times. This prolongs the message delivery delay and consumes unnecessary backbone network bandwidth between AS1 and AS2. If network proximity is considered and a proximity-aware overlay is built as in Figure 8.6(b), the message would not transfer the link between two autonomous systems. It is a challenge to find physically close nodes in a decentralized P2P overlay with a central server and no node has the global knowledge.

### 8.1.2.5 Routing Efficiency

As in IP network, routing routines are demanded in forwarding a message to its destination. They directly affect the performance of P2P systems. A pure P2P system avoids single points of failure. However, it still faces failures related to disconnections and unreachability, partitions, and peer node failures. The system should run properly even if there are failures, and it should repair failures in an ad hoc way. Replication is an

efficient way to improve system availability and fault tolerance. Maintaining the consistency of replicas in a distributed system is always a big challenge.

### 8.1.2.6 Free-Riding Avoidance

Another challenge in P2P computing systems is the free-riding problem. P2P computing systems aggregate resources at the edge of the Internet to improve performance. The participant peers may be selfish and unwilling to contribute anything. These peers are called *free riders*. The free-riding problem is common in systems that rely on voluntary contributions from individuals since people are not willing to pay the cost of public goods when they hope someone else will bear the cost instead. Thus, incentive mechanisms to encourage contributions from individual peers are required. The incentive mechanism should evaluate peers' contributions correctly and give awards to the contributors, while maintaining anonymity and defending malicious attacks.

### 8.1.2.7 Anonymity and Privacy

Peers in P2P systems like to hide their information from others. Anonymity should be an option to peers, especially for peers in P2P communication systems. There are three levels of anonymity: namely initiator anonymity, responder anonymity, and mutual anonymity. The first two levels of anonymity can be further seen as directional anonymity, while the third level can be seen as bidirectional anonymity. Basically, directional anonymity can be implemented using onion routing.

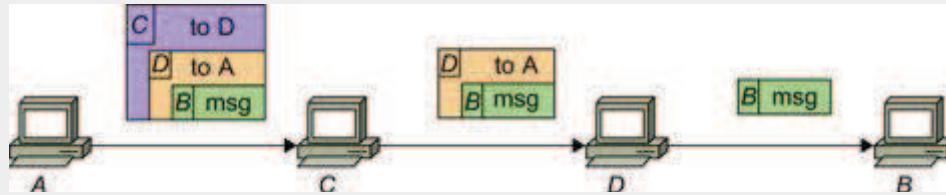
#### Example 8.3

#### Onion Routing to Protect Anonymity of Peer Machines

In onion routing, the initiator knows everything. However, intermediate routers does not. Onion routing is used to hide the identify of the originator of the final destination. This is done by encryption of the incoming message and the next hop information recursively using public keys. It creates a number of layers for the message. Each layer contains the information of the next hop nodes (called *onion routers*). Each onion router removes one layer and forwards the remaining message to the next hop onion router. This way, any node knows only its immediate sender and who is the next receiver.

Figure 8.7 shows an example of onion routing. The message is sent from A to B. The identity of A is hidden from C, D, and B, while the message itself is hidden from C and D. Onion routing can be improved using encryption with symmetric keys. Mutual anonymity requires hiding both the initiator and responder's identities, and no others are able to guess the two parties. Encrypting messages and building agents for peers are possible ways to implement mutual anonymity. However, these techniques bring high overhead and conflict with some design goals of P2P systems. For example,

using encryption incurs cryptography cost, while using agent conflicts with decentralization.



**FIGURE 8.7** An example of onion routing for anonymous communication.

### 8.1.2.8 Trust and Reputation Management

Lack of trust among peers poses another problem. Peers are strangers to one another, and inevitably some of them act evilly or in an uncooperative way. It is required in P2P systems to provide a trustworthy environment in which peers' trust is measurable and malicious peers are punishable. However, P2P systems are fully distributed. Transactions are performed directly between peers. No central server keeps track of the transactions. Hence, building a trustworthy P2P environment is a unique challenge.

### 8.1.2.9 Network Threats and Defense against Attacks

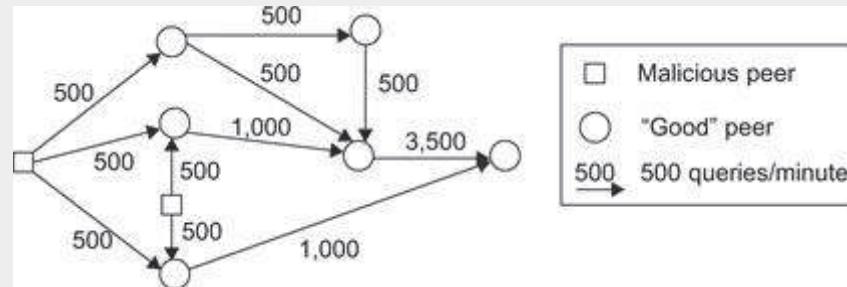
Due to the decentralization and ad hoc characteristics, attacks in P2P systems can be easily performed. *Denial of service* (DoS) and *distributed denial of service* (DDoS) attacks are possible by flooding messages to particular peers and claiming that the victim has all files requested. QoS attacks can be implemented by serving files slowly or sending different files to requesters. The anonymity design of P2P systems helps malicious peers easily hide behind others.

#### Example 8.4

##### Defense against DDoS Attacks

Figure 8.8 shows a DDoS attack by two malicious nodes through flooding queries throughout the entire network. The malicious peers flood one or more network nodes with unwanted queries or messages so that their communication buffers get overflowed, and thus TCP connections get denied. The flooded node is saturated by these messages and is unable to handle more queries from good peers. A well-designed P2P network should be able to handle this type of flooding attack to any node in the system. Sometimes the attack could be initiated from outside the P2P network and use some of

the peer nodes simply as zombies.



**FIGURE 8.8** An example of a DDoS attack in a small P2P network through message flooding.

### 8.1.2.10 Churn Resilience

The peers in P2P computing systems are from clients at the edge of the Internet. They come, leave, and even fail at random. Faulty nodes no longer forward messages. In some cases, failure of a node would cause others to disconnect from the whole system or lose index information. For example, in KaZaA, if a super node fails, the nodes assigned to it would lose connections to the system and have to find other super nodes to connect with. The index information kept in the faulty super node would be lost. Node failure imposes a great challenge to achieve fault tolerance in P2P networks. Since there is no central coordinator, peers use periodic heartbeat messages to check the status and availability of other peers.

### 8.1.2.11 Collusive Piracy Prevention

*Online piracy* has hindered the legal and commercial use of P2P file sharing. The illegal file contents are from the peers that have the legal contents and spread them to the pirates. This is called collusion. An agent recruits clients to send illegal download requests to suspected peers. For each request, if the suspected peer returns a clean copy, the feedback is 1. Otherwise, the feedback is -1. If the sum of feedbacks exceeds a predefined threshold, the suspected peer is recognized as a colluder.

Collusive piracy is the main source of intellectual property violations within the boundary of a P2P network. Paid clients (colluders) may illegally share copyrighted content files with unpaid clients (pirates). Such online piracy has hindered the use of open P2P networks for commercial content delivery. Lou and Hwang [32] have developed a proactive content poisoning scheme to stop colluders and pirates from alleged copyright infringements in P2P file sharing. The basic idea is to detect pirates in a timely manner with identity-based signatures and timestamped tokens. This scheme stops collusive piracy without hurting legitimate P2P clients by targeting poisoning on detected violators exclusively.

## 8.1.3 Taxonomy of P2P Network Systems

This section classifies the existing or proposed P2P networks by their structures and functionality. P2P network systems are identified by their names or by a short description in [Figure 8.8](#). Some of the well-known networks will be treated in more detail in examples in subsequent sections.

### 8.1.3.1 Unstructured P2P Overlays

The neighborhood relationships in an unstructured P2P overlay network are randomly connected without much constraint. Unstructured overlays are easy to build up or down when user anonymity and minimal administrative overhead are desired. At the time of this writing, many P2P networks can be started with no structural constraints. Well-known examples include Napster, Gnutella, KaZaA, Skype, BitTorrent, and eMule. Here are some interesting features in an unstructured P2P overlay:

- Data is distributed randomly over peers.
- The overlay can start with centralized control and gradually shift to fully decentralized control.
- There are no broadcasting mechanisms (or if there are, they are constrained).
- Flooding queries to entire network may generate heavy network traffic.
- There is no guarantee of a deterministic search result.
- The TTL (time to live) limit on a query message may reach the entire network.

### 8.1.3.2 Structured P2P Overlays

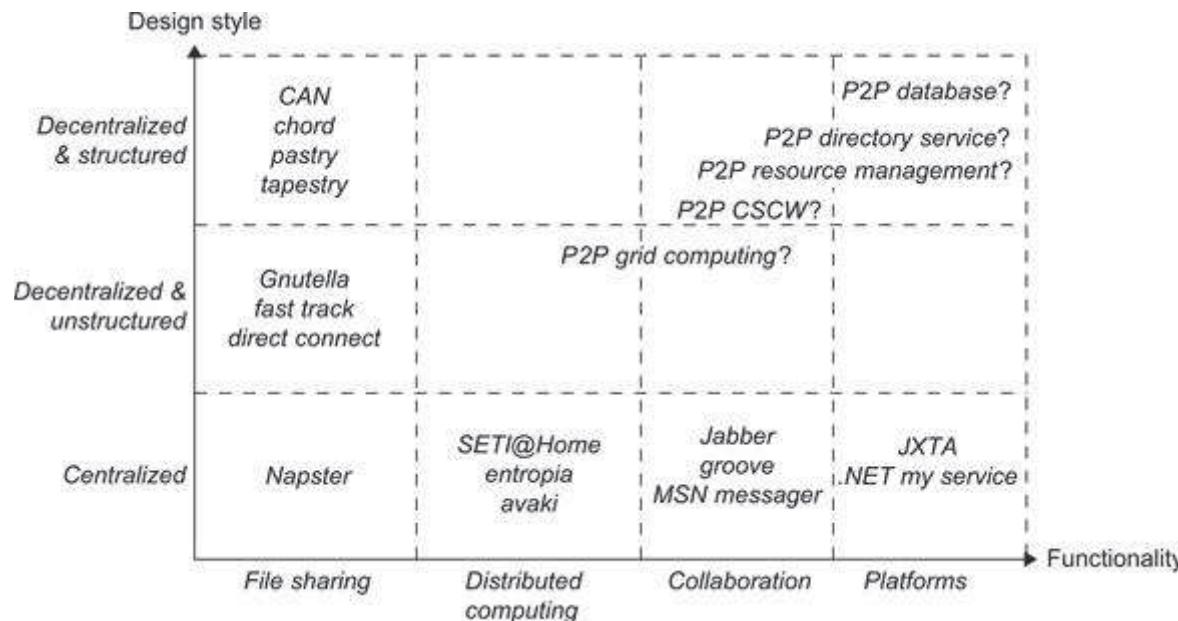
In a structured P2P overlay network, peer nodes are organized into predefined and restrictive structures, such as a ring, a chord, or a tree to be characterized in [Section 8.2](#). Good examples of structured overlays include Chord, CAN, Pastry, and Tapestry. Many P2P networks may start with a random structure, but later adopt a more efficient overlay structure to speed up searches and downloads with enhanced security and efficiency. Here are some useful features of structured overlays:

- Structured routing mechanisms in the overlay
- Add application-level overlays over the peers
- Reduce routing hops, compared with using random graphs
- Avoid flooding and eliminate hotspots
- Guarantee the search results
- Provide load balancing among the peer nodes
- Provide enhanced scalability and fault tolerance
- Preserve data locality if desired
- Promote self-organization under constraints
- Provide enhanced security protection
- Support node heterogeneity

### 8.1.3.3 Taxonomy by Structure and Functionality

[Figure 8.9](#) maps existing P2P systems according to their design style and functionality.

The bottom row denotes centralized systems built in the earlier years. The middle row covers unstructured P2P systems that are decentralized, like Gnutella, FastTrack, and DirectConnect. The upper row is for decentralized and structured P2P systems. At the lower-left corner is Napster, which is centralized and unstructured. At the upper-right corner are P2P resource managers, P2P directory services, and P2P databases that are all decentralized using structured overlays are expected.



**FIGURE 8.9** Classification of various P2P systems by functionality and control style. Courtesy of Min Cai, EE 657 Report, University of Southern California, 2007

CSCW refers to *computer-supported cooperative work*. CAN, Chord, Pastry, and Tapestry are now all using structured overlays with decentralized control. Functionally speaking, the P2P grids are mainly for distributed computing and collaboration applications that have no fixed structures. SETI@Home has centralized control for distributed supercomputing applications. The rest of the P2P networks fall in various squares in the 2D design space as shown. Note that some lattice squares are empty or are labeled with a question mark, meaning no systems are being built yet under that category.

## 8.2 P2P Overlay Networks and Properties

Overlay networks are built on top of physical IP networks. The nodes are end hosts from the physical networks, while the links are virtually built among nodes with TCP connections or simply pointers to IP addresses. The virtual links do not need to have the same weights. Different weights can be designed for different types of links. Since end hosts are dynamic, overlay networks require protocols for overlay maintenance. A new node contacts another node which is already on the overlay network for bootstrapping. Nodes in overlay networks leverage periodic heartbeat messages to verify neighbor aliveness. If a current neighbor fails, a node may connect to another node according to the maintenance protocol.

Recall that in [Figure 1.16](#), one can map some nodes from physical IP networks to an overlay network built with virtual links. The mapping is shown by the vertical dashed lines between physical hosts to virtual nodes called peers. Overlay networks do not require additional physical infrastructure; hence they are easy to deploy and use. The overlay topology can change based on application. Node failures are easily recovered since nodes can choose others to connect with. There is no restriction on the communication protocol. Application designers can define any protocol they like. The underlying physical networks are transparent to the overlay designers. However, to better utilize network resources (e.g., network proximity), one should take physical networks into consideration.

A P2P network is a type of overlay network. According to overlay graph properties, P2P networks are classified into two categories: namely *unstructured* and *structured* overlays. Unstructured overlay networks are always built based on random graphs. Nodes select their neighbors randomly from all the peers in the overlay. Structured overlay network graphs, however, have predefined structures (e.g., ring, hypercube, etc.). Each peer has a unique identifier and can only build neighborhood relationships to the peer nodes whose identifiers have satisfied the predefined requirements. Some hybrid P2P overlay networks, which leverage the advantages of both unstructured and structured overlay networks, are also used to reach the advantages of these two types of overlay networks.

## 8.2.1 Unstructured P2P Overlay Networks

To build a good unstructured P2P overlay, the node degree (i.e., the number of neighbors) of a peer and the expected number of nodes that are traversed from one node to another should be kept as small as possible. In addition, the join or departure operations cannot cause extensive modifications of the graph. Finally, the overlay can determine the forwarding path easily, when some nodes fail or depart unexpectedly.

### 8.2.1.1 Construction by Random Graphs

The Erdos-Renyi random graph [38] can be considered as a baseline model for unstructured P2P overlay networks. The probability  $p$  that any two vertices (nodes) have an edge is equal and independent. The probability that a vertex has a degree of  $k$  is as follows:

(8.1)

The expected degree of a vertices is

Apparently, if  $\lambda = np$  then for a large  $n$ ,

That is how the node degree follows a Poisson distribution. For unstructured P2P systems, the Erdos-Renyi random graph is too random to design a distributed algorithm for determination of the forwarding path from one peer to another peer.

### 8.2.1.2 Small-World Model

Small-world graphs [2] have two prominent properties: a short mean-shortest path length and high cluster coefficient. For P2P systems, these two properties are important. The former one helps us to reduce the expected hop count between peer nodes and the latter one facilitates the handling of the flash crowd. Suppose each node has two types of neighbors, namely nearby neighbors and distant neighbors. The nearby neighbors of a node  $i$  are the nodes whose distances are smaller than  $p$ . Node  $i$  also has  $q$  distant neighbors that are selected randomly from the nodes that are out of this scope. A node  $i$  builds a link to any node  $j$ , where the distance of  $i$  and  $j$  is smaller than  $p$ , that is, Node  $i$  builds a link to a distant node  $v$  with probability  $r^{d(i,v)}$  where  $r$  is the clustering exponent. A search query is forwarded to the closest neighbor to final destination. The clustering exponent  $r$  affects the search time.

### 8.2.1.3 Scale-Free Graph

The scale-free graph model is also widely used in the context of P2P applications. The node degree follows a power-law distribution, which means the probability that a node has a degree of  $k$  is proportional to  $\alpha^{-k}$  where  $\alpha$  is a constant and always in the range  $(2, 3)$ . The graph diameter (i.e., the longest minimum distance of any two nodes) is of the order of  $\sqrt{\log n}$  where  $n$  is the number of nodes in the graph. Thus, the diameter almost remains unchanged when the graph scale grows. In practice, many real networks are thought to be scale-free, such as social networks and the Internet graph.

In the Gnutella 0.4 overlay network, the node degree follows a power-law distribution with  $\alpha$  approximately equal to 2.3 [41]. This is mainly because new join nodes are likely to build overlay connections to the nodes already with high degrees. A scale-free network is also resilient to the random node failures. However, if the network is attacked by knowledgeable attackers and some of the high-degree nodes break down, the graph may be partitioned into several disjointed parts or the graph diameter will grow quickly. Thus, to build a resilient scale-free graph based on P2P overlays, high-degree node identities should be hidden if possible.

### 8.2.1.4 P2P Networks for Distributed File Sharing

The most popular applications that are built on top of unstructured P2P networks are P2P file-sharing systems. Table 8.2 compares the P2P file-sharing systems Napster, Gnutella, and KaZaA. These three systems represent three distinct types of P2P service models. Basically, in these file-sharing systems, data is distributed randomly over the peers. Flooding on the overlay is used to find files in response to requests. The TTL is imposed to limit the flooding, in order to save heavy traffic generated by flooding. Moreover, there is no guarantee on the search results.

---

**Table 8.2**

P2P File-Sharing Applications Using Unstructured Overlay Networks

---

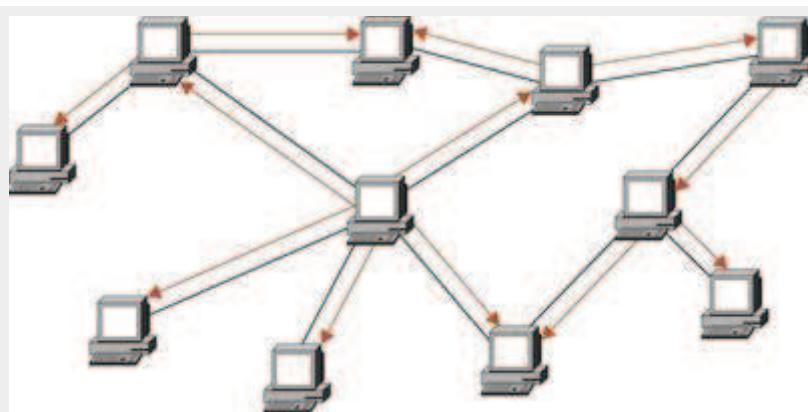
P2P Network	Napster	Gnutella	KaZaA
Architecture	Centralized model	Pure P2P model	Hierarchical with super peers
File indices	Centralized index servers keep all links	Each node keeps file indices for itself only	Super node keeps the indices for the nodes that connect to them
Search algorithm	Queries are sent to servers, which return the requested files	Queries are flooded on entire overlay Matching peers reply to originators	Queries to super nodes with flooding among themselves and reply to the originators
Search time (hops)	$O(1)$	$O(\log n)$	$O(\log m)$ , $m$ counts supernodes
Search cost (#msgs)	$O(1)$	$O(d^* n)$ , $d$ is node degree	$O(c^* m)$ , $c$ is supernode degree
File fetch method	Originator gets the file from a single peer	Query originator gets the file directly from a peer	Originator gets the file from multiple peers concurrently
Scalability	Limited by the central servers	Limited by the high search cost	Relatively good scalability due to super peers
Fault tolerance	Single point of failure	Good fault tolerance due to equality among peers	Super peers are vulnerable to attacks

The major difference among these systems lies in the fact that Napster applies a centralized control, Gnutella uses pure P2P networking, and KaZaA applies a hierarchical structure. Napster has a constant search time. Flooding in Gnutella takes the longest search time across the entire network. KaZaA performs in between, because binary search is conducted only across the super nodes. Both Napster and Gnutella have limited scalability. KaZaA has better scalability. However, these three P2P file-sharing services have all suffered greatly from illegal downloads by copyright violators in recent years.

### Example 8.5

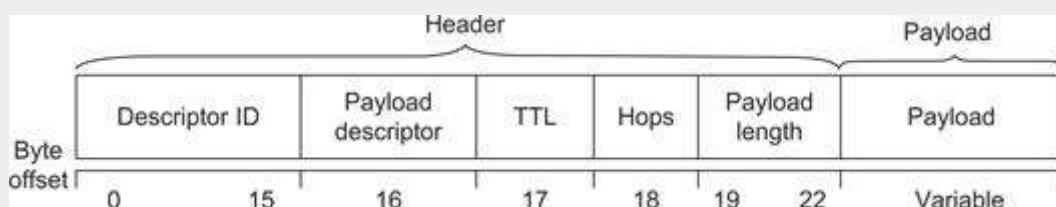
#### Gnutella (Version 0.4) Networks with Flooding Search

The Gnutella file-sharing application leverages the pure P2P model. Each peer has equal roles and acts as either a server when uploading files to others or a client when downloading files from others. A peer is also called a *servant*. Gnutella defines a number of messages, called *descriptors*. Flooding-based search is used to locate highly replicated items, as illustrated in Figure 8.10. Each node sends the requests to multiple neighboring nodes, which can send to more nodes, and thus flood the entire network. This flooding scheme may create heavy traffic to saturate the network. Therefore, the messages to send or to forward and the hop count in forwarding must be limited.



**FIGURE 8.10** Flooding scheme used in Gnutella to search for peers that can provide files of digital content.

Figure 8.11 shows the descriptor header and payload. The header consists of 22 bytes. The descriptor ID uses the first 16 bytes. It is a unique identifier for the descriptor in the network. The payload descriptor only uses 1 byte. Gnutella defines five main types of descriptor, including *Ping* (0X00), *Pong* (0X01), *Query* (0X40), *QueryHit* (0X80), and *push* (0X80). The TTL and Hops fields use one byte each. TTL is initialized by the descriptor originator and is decreased by one once the descriptor traverses one servant. When TTL reaches 0, the descriptor is discarded.



**FIGURE 8.11** Gnutella packet descriptor format.

The TTL is used to limit the horizon of the descriptor in order to save the communication cost of unlimited flooding. The Hops field counts the number of hops the descriptor has traveled. The last four bytes in the header is for payload length, which is followed by variable payload. For example, The Ping has no payload, while other four descriptors, payload is filled with bytes of variable size.

The Ping descriptor is used by servants to discover others. A servant that receives a Ping sends back one or more Pong descriptors along the path of Ping with a reverse direction. A Pong consists of the address of the responsible servant and the amount of data it shares. Query descriptors are for search, while a QueryHit descriptor is the response to a Query. The Push descriptor enables firewalled servants to share files with others. A new servant contacts bootstrap nodes to join. After joining the system, a servant communicates with others using descriptors. It sends a Ping to discover more servants to connect with.

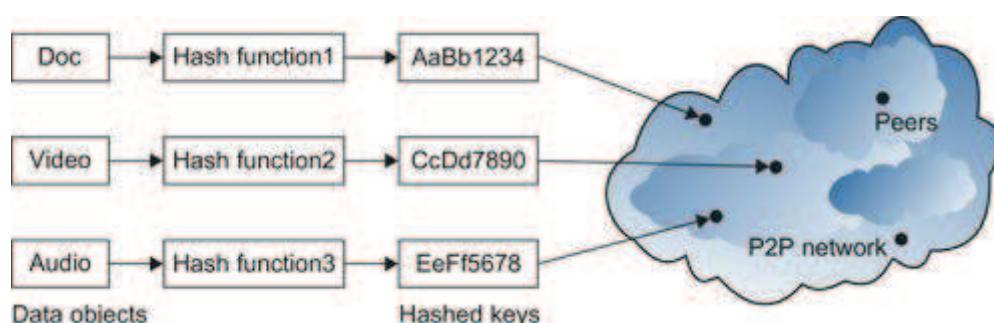
In general, the Ping and Query descriptors are sent in a flooding fashion. A servant that receives such a descriptor will forward the descriptor to its neighboring servants. Clearly, if the horizon of the flooding is not limited, such descriptors would, on average, be forwarded to  $d$  neighbors by each peer, where  $d$  is the average number of neighboring peers of each node. Thus, a  $n$ -node Gnutella system must use  $d^*n$  messages, within which  $(d-1)^*n$  messages are duplicated. Such a large number of messages limits system scalability.

Gnutella limits the horizon of flooding with a TTL in each descriptor. The TTL is decreased by one each time the descriptor is forwarded. If the TTL expires, that is, it reaches 0, the descriptor is discarded without further forwarding. By default, the TTL is set at 7 since within this horizon, 95 percent of servants can be reached. However, the requested files may be owed only by the unreached 5 percent of servants. In such cases, the requests cannot be satisfied, even though files that match the requests do exist.

The flooding search in Gnutella is suitable for highly replicated files (i.e., hay), not for rarely replicated ones (i.e., needle). Since users often request hay, Gnutella works well in practice. The request originator downloads the file from one peer directly using HTTP. From version 0.6, Gnutella leverages powerful peers as *ultrapeers* to construct hierarchical architecture similar to KaZaA's. The descriptors are only flooded among ultrapeers, greatly reducing search cost and improving scalability.

## 8.2.2 Distributed Hash Tables (DHTs)

As shown in Figure 8.12, a *distributed hash table* (DHT) is middleware which offers information search or table lookup services for a distributed system, especially for P2P systems. A hash table is often represented by  $(key, value)$  pairs. DHT stores such pairs in an *identifier space*. For example, an identifier space, represented by 64-bit binary strings, can accommodate  $2^{64}$  keys, representing the same number of data objects. Using separate hash functions, the DHT maps the data objects into the keys. The  $(key, data)$  pairs are distributed to the peers in a P2P network. The identifier of a peer is the hash value of its address using certain hash function such as SHA-1.



**FIGURE 8.12** Key mapping in distributed hash tables (DHTs).

The ownership of the identifier space is distributed among the peer nodes. DHTs provide a lookup service similar to a hash table:  $(key, value)$  pairs used in other mapping services. Any participating peer can retrieve the value associated with a given key. In a P2P network structured with DHT support, the mapping from keys to values is distributed among the peer nodes. Any peer joining or leaving should cause minimal disruption to the entire P2P network. An efficient DHT design should scale to an extremely large identifier space such as  $2^{64}$  or even greater such as  $2^{256}$ .

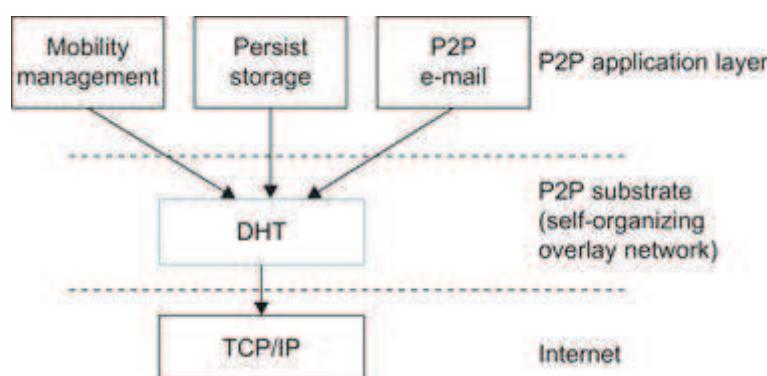
The DHT applies *consistent hashing* to map keys to nodes. This consistent hashing should be able to define an abstract notion of proximity or distance between two nodes. Keys are mapped to the nearest nodes with the minimum distance or maximum proximity. When a new node joins or an existing node leaves, consistent hashing causes a small impact on existing nodes by reassigning their keys. For  $n$  nodes with  $k$  keys, each node is responsible for, at most, \_\_\_\_\_ of the whole ID space with a high probability. The addition or deletion of a node requires \_\_\_\_\_ keys to reassign, where \_\_\_\_\_. Thus, the DHT is suitable for P2P systems where nodes leave and join frequently.

The DHT provides two primitive operations: *put* ( $key, data$ ) and *get* ( $key$ ). The *put* primitive stores the date or data index with the *key* on the node which has the closest identifier to the *key*, while the *get* primitive retrieves the node that stores the data or data index with the *key*. Peer nodes form an overlay network. Every node keeps several links to a subset of other nodes. The *get* request for a key with identifier  $k$  is forwarded over the overlay to the nearest node, until the request arrives at the node with an identifier closest to  $k$ .

Popular P2P networks that use DHTs include BitTorrent's distributed tracker, the Bitcoin monetary network, the Kad network, the Storm botnet, YaCy, and the Coral Content Distribution Network. Some prominent research projects including the Chord project; the PAST storage utility; P-Grid, a self-organized and emerging overlay network; and the CoopNet content distribution system also apply the DHT in their overlay constructions. Others apply the DHT in resource discovery in grid computing systems. Resource discovery searches for suitable resource types to match with user application demands.

### 8.2.2.1 DHT Deployment

The DHT acts as a substrate to provide two primitives: *put* ( $key, data$ ) and *get* ( $key$ ). The basic idea is to map nodes and keys into the identifier space and assign keys to closer nodes in this space. This enables fast search with guaranteed lookup success and provable bounds on the search time. Moreover, such overlay networks avoid high search costs by flooding, and thus are more scalable. However, it should be noted that DHT-based structured P2P overlays only directly support exact-match search, rather than keyword search. [Figure 8.13](#) shows the role of a DHT in mapping P2P user applications to desired source machines through proper TCP/IP connections.



**FIGURE 8.13** DHT for fast and secure search and other Internet applications.

### 8.2.3 Structured P2P Overlay Networks

In a ring structure, each node has two neighbors: its successor and its predecessor along the ring. If a node A is inserted at the interval between B and C, it has to take these two nodes as its neighbors. In a tree structure, a node can only build neighboring links with its child nodes and parent node. Although any P2P overlay network that has a logical predefined structure is considered structured, the most popular structured overlays are DHT-based.

Structured P2P networks employ a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has the desired file, even if the file is extremely rare. Such a guarantee necessitates a more structured pattern of overlay links. By far the most common type of structured P2P network is the DHT, in which a variant of consistent hashing is used to assign ownership of each file to a particular peer, in a way analogous to a traditional hash table's assignment of each key to a particular array slot. **Table 8.3** summarizes various structured P2P overlay networks; three examples follow the table. An  $n$ -node network with  $b$  bits per node identifier is considered here.

**Table 8.3**

A Comparison of Three Structured P2P Overlay Networks

Overlay Networks	Chord [45]	CAN [40]	Pastry [39]
Identifier Space	Chordal ring with direct fingers connecting to nodes in distance intervals	$d$ -dimensional coordinates	High-radix identifier space that divides nodes into nested proximity groups
Routing state	$\log n$	$2d$	$2 \times 2^b \log_2 n$
Lookup Protocol	Matching data key and node identifier	Mapping key to a point in the $d$ -dim coordinate	Matching key and prefix of node identifier
Routing complexity	$O(\log n)$	$O(d \times n^{1/d})$	$O(\log_2 n)$
Number of messages per join/leave operation	$\log^2 n$	$2d$	$\log_2 n$

- **Distributed hash tables** Use distributed hashing with key lookup to avoid flooding at the expense of losing data locality. Good examples are Tapestry, Pastry, Chord, and

CAN.

- **Tree-structure system** Preserves data locality with a tree-structured data access hierarchy. A good example is the TerraDir system.
- **Skip-list-based systems** Accelerates query processing by key ordering rather than key lookup. Two good examples are Skip Graphs and SkipNet.

Due to the random selection of neighboring peers, there often exists a topology mismatch problem between overlay and physical topologies in P2P systems. Location awareness or network proximity awareness solves the mismatch problem by allowing nodes to select neighboring peers on overlay networks according to peers' locations on physical networks.

In DHT-based structured P2P overlays, the neighbors of a node are not selected randomly from all the peers on the overlay, but only from the ones whose identifiers are subject to special requirements. Addition or deletion of a node requires few recovery operations, which facilitates large-scale deployment. There are different overlay structures for DHT-based overlay networks, which yield different structured P2P systems, such as Chord, Pastry, CAN, and Kademlia. This section focuses on the structures of these structured P2P systems. In [Table 8.3](#),  $n$  is the number of nodes in the P2P network. The quantity  $2^b$  for specifying Pastry is the base of the identifier space. The "log" implies the binary base 2.

A good example of DHT systems, *Chord*, is introduced in [Example 8.6](#). The identifier space of  $m$ -bit strings is mapped to a unidirectional logical ring modulo  $2^m$ . The distance between the identifier  $id_1$  and the identifier  $id_2$  is measured by the interval traveled from  $id_1$  to  $id_2$  in a clockwise direction. Nodes are placed at the position where the node's identifier can be found. A node with identifier  $id_x$  stores the data or data index with key identifiers falling into the interval from the immediate predecessor of  $id_x$  to  $id_x$  itself. Each node keeps  $m$  chordal links to the nodes whose identifiers immediately succeed for

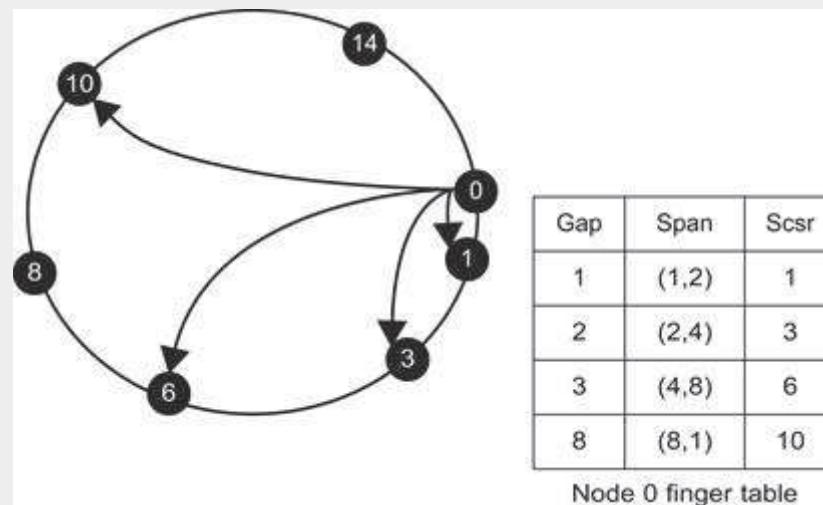
### Example 8.6

#### Chord System—A DHT-Based Chordal Ring

Chord is a typical DHT system which organizes peers in a unidirectional ring augmented with chord links for scalable search [45]. Nodes and keys are mapped to the same  $m$ -bit identifier (ID) space using SHA-1 as the base hash function. A peer's ID is generated by hashing the peer's IP address, while a key's ID is assigned by hashing the data key. Identifiers are represented on the ring modulo  $2m$ . A peer is responsible for the ring interval from the previous ID to its own ID. That is, the data (or the index) with keys mapped to this interval is stored on this peer. The distance from an ID1 to another ID2 is the interval from ID1 along the clockwise direction of the ring to ID2.

Besides the predecessor and successor on the ring, each node maintains a

*finger table* which consists of the nodes pointed by the chord links from that node. The  $i$ -th entry in a node's finger table refers to the node whose identifier is  $2^i$  away along the clockwise direction, where  $m$  is the identifier length. A 3-bit identifier ring with four nodes and the finger table of node 8 are shown in [Figure 8.14](#). Node N6 is  $2^2$  away from N1. Although there are  $m$  entries in a peer's finger table, on the average, a peer's finger table contains  $O(\log n)$  nodal information, where  $n$  is the total number of nodes in the search space.



**FIGURE 8.14** An example of a Chord network with a 16-key search space. The finger table establishes the direct links between the populated nodes located in various span intervals.

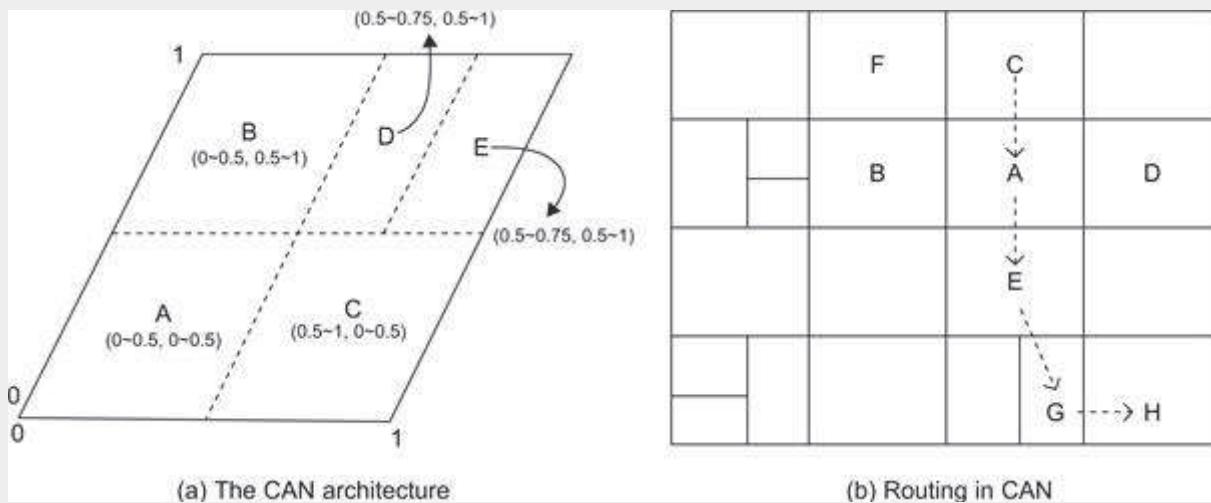
In [Figure 8.14](#), N1's finger table has three rows. However, the first two rows contain the same node's information, that is, N3's information. This is stemmed from the fact that the identifier range which starts from a node to its  $i$ -th entry is within distance  $2^i$ . Since nodes are randomly distributed on the identifier ring, an average of one node is within the range  $[id, id + 2^i]$ . Thus, on average, the number of nodes in a node's finger table is  $\frac{1}{2} \cdot \log_2 n + O(1)$ . DHT-based search is often used to locate rarely replicated items.

To improve robustness, each node maintains a successor list. The list starts from its successor down the ring. If the current successor fails, a node replaces the fault successor with the first live entry in the list. A newly coming node of ID X first contacts a bootstrap node, which forwards the join message to X's predecessor Y. The new node joins as Y's successor and Y finds entries to fulfill the finger table for the new node. The nodes whose finger tables should contain X update their tables. The join/leave operation of a node requires using  $O(\log n)$  messages. Chord uses a stabilization procedure running periodically on each node to update the successor and finger table entries.

## Example 8.7

### The CAN System: A Multidimensional Mesh Architecture

CAN (Content Addressable Network) organizes nodes in a virtual  $d$ -dimension Cartesian coordinate space on a  $d$ -torus. The coordinate space is dynamically split into  $n$  disjoint zones and each node owns one, where  $n$  is the number of nodes. An example of a 2D coordinate space which is partitioned into five nodes is given in Figure 8.15. Nodes store their neighbors' information (i.e., IP addresses and the zone information that they own) as their route tables. Two nodes are neighbors if their zones span overlap along  $(d-1)$  dimensions and abut along one dimension.



**FIGURE 8.15** A CAN network constructed by the repeated partitioning of the 2D coordinated space and its routing process. Courtesy of S. Ratnasamy, P. Francis, and R. Karp [40]

In Figure 8.15(a), node B and node C are neighbors of node A, but node D is not A's neighbor. Keys are mapped to coordinate space using uniform hash functions. Each key is associated with a  $d$ -dimension vector, which represents its position in the coordinate space. If a key's position in the coordinate space falls in the zone owned by a node  $i$ , node  $i$  stores the pointer to the object related to the key or directly stores the object. In a  $d$ -dimension coordinate space, each node maintains  $2d$  neighbors on average and the route complexity is of the order of

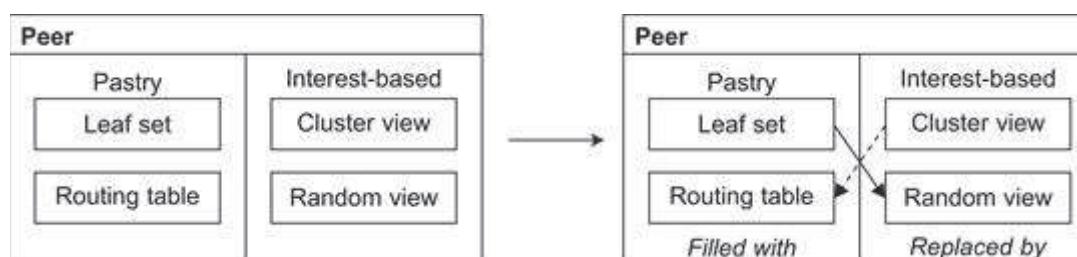
A new coming node  $X$  contacts a bootstrap node, which is located using the DNS service. The bootstrap node supplies the address information of some randomly chosen peers populated in the overlay. The new node randomly chooses a point  $P$  in the coordinate space as its position and asks a populated node to route the join message to the node  $Y$  whose zone contains  $P$ . The node  $Y$  splits its zone into two parts with equal quota and hands over one

subzone to the incoming node.

The CAN overlay uses a greedy routing algorithm. A node greedily forwards the message to its neighbor with coordinates closest to the destination. [Figure 8.15\(b\)](#) shows a 2D CAN overlay and the routing path from node C to node H. In a CAN overlay with  $d$ -dimension coordinates, on the average, each dimension has  $n$  nodes. On a torus, there are  $1/4$   $n$  nodes between any two nodes on the average. Since a message to its destination traverses all  $d$  dimensions, this yields the routing complexity, which is

## 8.2.4 Hierarchically Structured Overlay Networks

A hybrid P2P overlay network holds the features of unstructured and structured overlay network at the same time. There are always two types of approaches to building hybrid P2P overlay networks. The first involves adding structured overlays on top of unstructured overlays. For example, one can embed Chord on top of a Gnutella overlay. The flooding search protocol in Gnutella is suitable for locating highly replicated files (hays). However, for queries of rarely replicated files (needles), flooding requires a long time for a response, or even gives no response. A hybrid P2P overlay is often built by maintaining the primary components while getting the secondary components for free. In [Figure 8.16](#), the route table of a Pastry node is selected. The global random peer information is provided by the leaf set of the Pastry node. In this way, the overhead of Pastry is related to locate nearby nodes to fill the routing tables.



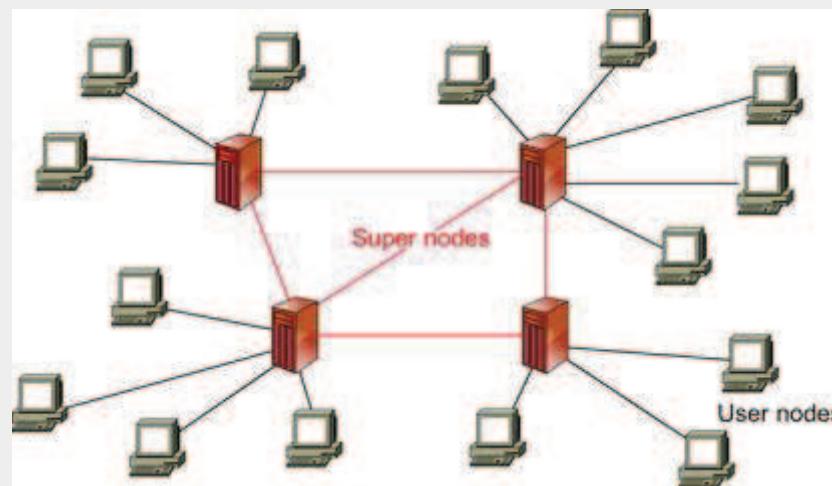
**FIGURE 8.16** The intuition of the hybrid P2P overlay construction by keeping the primary components and getting the secondary components for free.

### Example 8.8

#### KaZaA Network Using Super Nodes to Form a Backbone Overlay

KaZaA connects peers in a two-layer hierarchical structure (see [Figure 8.17](#)).

The upper layer consists of *super nodes* that are more powerful in terms of CPU, bandwidth, and stability. Lightweight user nodes are assumed at the second layer, and are connected to a parent super node. A super node acts as a hub for user nodes. A user node uploads to its parent super node the filename, content hash, and file description of each requested file. Thus, each super node maintains the file indices for all of its children nodes. The super node provides each child with a super-node list, which contains up to 200 super nodes in the system.



**FIGURE 8.17** The KaZaA architecture built with super nodes forming a backbone overlay.

In 2006, more than 3 million peers of KaZaA shared more than 3,000 terabytes of digital content, including MP3s, videos, and games. At one time, more than 50 percent of Internet traffic was generated by KaZaA users. On average, it takes 56 minutes to connect to the end super node. Each super node has about 100 to 150 children. In total, the system has roughly 30,000 super nodes. Each super node has TCP connections to 30–50 other super nodes. The system is not fully connected. Among the super nodes there is 0.1 percent connectivity with an average of 23 minutes to connect two super nodes.

Each super node acts like a mini-Napster hub by tracking the content and IP addresses of its descendants. The super node is represented with its IP address, port number, workload, timestamp, and similar data. This list is cached at the user node for use in bootstrapping and failure recovery. KaZaA encrypts all its signaling traffic. A query from a user node is first submitted to its parent super node. If any files match the query, the super node directly answers the query. Otherwise, the super node forwards the query to other super nodes using TTL-limited flooding. KaZaA uses the UUHash to identify files uniquely. Nodes request files from multiple peers using this unique identifier in order to enable parallel downloading.

The UUHash hashes the first 300 KB of a file using MD5, yielding a 128-bit

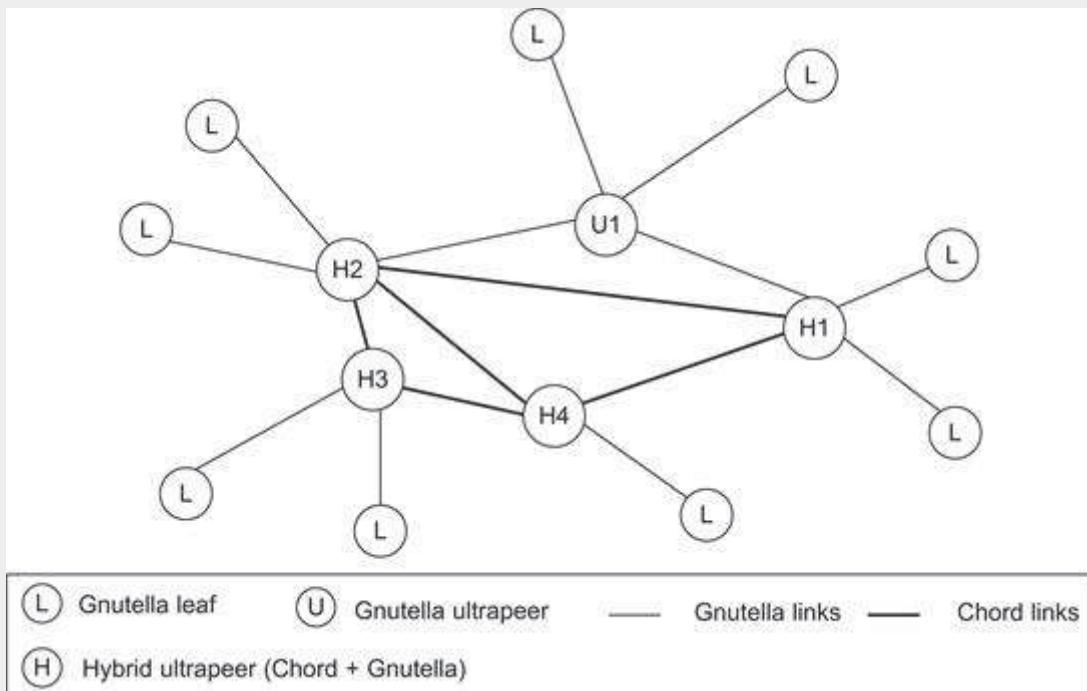
hash value. It applies a smallhash function to 300 KB blocks at file offsets  $2^m$  MB, with  $m$  being an integer from 0 until the offset reaches the end. The smallhash generates a 32-bit hash value. The MD5 hash and this smallhash are concatenated, yielding a 160-bit hash as the file identifier. The UUHash is simple since it only hashes a fraction of the file. This weakness allows users to easily modify large parts of the file without changing the UUHash identifier. However, copyright infringement is still a serious problem. Both DDoS attacks on super nodes and content pollution have been encountered in the past.

The system automatically switches to a new download server when the current server fails to deliver. The system also gives the estimated download time to users. Super nodes are relatively stable; thus node churns have a limited effect. The bandwidth on a super node is low, even in relaying VoIP traffic. The two-layer structure takes heterogeneity in capacity into account and enables network proximity awareness by assigning user nodes to nearby super nodes. Flooding is performed on top of the upper layer, thus improving scalability. However, it still does not provide real guarantees on search scope and search time.

### Example 8.9

#### Embedding Chord on Top of a Gnutella Network

Figure 8.18 shows the hybrid architecture that results after embedding Chord on top of Gnutella. For any query, the DHT-based Chord responds almost within the same time as long as there is at least one file to satisfy the query. Thus, Chord performs well for needle queries.



**FIGURE 8.18** A hybrid P2P architecture of Gnutella and Chord.

For queries to hays, it takes longer to respond. In Gnutella version 0.6, a portion of the powerful nodes (ultrapeers) are selected as hubs for ordinary nodes (leaf peers). Queries for hays are forwarded using flooding, in order to be responded to, while queries for needles follow the Chord search protocol. Hence, queries to hays and needles can respond faster.

### 8.2.4.1 Overlay Complexity Analysis

Table 8.4 summarizes the functional features and complexity characteristics of five P2P networks. All five are self-organized and cannot support node heterogeneity. Data locality is not preserved in all five overlay structures. With uniform routing, both Chord and CAN destroy data locality easily. Network proximity is supported only in Chord, CAN, and SkipNet. They all (except Gnutella) guarantee determinate search results. Gnutella applies a random graph. Chord applies a chordal identifier space using fingers to identify immediate successors.

**Table 8.4**

Complexity Properties of Six P2P Overlay Networks

Features	Gnutella [60]	Chord [45]	CAN [40]	TerraDir	SkipNet
Overlay structure	Random	Circular identifier space r: Immediate successors	Multi-dim. Cartesian space d: dimensionality	Tree data hierarchy	Distributed skip list
Design parameters	N/A	O( $\log N$ )	$O(d \cdot N^{1/d})$	N/A	N/A
Routing hops	O(1)	O( $\log N$ )	O(2 d)	O(h), tree height	O( $\log N$ )
Node state	O(1)	O( $\log N$ )	O(2 d)	O(1)	O(2 $\log N$ )
Node join	O(1)	O( $\log N$ )	O(2 d)	O(1)	O( $\log N$ )
Load balancing and reducing query hotspots	N/A	Via consistent hashing and virtual nodes	Partitioned space, zone reassignment, caching and replication	Via caching and replication	Consistent hashing the suffix of object name
Fault handling	Flooding to other valid neighbors	Successors list, application layer replication	Multiple realities, overloading zones	Node replication, routing to next best prefix node	Redundant neighbors at level 0, survive failures at boundary
Network proximity	No	Proximity routing to finger's successor with lowest latency	Topology-aware overlay and proximity routing	No	Proximity neighbor selection via P-Table and C-Table
Support node heterogeneity	No	No, but might through virtual nodes	No, but might split zones proportional to node capabilities	No	No
Self-organization	Yes	Yes	Yes	Yes	Yes
Guarantee determinate searching results	No, only within broadcast scope	Yes	Yes	Yes	Yes
Data locality	No	No, destroyed by uniform hashing	No, destroyed by mapping to hyper space via uniform hashing	Yes	Yes
Applications	N/A	CFS [27]	Data-centric sensor networks	Resource discovery	Global event notification service

Courtesy of Min Cai, EE 657 Report, University of Southern California, 2006.

CAN applies a multidimensional Cartesian space. TerraDir has a tree data hierarchy and SkipNet uses a distributed skip list. The major attraction of Gnutella is its constant node state and node joining time, but its routing time is the longest,  $O(n)$ . Flooding in Gnutella is an expensive way to search, but it supports fault tolerance. The rest all support fault tolerance using replication techniques.  $O(\log n)$  routing speed is achieved in Chord and SkipNet. The routing time in TerraDir is proportional to the tree height ( $h$ ) applied. The dimension  $d$  of CAN affects its performance greatly.

## 8.3 Routing, Proximity, and Fault Tolerance

In this section, two fundamental techniques related to P2P systems will be discussed, namely routing and locality awareness. In P2P systems, peer nodes are organized in decentralized and self-organizing overlay networks. The routing algorithms, which indicate how to get to a specific node from any node, should be distributed and only rely on nodes in the local view of the whole system. Locality awareness, also called proximity awareness, is a mechanism to let peer nodes be prone to connect with physically close peers, in order to reduce average overlay link latency and backbone network bandwidth consumption. The P2P overlay network is ad hoc. Thus, these

mechanisms are required to tolerate and recover peer failures and disconnections.

### 8.3.1 Routing in P2P Overlay Networks

Unstructured P2P overlay networks give no indication of how to locate a specified node since node neighborhoods are randomly selected and are not subject to constraints. The routing algorithms in unstructured P2P overlay networks are always based on blind flooding. When receiving a message, a peer node simply forwards it to all its neighbors except the one from which it receives the message. Suppose the number of nodes is  $n$  and the average node degree (i.e., the number of neighbors) is  $k$ . The number of messages used for locating a node is      on average.

The number of messages used for locating a node is      on average. Since a message follows the shortest path from the source to the destination, the route complexity, which is defined as the expected number of overlay hops for a message to reach a specified node from any node, is directly determined by the overlay graph diameter. Overlay networks based on small-world graphs have a short diameter, allowing for fast routing. Freenet [10] is such an overlay network.

#### Example 8.10

#### Freenet—An Unstructured P2P Network

Freenet is a decentralized storage service, which provides good anonymity and censorship resilience. Its architecture belongs to unstructured overlay networks since it allows any node to be a neighbor to any other node. Each node contributes a storage space to the system. Nodes and files are mapped to the same ID space using hash functions. A file is encrypted and stored at a node with the ID closest to the file's ID. Files may be split into multiple chunks, with additional chunks added to provide redundancy. Each chunk is handled independently over distributed nodes.

Freenet uses a key-based route protocol to find requested files. Both publishers and readers of files are hidden from the outside. Data files that are stored on a node form a datastore stack, which is illustrated as an example in Figure 8.19. The stack is divided into two parts. The top part stores the keys, node references of the next hop, and data, while the bottom part only stores keys and node references. When receiving a new file insertion request, a node puts the information at the top of the stack. If the stack is full, LRU (least recent used) is used to swap out old entries. Unlike Gnutella, in Freenet, even if the file publishers leave the network, files are still available.

Key	Next_hop	Data
100	000	00000000000000000000000000000000

123	2	0xabcde
234	3	0xc4z
564	7	
789	8	

**FIGURE 8.19** An example of a datastore stack of a node in the Freenet.

The routing in Freenet is similar as the routing used in IP. The datastore stacks are taken as routing tables. For a request of a file with key  $k$ , if the key can be found in its routing table, the node stops forwarding the request and replies to the upstream requester. Otherwise, the node forwards the request to the node indicated by the next hop reference of the entry with the closed key to  $k$ . The horizon of a request is limited by the TTL. If data is not found when the TTL expires, a failure is reported. If data is found, it is cached on each node along the request path.

The same process is used for insertion of data: The data to be inserted is cached on each node along the path to the target node. Thus, a request to a data file will cause the file to be replicated more. The request originator may connect the node that answered the request, while discarding old nodes that are least recently used. This is called path folding, and would cause a clustered structure. Path folding makes the simple routing protocol used in Freenet well performed. Anonymity is achieved by randomly modifying the source of messages as they traverse the network.

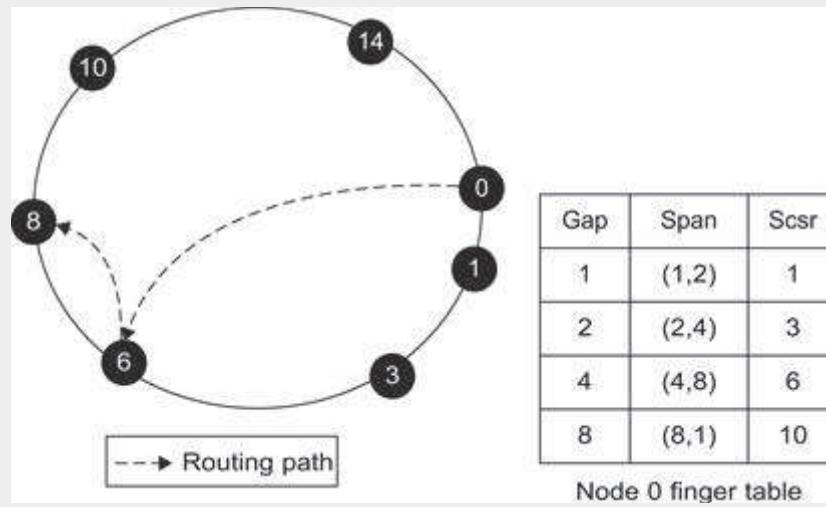
DHT-based structured P2P overlay networks have strict and predefined structures, which facilitate message routing. Each node is associated with a globally unique identifier. The routing process to a destination node gradually reduces the identifier's distance from the message processing node to the destination node. Although different structured overlays have different routing algorithms, the routing complexities are where  $n$  is the number of nodes. We describe the routing algorithms for three representative structured P2P overlays in the following examples.

### Example 8.11

#### Table-Lookup Search in a DHT-Based Chord Network

Chord overlay networks resort to finger tables to efficiently route messages. When receiving a message whose destination is  $y$ , a node  $x$  forwards it to the node in its finger table which is the closest one to  $y$  in the identifier space. To this end, node  $x$  checks its finger table in a reverse order. The first node

whose identifier is smaller than  $y$  is chosen as the next step node. For example, in Figure 8.20, seven nodes are populated in a 16-node chord structure. The minimum routing path from node 0 to node 8 is via node 6.



**FIGURE 8.20** An example of the table lookup search in a Chord overlay network.

The routing complexity for the chord is  $\log n$  for an  $n$ -node system. Suppose that node  $x$  selects the  $i$ -th entry in its finger table as the next step node  $z$ . Then node  $z$  is at least  $2^i$  away from  $x$  in the identifier space. Since the identifier of the  $(i+1)$ -th entry of  $x$ 's finger table is not smaller than  $y$ , node  $y$  is at most  $2^{i+1}$  away from  $x$  in the identifier space. Therefore, the distance between node  $x$  and  $z$  is at least equal to the distance between nodes  $z$  and  $y$ , which means node  $z$  halves the distance between nodes  $x$  and  $y$ . After  $m$  steps, the distance between the source node to the destination is equal to  $2^{-m}$ , where  $m$  is the identifier length. Since the populated nodes are randomly distributed, an average of one node exists in each of the range

### 8.3.2 Network Proximity in P2P Overlays

P2P overlay networks are logical structures over IP networks. While randomized overlay graphs are fault tolerant and have a low diameter, they ignore the network proximity information on the IP networks. This causes two physically close peers to be far from each other in the overlay network, while two nodes that are close in the overlay network are far from each other in the physical network. This scenario is denoted as topology mismatching. The problem also exists in structured P2P overlay networks. For the CAN network, (Example 8.6, also Figure 8.15a), if nodes C and E belong to the same autonomous system, while A and G belong to another, the messages from C to G traverse the links between the two domains three times, which inevitably prolongs the delay and consumes unnecessary network bandwidth.

As to the structured P2P overlay networks, node neighborhoods are so rigid that it is hard to optimize them in terms of network proximity awareness. There are three types of approaches to exploit network proximity in structured P2P overlays: *geographic layout*, *proximity routing*, and *proximity neighbor selection*.

### 8.3.2.1 Geographic Layout

Node identifiers are no longer random. Physically close nodes are assigned adjacent identifiers, and therefore are close in the overlay network. Since many structured P2P overlay networks (e.g., Chord, Pastry, Tapestry) rely on the randomness of node identifiers to guarantee robustness and performance, this approach is only suitable for some special overlays, such as CAN. For example, the landmark binning scheme can be leveraged to build proximity-aware CAN overlay networks. In this scenario,  $m$  publicly accessible nodes are chosen as landmark nodes. Each node measures the distance to these nodes and sorts them in terms of increasing distances, which yields an order of landmark nodes. The order represents the “bin” the node belongs to.

Physically close nodes are likely to belong to the same or similar bins. The coordinate space of a CAN overlay network is divided into  $m!$  zones, one zone per bin. A new node randomly selects a point from the zone which associates with its bin and joins. This achieves the goal that physically close nodes are close in overlay networks. However, nodes are no longer randomly distributed on the coordinate space. It is possible that a small portion of the nodes occupy most of the identifier space. These nodes are vulnerable to overloading.

### 8.3.2.2 Proximity Routing

There are always multiple paths between any two nodes in P2P overlay networks. For the CAN in [Example 8.7 \(Figure 8.15b\)](#), the node C can follow either path C-A-B or path C-F-B to get to node B. The idea behind proximity routing is to select the path with minimum delay. However, in distributed overlay networks, finding this path is as complex as a Traveling Salesman Problem, which is NP-hard. A heuristic way is that each node selects the closest neighbor as the target node to forward messages. However, this may increase routing path length. For example, in a Chord overlay network, if each node selects its successor as the target node, the routing complexity is  $O(n)$ . Therefore, the candidate target nodes are limited to a small number. The best example overlay for proximity routing is the Pastry network. We introduce the Pastry architecture in the following example.

#### Example 8.12

#### Pastry – A Proximity-Routing Overlay with Nested Groups

The Pastry is a structured overlay, which has an identifier space coded with high-radix digits. The purpose is to subdivide the peer nodes into nested groups with layers of subgroups. Those nodes in the same inner-most subgroup are very close neighbors, known as *proximity subgroups*. The overlay uses high radix identifiers to search the nodes in the nested groups or subgroups. For example, a 128-bit identifier space can specify a very large P2P network with identifier viewed as base-16 numbers (base  $2^b$  for  $b = 4$ ) like 65A1FC04. The identifiers are divided into groups with 16 subgroups per each group.

Each data key is stored in a node with the closest identifier. Node addressing defines the subgroups. Nodes in the same subgroup know each other's address. Thus, the search time can be significantly reduced for being in the same proximity group. Furthermore, each node knows the IP address of one delegate node in some of the other groups. Each node needs to know the IP addresses of all up nodes in its inner group. The proximity routing is applied to perform fast search in  $\log_2 n$  steps, where  $2^b = 16$ . For  $n = 2^{16}$  nodes or 64 K nodes, it takes at most 4 routing steps ( $\log_{16} 32K$ ) to route the messages among 16 groups, compared with  $\log_2 64K = 6$  steps needed in using Chord.

With node failures, the Pastry is definitely faster than Chord and CAN. During failure recovery, the Pastry may take  $O(n)$  routing hops at the worst case. Each routing table entry points to a node close to the local node in the proximity space, among all node identifiers with the appropriate prefix. In a way, Pastry has the best locality properties. The expected distance traveled by a message is within a small constant in the proximity space. Pastry is suitable to use in building very large P2P networks. The fast lookup search within proximity space makes Pastry very suitable to use in very large-scale P2P networks. Details of Pastry can be found in [21].

### 8.3.2.3 Proximity Neighbor Selection

Nodes choose physically close nodes as their neighbors in overlay networks. In structured P2P networks, the identifiers of a node's neighbors should be subject to some constraints. For example, in Pastry networks, the nodes in the  $i$ -th row of a node's routing table should share  $i$ -digit long prefix with the node identifiers. This approach can also be applied in Chord overlay networks. Instead of fixing the  $i$ -th entry of node  $x$ 's finger table to the node whose identifier is  $(x + 2^i) \bmod 2^m$  ( $m$  is the length of identifiers), node  $x$  selects the physically closest node whose identifier falls in the range

This change does not affect routing complexity. Landmark clusters with the dimension reduction algorithm is a wildly used scheme. Each node measures the delay to  $m$  landmark nodes which are publicly accessible. The measurement yields an  $m$ -dimension vector which acts as the node position in an  $m$ -dimension landmark space. The idea

behind this is that physically close nodes are also close in the landmark space. Then, space filling curves are used to map the  $m$ -dimension vectors to a space with lower dimensions. For Chord, the vectors are mapped to one dimension number. The space-filling curves preserve the proximity information. The mapped results are stored on the overlays based on their specific protocol, which facilitates archival of close nodes. The landmark cluster scheme is coarse-grained and cannot distinguish two physically close nodes. Thus, RTT measurement can be further combined.

Compared with structured overlay networks, unstructured networks are easier to optimize in terms of network proximity, because the node neighborhood relationships are more flexible in unstructured overlay networks. The optimization operation is a process of deleting high-cost overlay connections and adding low-cost ones. The key problem is how to identify high/low-cost overlay connections and cut/establish them in a decentralized way. A simple approach is that each node actively measures link costs (i.e., latencies) to its neighbors. The high-cost overlay links are disconnected if they are redundant. Nodes probe one another to locate closer nodes and build the overlay connections.

### 8.3.3 Fault Tolerance and Failure Recovery

This section covers fault tolerance and failure recovery techniques developed for upholding P2P operations under faulty and failure conditions.

#### 8.3.3.1 Faults and Node Failures

Node failures also have a vital impact on P2P overlay connectivity. The links to a node would be disconnected if the node failed. The extent of the impact on overlay connectivity caused by node failures depends on the graph property of the overlay and the degree (i.e., number of neighbor links) of the faulty node. For instance, in a power-law graph-based P2P overlay network like Gnutella 0.4, failures of a subset of randomly selecting nodes would not fragment the overlay into disconnected portions. However, failures of some high-degree nodes would easily shatter the overlay into pieces of disconnected components.

DHT-based structured P2P systems implement lookup time by building neighbor links for every node. The overlay neighbors of a node should be selected according to the predefined rules, rather than selected randomly. For example, in Chord, the  $i$ -th entry in a node  $x$ 's finger table should be the successor of  $x$  where

and  $m$  is the length of the ID. Structured P2P systems rely on these neighbors to accelerate the lookup service, rather than guarantee overlay connectivity. Thus, a neighbor failure would delay resource lookups. Overlay connectivity is guaranteed in other ways. In Chord, each node maintains immediate successors to achieve this goal.

The aforementioned node faults are simply considered crash failures or silent failures, which means the fault nodes just do not respond to the messages sent to them (i.e., they stay silent). Another type of fault is the Byzantine failure, which means the fault nodes

behave inconsistently when communicating with others. Byzantine failures are often triggered by attacks and discussed in the context of DHT-based P2P systems. DHT-based P2P systems are more vulnerable in the case of Byzantine failures. An adversary can elaborately select the IP addresses for Byzantine peers and where the peers join the network in order to put them in critical positions. When processing lookup requests, the Byzantine peers provide inconsistent answers for the same request, which delays the lookup service or even makes the system useless. On the other hand, unstructured P2P systems flood the lookup requests. Thus, there are multiple disjointed paths from the query source to the nodes which satisfy the requests.

### **Example 8.13**

#### **Failure Effects on Real-Life P2P Networks**

The impact of node failures on a P2P network depends on the system architecture and the protocols applied. In a central index server model such as Napster, the failures of the index servers would pull down the entire system, since the resource lookup service would be completely down, while in a pure P2P model such as Gnutella 0.4, failures of individual nodes would have very little impact on lookup services because all nodes would keep the indexes by themselves.

In a super-node hybrid model like KaZaA, failure of the super nodes would cause index information loss for those peers that connect with it. This implies that the resource lookup service will be downgraded or shut down. In a DHT structured P2P network such as Chord, the resource index information is distributed almost evenly among all the nodes. Thus, the failures of the nodes inevitably deteriorate the performance of the lookup service, but do not necessarily pull down the whole system. The resource sharing service between two nodes would be interrupted if either node fails. P2P networks resort to service backup and resource relocation to resume file-sharing services.

#### **8.3.3.2 Analysis of Failure Recovery**

Since failures occur frequently, P2P systems demand safe recovery from node failures. Chord leverages stabilization operations to fix node failures. Another method is to let a node periodically check the aliveness of a random neighbor in its finger table. In the case of an  $i$ -th entry failure in a chordal ring, node  $x$  relocates the successor to  $\dots$  where

Nodes also periodically verify their immediate successor. In Pastry, a failure of a routing table entry is detected when forwarding lookup requests. As to the method for new node selection from backup node lists, the *random replacement* strategy performs

better than others in terms of churn if the distribution of node duration time is highly skewed. Here, the new node is randomly chosen from several nodes to replace the faulty one. The idea behind this is that the strategy is biased in favor of longer duration nodes.

### 8.3.3.3 Fault Tolerance Techniques

Unlike traditional client/server model-based distributed systems, no node in P2P systems has a global view. Nodes rely on themselves to detect faults and recover from failures in a fully decentralized fashion. P2P overlay networks ensure stable throughput by providing redundancy. For example, each node has multiple neighbors for service providers. The question is how many redundancies are adequate. In Chord overlay networks, the successor lists are the key component for connectivity and routing correctness.

Each node maintains a sequential successor list with length  $n$  where  $n$  is the number of nodes. If the current successor fails, a node replaces it with the first live entry in its successor list. Suppose that node fails independently with probability  $1/2$ . The probability that all the nodes in a node's successor list fail is  $(1/2)^n$ . Thus, with high probability every node is aware of its immediate live successor, which is sufficient to guarantee overlay connectivity.

In unstructured P2P overlay networks, node neighborhoods are random and flexible. Nodes are not required to maintain restrictive structures. Thus, they are considered more reliable than structured P2P overlays. The reliability of structured P2P networks means predefined structures are maintained, while reliability demands at least one path between any pair of nodes.

In an overlay network based on a random graph with  $n$  nodes, if the probability  $p$  with which there is a link between each pair of nodes is  $\frac{1}{n}$  the overlay network is connected with probability  $\frac{1}{e}$ . This result can be easily extended to a scenario where there are node and link failures. Suppose link failures are independent of each other with probability  $q$ . In this case, to achieve the connectivity probability  $\frac{1}{e}$  the probability is

### 8.3.3.4 Failure Analysis

If any node fails with a probability  $\frac{1}{n}$  independently, the base ring is connected with a high probability that is more than  $\frac{1}{e}$  where  $e$  is the base of the natural logarithm. The results confirm that REM overlay is as resilient as the one in which each node maintains  $\sqrt{n}$  random neighbors in the presence of random failures. In the preceding analysis, the failure resilience of overlay networks is evaluated under random failures. However, in a proximity-aware overlay, the failures of neighbor nodes may be correlated. Physically close nodes may fail concurrently due to network congestion. Further work should evaluate the fault tolerance for proximity-aware overlays.

## 8.3.4 Churn Resilience against Failures

A P2P network often faces disturbance from high node chucks. The chucks come from unexpected node joins, departures, or failures. Node failures or sudden departures

impose a great adverse effect on network performance. This is caused by unavailability of data stored on the faulty nodes. The nodes requesting service from the faulty nodes need to relocate the services elsewhere. Hence, a P2P overlay network should be designed to be fault tolerant and churn resilient. This section is devoted to overcoming the difficulties associated with churn and failure conditions that are frequently encountered in a P2P network. A *churn-resilient protocol* (CRP) is presented to alleviate churn effects.

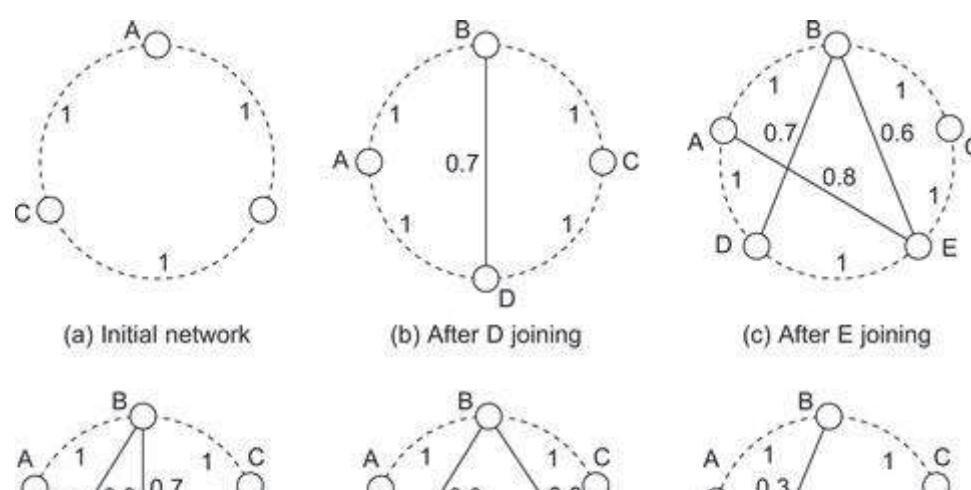
### 8.3.4.1 Churn-Resilient Protocol (CRP)

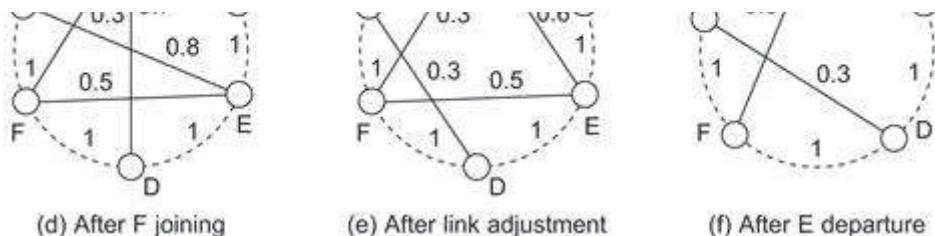
A solution to increase fault tolerance in proximity-aware overlays is to guarantee connectivity of critical links, such as random links among clusters or nodes. This can be achieved by providing backup nodes to build the critical links. For example, in GoCast [47], each node maintains a node cache as backups. Another example is CRP [24] for construction of proximity-aware and churn-resilient P2P overlay networks.

The topology of the CRP-enabled overlay design looks like a ring augmented with extra chord links. However, the overlay topology differs from Chord in that the extra links are not fixed fingers from nodes. Instead, the chord links are established among nodes satisfying certain proximity properties. Both network proximity and capacity proximity are applied in CRP. Network proximity is measured by the latency or closeness of two nodes in physical IP networks. This proximity enables construction of low-latency chord links. Capacity proximity is measured by the closeness of nodes with respect to node capacities. Capacity proximity allows the grouping of nodes with similar capacities into clusters.

### 8.3.4.2 Overlay Construction Using CRP

The initial base ring with three nodes is shown in Figure 8.21(a). The ring is assumed to be unidirectional with clockwise links shown by dotted edges. We assume that all ring links are weighted with one unit. Node A's successor and predecessor are B and C, respectively. The fourth node D is inserted into the interval between A and C in Figure 8.21(b). A chord link weighted with 0.7 is added between nodes B and D. In Figure 8.21(c), the fifth node E is inserted between nodes C and D. Two new chord links are added, with weights of 0.6 and 0.8.





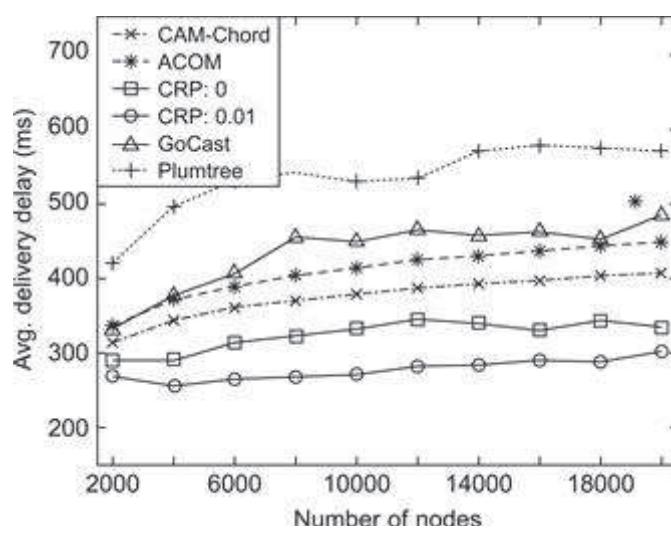
**FIGURE 8.21** An example CRP-enabled overlay network design in six steps. Courtesy of Li, Xie, Hwang, and Li [24]

A newly inserted node may establish new chord links with other nodes by removing some existing chord links. An existing chord link is removed when it has higher weight than the added new links. In Figure 8.21(d), a new node F is inserted between nodes A and D. Two new chord links are added to connect F with nodes B and E replacing the old link between B and E in Figure 8.21(c). The newly added chord links must have lower weights than those of the links being replaced. The overlay is dynamically adapted by replacing higher-weight chord links with lower ones, without changing any node's degrees.

In Figure 8.21(e), the links between B and D, A and E in Figure 8.21(d) are replaced with two new lower-weight links between A and D, B and E. Figure 8.21(f) shows the overlay after the departure of node E in Figure 8.21(d). The links associated with E are removed. The base ring is always connected as nodes keep their correct successors. To improve the connectivity, the CRP borrows the idea of the successor list from Chord. Each node maintains a list of successor nodes down the base ring. If a node detects a faulty successor, it replaces that successor with the first live node in the list. In Figure 8.21(e), for example, node C keeps two successor nodes E and D. If node E fails or leaves, C takes D as its new successor.

### 8.3.4.3 Performance of CRP Scheme

Figure 8.22 depicts the relative performances of CRP and four other tree-based algorithms. The notation CRP: $q$  denotes a CRP system with a message replication ratio  $q$ . CAM-Chord [56] builds on Chord overlays. It takes the heterogeneity of node capacities into account. ACOM [9] implements data dissemination by combining scope-limited flooding and random walks on overlay networks. GoCast [47] and Plumtree [23] apply a tree-based gossip scheme for data dissemination. It can be found that CRP outperforms all other designs in terms of delivery time due to proximity awareness. The message replication in CRP is close to 0 and can be controlled by adjusting  $q$ .



**FIGURE 8.22** Average delivery time of five data dissemination schemes over a P2P network. Courtesy of Z. Li, et al. [24]

The CRP-enabled overlay networks consider both network proximity and capacity proximity. It uses a hybridization of scope flooding and tree traversing to disseminate data promptly. The overlay networks provide redundant proximity links for tree structures for fault resilience. The churn-resilient CRP-enabled overlay networks provide adequate backup links for the tree structure. Once a node  $x$  receives a departure notification from its parent or detects the failure of its parent by heartbeat messages, it actively switches its tree parent. The node  $x$  prefers its proximity chord neighbor as the new parent to save on delivery time.

## 8.4 Trust, Reputation, and Security Management

The anonymous and dynamic nature of peer activities makes P2P networks very vulnerable to abuses by selfish and malicious peers. For example, most P2P file-sharing networks (e.g., Gnutella) consist of autonomous peers with special self-interests. There is no efficient way to prevent malicious peers from joining open networks. To encourage resource sharing among peers and combat malicious peer behavior, trust and reputation management becomes critical to maintain the normal operation of P2P networks.

Without trust, peers will have little incentive to contribute their resources to other peers. The peers may hesitate to interact with unknown peers due to the concern of receiving corrupted or poisoned files or being exploited by malware. Identifying trustworthy peers demands reputation systems in commercial P2P applications, such as for online shopping, auctioning, content delivery, pay-per-transaction, etc.

### 8.4.1 Peer Trust and Reputation Systems

This section characterizes the trust model needed to establish harmony among the peers. We will define trust matrix for aggregating peer reputations in P2P reputation system development.

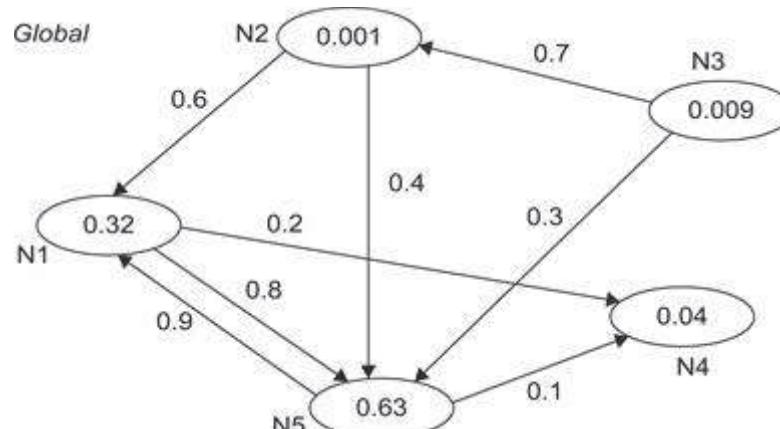
#### 8.4.1.1 Peer Trust Characteristics

There are two ways to model the trust or distrust among peers: namely *trust* and *reputation*. Trust refers to the belief of one peer on another, based on his direct experiences with the peer. Reputation is a collective opinion on a peer by other peers based on recommendations. To face the reality of an open P2P network, one has to assume that the participating peers in a P2P system do not trust one another unless proven otherwise.

A fair *reputation system* is thus needed to establish the trust or distrust among peers, based on recorded historical behaviors of the peers. The purpose is to distinguish good peers from bad ones through a scientific screening process. The quality of the reputation system is primarily indicated by its accuracy and effectiveness in updating periodically.

### 8.4.1.2 Trust Matrix for Computing Reputations

Consider an overlay network for global reputation aggregation in a P2P system over five peers, denoted by nodes N1, N2, ..., N5 of a directed graph shown in [Figure 8.23](#). The trust relationships among the peers are represented by a square *trust matrix*  $M(t) = [m_{ij}(t)]$ , where  $m_{ij}(t)$  is the *local score* issued by node  $i$  in evaluating node  $j$  at time  $t$ . All trust scores are fractions in the range  $(0, 1)$  with 0 meaning no trust (or no contact) and 1 for 100 percent trust. Therefore, trust is relatively measured by these fractions. For the five-node network, the following trust matrix at certain time  $t$  is achieved. Note that all row sums are 1. A zero entry,  $m_{ij}(t) = 0$ , means node  $i$  does not evaluate node  $j$  for lack of direct contact. The diagonal scores are all zero, meaning no peer evaluates himself.



**FIGURE 8.23** A directed graph showing the trust relationship among five peers in a P2P network.

(8.2)

The edge labels are the *local scores* issued between all (source, destination) pairs. The fraction value inside each node in [Figure 8.23](#) is the *global reputation score* of that peer at

time  $t$ . This global score is resulted from aggregating all local scores issued by all peers toward the peer being evaluated. However, all local scores must be weighted by their own global reputations. In another words, the global reputation is the weighted sum of all local scores. For clarity, all the global reputation scores are normalized so that their sum is always a “1”. For example, the global scores of the five peers are represented by a *reputation vector* of five components which add up to 1.

(8.3)

### 8.4.1.3 Reputation Systems

A *reputation system* calculates the global reputation score of a peer by considering the opinions (i.e., *feedback*) from all other peers that have interacted with this peer. After a peer completes a transaction(e.g., downloading a music file), the peer will provide his feedback for other peers to use in future transactions. By making the reputation scores publicly available, peers are able to make informed decisions about which peers to trust.

The eBay reputation system is a simple and successful one, since it has a centralized authority to manage all user feedback scores. However, in an open and decentralized P2P system, peers will not have any centralized authority to maintain and distribute reputation information. Instead, most existing P2P reputation systems calculate global reputation scores by aggregating peer feedback in a fully distributed manner. Building an efficient P2P reputation system is a challenging task due to several intrinsic requirements of large-scale P2P systems.

To measure peer reputations, one can set up an evaluation system. After each transaction, the two participant parties evaluate each other by giving an honest score. This is like what we do in online auction systems such as eBay. Each peer's reputation can be retrieved from the voting system. However, not every peer is trustworthy. The scores from malicious peers are useless, while the scores from more trustworthy peers are more useful. This advises us to assign different weights to votes based on the voting peers' reputations. It should be noted that a peer's reputation may be different from other peers. The reputations can be expressed with a reputation matrix.

### 8.4.1.4 Global Reputation Aggregation

As global reputation scores are aggregated from local feedback, the distribution property of feedback plays a significant role in the design of an efficient reputation system. Surprisingly, most previous work either ignored the distribution of peer feedback or assumed an arbitrary random distribution, which could be misleading. To achieve global reputation scores, each peer calculates its own parts and all peers cooperatively compute a global matrix. For example, at time  $t + 1$ , the global reputation score of node N5 is computed as follows:

Similarly, the remaining global scores for the other four peers are computed to yield the following updated global reputation vector:

(8.5)

This vector is unnormalized. Dividing each peer score by the total sum of all component scores, one can obtain the following *normalized global reputation vector*. Note that the sum of the five peer reputation scores should be 1 in the normalized vector.

(8.6)

#### **8.4.1.5 Design Objectives of Reputation Systems**

The following six key issues should be addressed in the design of a cost-effective P2P reputation system:

- **High accuracy** To help distinguish reputable peers from malicious ones, the system should calculate the reputation scores as close to their real trustworthiness as possible.
- **Fast convergence speed** The reputation of a peer varies over time. The reputation aggregation should converge fast enough to reflect the true changes of peer behaviors.
- **Low overhead** The system should only consume limited computation and bandwidth resources for peer reputation monitoring and evaluation.
- **Adaptive to peer dynamics** A peer joins and leaves an open P2P system dynamically. The system should adapt to these peer dynamics instead of relying on predetermined peers.
- **Robust to malicious peers** The system should be robust to various attacks by both independent and collective malicious peers.
- **Scalability** The system should be able to scale to serve a large number of peers in terms of accuracy, convergence speed, and extra overhead per peer.

#### **8.4.1.6 Three Representative Reputation Systems**

The PeerTrust system [53] was developed at Georgia Institute of Technology. The system computes the reputation score of a given peer as the average feedback weighted by the scores of the originators. Five trust attributes were suggested in this system. The EigenTrust system was developed at Stanford University [20], which aggregates reputation information using the eigenvector of the trust matrix over the peers. EigenTrust relies on a good choice of some pretrusted peers. This assumption may be

overly optimistic in a distributed computing environment because the pretrusted peers may change with time. The PowerTrust reputation system [57] was developed at the University of Southern California. PowerTrust will be covered in Section 8.4.3. Table 8.5 compares these three systems in four technical aspects.

**Table 8.5**  
**Comparison of Three P2P Reputation Systems**

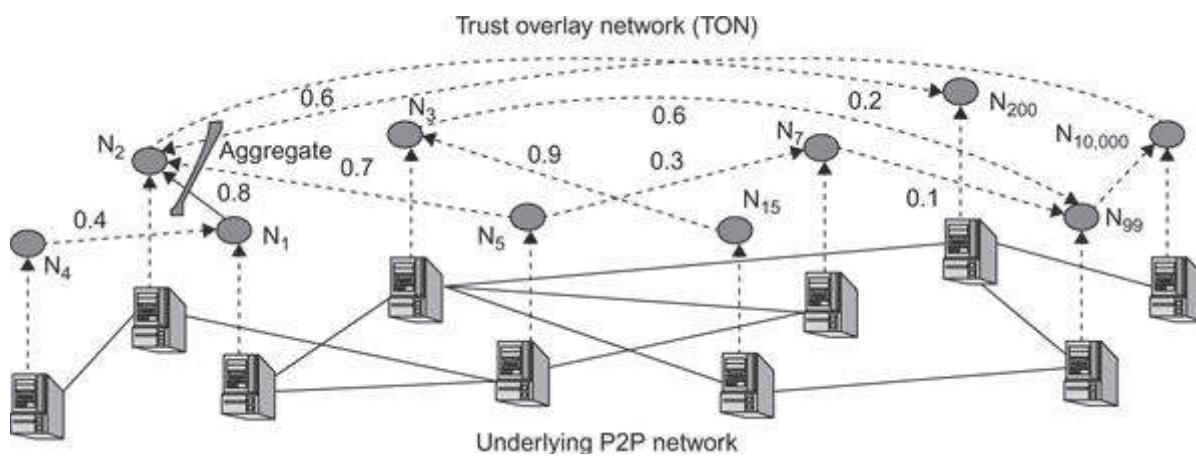
Reputation System	Local Trust Evaluation	Global Reputation Aggregation	Implementation Overhead	Scalability and Reliability
<b>EigenTrust</b> at Stanford University [20]	Uses sum of positive and negative ratings	Uses pretrust peers to compute global scores from trust matrix	Moderate overhead experienced in assigning score managers and in messaging for global score aggregation	Limited scalability and reliability if pretrust peers leave
<b>PeerTrust</b> at Georgia Tech [53]	Normalized rating on each transaction	Peer calculates trust score over five factors in a distributed manner	Moderate overhead experienced in global score calculation over five factors and on the establishment of the trust manager	Partially scalable and resistant to malicious peers
<b>PowerTrust</b> at USC [57]	Uses Bayesian method to generate local trust scores	Distributed ranking of power nodes and LWR strategy to aggregate global reputation scores	Low overhead in using locality-preserving hashing to locate power nodes. Global aggregation time drops sharply with look-ahead random walk strategy applied	Highly scalable and robust with dynamic peer join and leave and malicious peers

## 8.4.2 Trust Overlay and DHT Implementation

This section introduces a trust overlay concept for fast aggregation of trust information in global reputation calculations. Then a DHT implementation of the reputation system is described.

### 8.4.2.1 Trust Overlay Network (TON)

This is a virtual network on top of a P2P system. A TON is represented by a directed graph in Figure 8.24. The graph nodes correspond to the peers. The directed edges or links are labeled with the feedback scores between two interacting peers. The feedback score is issued by a peer (source of the link) for the service provided by the interacting peer (destination of the link). For example, node  $N_5$  after downloading music files from nodes  $N_2$  and  $N_7$  issues the feedback scores, 0.7 and 0.3, to the two provider nodes, respectively. If a node gets more than one service from the same provider, this consumer generates a newly updated score after each transaction.

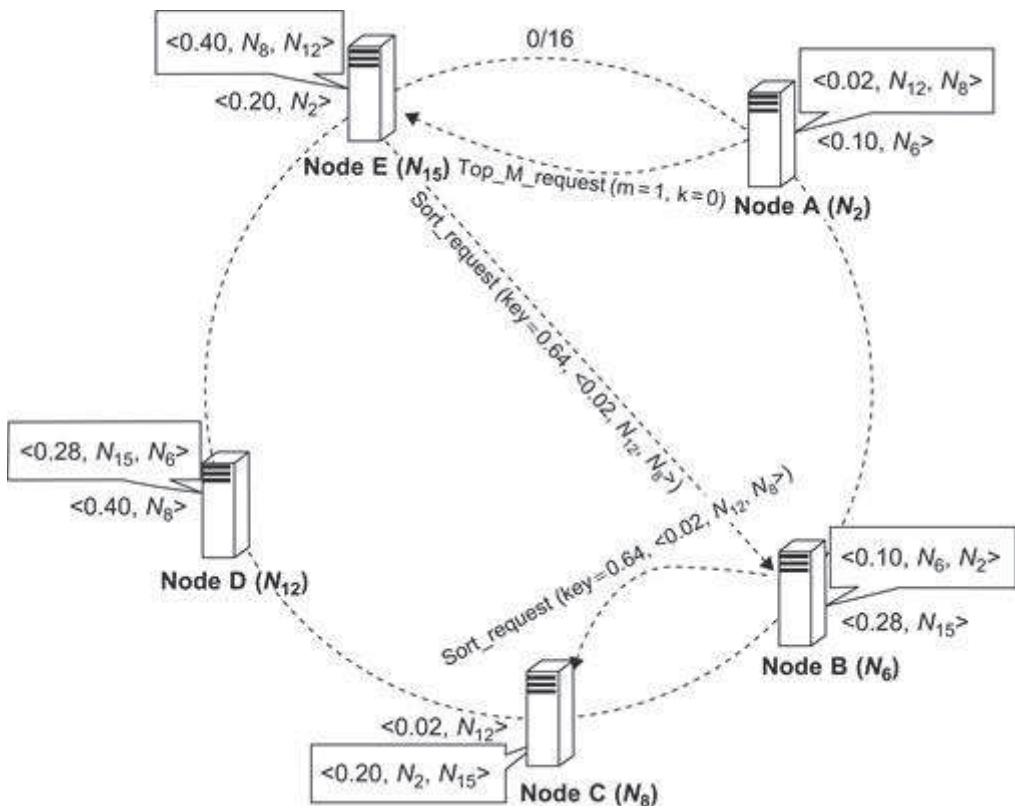


**FIGURE 8.24** A trust overlay network for a P2P trust management, where the edge is weighted by the peer feedback score for the service provided. The global reputation of a peer (node) is calculated by the weighted sum of all local trust scores received on the incoming edges. Courtesy of R. Zhou and K. Hwang [57]

Our system can incorporate different methods to generate feedback scores, such as Bayesian learning. In a TON, every node keeps feedback scores for its neighbors. Because every peer has its own criteria to generate feedback scores, the feedback will be normalized to *local trust scores*. Each node  $N_i$  is rated with a *global reputation score*  $v_i$ . This global reputation is aggregated from local trust scores weighted by the global reputations of all in-degree neighbors. For example, the global reputation score of  $N_2$  could be calculated by weighting three incoming local scores (0.8, 0.7, 0.6) from  $N_1$ ,  $N_5$ , and  $N_{10,000}$ , respectively:  $v_2 = 0.8 v_1 + 0.7 v_5 + 0.6 v_{10000}$ . Given  $v_1 = 0.04$ ,  $v_5 = 0.0007$ , and  $v_{10000} = 0.000005$ , then  $v_2 = 0.8 \times 0.04 + 0.7 \times 0.0007 + 0.6 \times 0.000005 = 0.032 + 0.00049 + 0.00003 = 0.032493$ . It is interesting to note that node  $N_1$  has a much higher reputation score  $v_1$ , compared with  $v_5$  and  $v_{10000}$  in this example. Thus, node  $N_1$  carries more weight in the global reputation aggregation process.

#### 8.4.2.2 DHT Implementation

Basically, distributed reputation ranking requires two different hash overlays. One assigns peers to their score managers and the other ranks the peers by their global reputation scores. Figure 8.25 presents a five-node system built on top of a chord with a 4-bit circular hash space. Node  $N_{15}$  is the score manager of node  $N_2$  whose global reputation is 0.2. Node  $N_{15}$  hashes the reputation value 0.2 using a simple LPH function  $H(x) = 32x$ . The resultant hash value is 6.4. Node  $N_{15}$  sends out *Sort\_Request{ key = 6.4, (0.2, N<sub>2</sub>, N<sub>15</sub>) }* message, which is routed to node  $N_8$ . Node  $N_8$  stores the triplet  $(0.2, N_2, N_{15})$ , since it is the successor node of hash value 6.4.

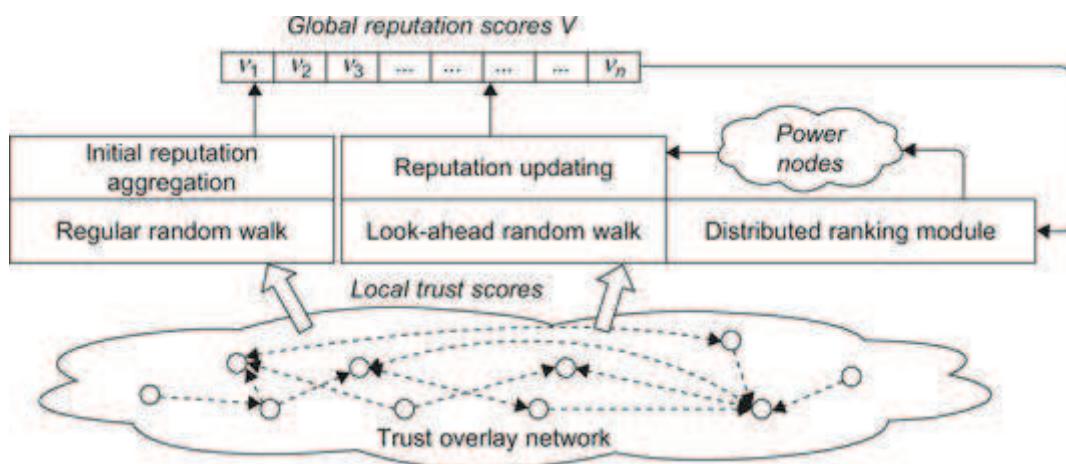


**FIGURE 8.25** Distributed reputation ranking using the locality-preserving hash function over a DHT-based P2P system. Courtesy of R. Zhou and K. Hwang [57]

Node  $N_2$  is the successor node of the maximum hash value 15, so node  $N_2$  initiates the process to find  $m$  power nodes. Node  $N_2$  is responsible for the hash values in the union of two ranges  $[15, 16] \cup [0, 2]$ . Since it has no corresponding triplets within the range  $[15, 16]$ , it stores zero triples with the highest reputation values, that is,  $k = 0$ . Therefore, it sends a  $Top_m\_Request(m = 1, k = 0)$  message to its predecessor node  $N_{15}$ , which finds its stored triplet with value 0.4 being the highest one. So, node  $N_8$  is the most reputable node in this example system. Multiple LPH functions could be used to prevent cheating by the malicious peers.

### 8.4.3 PowerTrust: A Scalable Reputation System

Figure 8.26 shows the major building blocks in a PowerTrust system. First, a TON is built on top of all peers (nodes) in a P2P system. All peers evaluate one another, whenever a transaction takes place between a peer pair. Therefore, all peers send *local trust scores* to one another frequently. These scores are considered the raw data input to the PowerTrust system. The system aggregates the local scores to calculate the global reputation score of each participating peer. All global scores form a *reputation vector*,  $V = (v_1, v_2, v_3, \dots, v_n)$ , which is the output of the PowerTrust system. All global scores are normalized with  $\sum_i v_i = 1$ , where  $i = 1, 2, \dots, n$  and  $n$  is the TON network size.



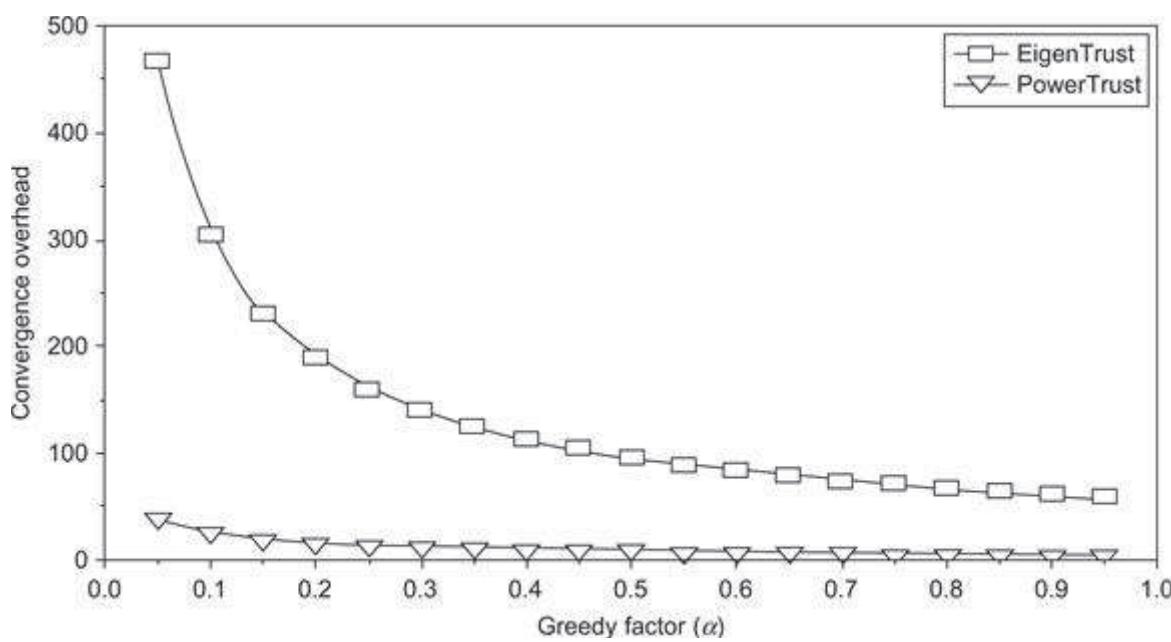
**FIGURE 8.26** Functional modules in the PowerTrust system for trust score collection and global reputation aggregation. Courtesy of Zhou and Hwang [57]

The system is built with five functional modules, as shown in Figure 8.26. The *regular random walk* module supports *initial reputation aggregation*. The *look-ahead random walk* (LRW) module is used to update the reputation score periodically. To this end, the LRW also works with a *distributed ranking module* to identify the power nodes. The system leverages the power nodes to update the global scores' reputation. PowerTrust achieves high aggregation speed and accuracy, robustness to resist malicious peers, and high scalability to support large-scale P2P applications.

#### 8.4.3.1 Reputation Convergence Overhead

The *convergence overhead* is measured as the number of iterations before the global reputation convergence. Convergence means the distance between two consecutive reputation vectors is smaller than the threshold. The EigenTrust approach relies on a few pretrusted nodes to compute the global reputations. They assumed that some peers are known to be trustworthy, essentially among the very first few peers joining the system.

For fairness, the same number of power nodes equal to that of the pretrusted nodes used in EigenTrust is chosen. Figure 8.27 shows the convergence overheads for these two reputation systems. Here, the power nodes in PowerTrust and the pretrusted node in EigenTrust are allowed to leave freely. A sharp drop in iteration count in using PowerTrust to a flat small number less than 50 is observed, when  $\alpha$  increases from 0.15 to 1, while EigenTrust still requires more than 100 iterations to converge. The EigenTrust system overhead can reach as high as 400 iterations as the system increases to 4,000 nodes.



**FIGURE 8.27** Convergence overhead of two reputation systems in a P2P network. Courtesy of Zhou and Hwang [57]

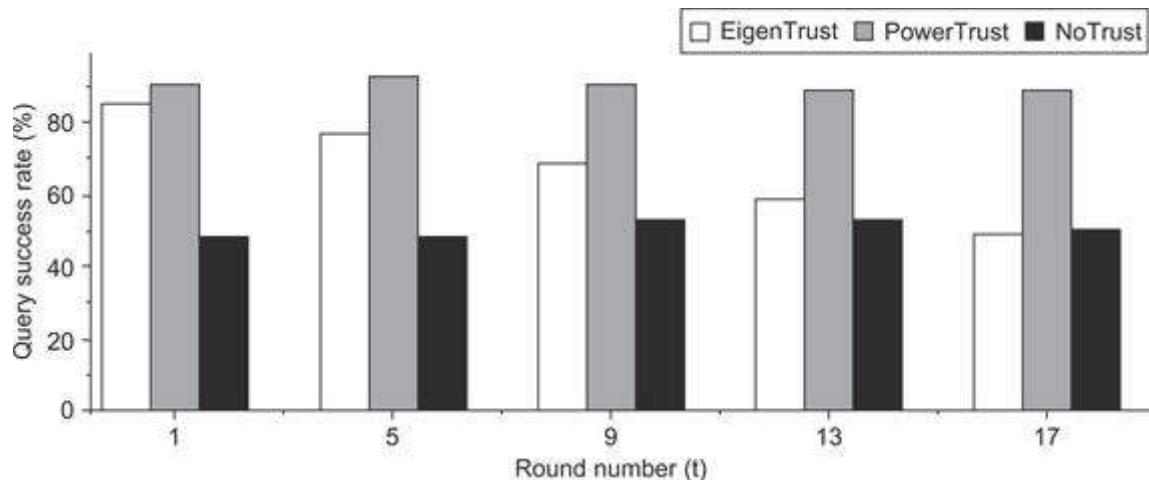
The EigenTrust system converges very slowly. The system cannot guarantee its convergence, when the pretrusted nodes are allowed to leave the system freely. In the PowerTrust system, the power nodes are reelected after each aggregation round. Based on the distributed ranking mechanism, the score managers of the departing power nodes notify the system to replace them in a timely manner with other, more qualified power nodes. The decrease in computation overhead means a significant traffic reduction on the network, and less work for all peers involved. The low overhead in using the PowerTrust system makes it attractive for performing highly scalable P2P applications.

#### 8.4.3.2 Query Success Rate

The PowerTrust system was applied in simulated P2P file-sharing experiments. There are more than 100,000 files in the simulated P2P systems. The number of copies of each file in the system is determined by a content Power-law distribution with  $\beta = 1.2$ . Each peer is assigned with a number of files based on Saroiu distribution. At each time step, a query is randomly generated at a peer and completely executed before the next query/time step. The query distribution determines which file each query searches for. The queries are ranked according to their popularity. A Power-law distribution with  $\beta = 0.63$  for queries ranked 1 to 250 and  $\beta = 1.24$  for the remaining lower ranking queries is used. This distribution models the query popularity in existing P2P systems. When a query for a file is issued, the file owner with the highest global reputation is selected to download the desired file.

The query success rate is measured by the percentage of successful queries over the total number of queries issued. Every node may respond to a query with unauthentic files. For simplicity, this behavior is modeled as inversely proportional to the node's global reputation. Both cases of allowing or disallowing power nodes or pretrusted nodes to leave the system are considered. The case of a *no-trust system*, meaning no trust management in the P2P system, is also evaluated. The no-trust system randomly selects a

node to download the file without considering reputation. Figure 8.28 shows the results without the departure of power nodes or pretrusted nodes.



**FIGURE 8.28** Query success rate in distributed file sharing using two reputation systems to compensate for the free departure of power nodes or pretrusted peers. Courtesy of Zhou and Hwang [57]

There are 1,000 queries issued after each round of global reputation aggregation. The query success rate of PowerTrust is maintained at 90 percent after just one round of reputation aggregation. The query success rate of EigenTrust drops from 85 percent to 50 percent after several aggregation rounds. This is because the pretrusted nodes may leave the system randomly. The EigenTrust query success rate drops to a 50 percent low, equal to that of a no-trust system, after 17 rounds of reputation aggregation.

#### 8.4.4 Securing Overlays to Prevent DDoS Attacks

The security of a P2P network may be in doubt if the peer nodes behave maliciously to attack other innocent peer machines. The attackers could be hidden in the P2P group. They can view or modify packet routing or can collude with other peers to stage the attacks or to steal the contents. What can be done to hinder the attacks is crucial to achieve harmonious operation in a P2P network. Four classes of network attacks often occur in a P2P network:

1. A P2P network could enter churn condition, if large number of peers join or leave the network rapidly or randomly. Churn could create inconsistent behavior or resources deadlocks.
2. Distributed denial of service (DDoS) attacks are caused by flooding attacks on a target node.
3. Routing attackers attempt to re-route messages in order to steal contents or to stage DDoS attacks.
4. Storage/retrieval attacks are caused by attackers to prevent the delivery of the requested data.

To become resilient to network chucks, one can force nodes to sign all messages so that inconsistency can easily detected. To cope with DDoS attacks, one can replicate the

content and spread over the network. The SOS example below presents a more sophisticated solution. For *identity spoofing attacks* caused by a peer node to use other node's identity or to deliver bogus contents. One solution is to insist that all peers must have certificates signed by a trusted authority. Another solution to prevent proofing is tying the identity with protected IP address or send query to verify the address.

To prevent routing attacks, one solution is to use a converging policy iteratively so that each hop moves closer to the destination in the ID space. Another solution is to provide multiple paths to re-route messages around the attackers. Onion routing is also used to hide the identify of the originator of the final destination. This done by encryption of the incoming message and the next hop information recursively using public keys. This way, any node only knows who was the immediate sender and who is the next receiver. To combat storage/retrieval attacks, one can replicate the data object to enhance availability.

### **Example 8.14**

#### **SOS Scheme to Prevent DDoS Attacks[21]**

SOS is an architecture for *Secure Overlay Services*, which was proposed by Keromyts, et al. [21] in 2002. The main goal of SOS to allow a majority of legitimate or authenticated users to communicate with a secure checkpoint node, where DDoS attacks can be stopped to be forwarded to the final target machine. This is made possible by making the target address difficult to replicate or legitimate users may be mobile using changing IP addresses. Good examples of SOS include FBI, police or bank officers, who always check with their central database.

A SOS system will protect legitimate users and innocent target machines from DDoS attacks. At the same time, the attackers are blocked or caught. To this end, we need to screen good users into a pre-approved class. Attackers should not allowed to attack a bound number of distributed end systems. The construction of an SOS system requires to perform filtering, to hide the proxy as *secret servlets*, to build reliable overlays with hidden proxies, and to advertise the trustworthy access points for users to join the P2P system. The SOS can be also assisted by using the Chord with SOAP servers or adding redundancy in SOAP servers. More details on the effectiveness of SOS preventing DDoS attacks can be found in [21].

## **8.5 P2P File Sharing and Copyright Protection**

P2P technology enables files to be shared among peers in a distributed way freely. The client first performs search operations to locate peers holding the files the client wants.

The client downloads the files directly from these provider nodes. The ultimate goal is to deliver content to all requesters as fast as possible. Caching of P2P content is helpful to improve content downloading speed and traffic locality. This section reviews fast file search methods. Replicas and consistency techniques to enhance delivery speed and success rate are discussed as well.

### 8.5.1 Fast Search, Replica, and Consistency

For P2P file-sharing applications, search algorithms play the most important role. A search algorithm determines which nodes hold the specified files. The performance of a search algorithm is evaluated by two metrics: query path length and message overhead. Query path length is measured by the average number of nodes a query message traverses before reaching the destination, while message overhead is measured by the average number of query messages forwarded for one search operation.

Search algorithms for file-sharing applications based on structured P2P overlays and unstructured P2P overlays are different. In a structured P2P overlay application, keys of data objects and nodes are mapped to the same identifier space and the data object information is stored on the node that holds the identifier zone within which the key is mapped. Search algorithms are like routing algorithms in overlay networks. In an unstructured P2P overlay application, however, each node often only holds the information of the data objects that it shares. The query message has to traverse a lot of nodes until it reaches the destination. In the following paragraphs, we discuss algorithms for structured and unstructured P2P overlay applications.

In structured P2P overlay networks, each node is responsible for a portion of the identifier space. For example, in Chord, a node is responsible for the range from its predecessor to the node on the ring identifier space. The information of the data objects whose keys are mapped to the identifier range held by node  $x$  is stored on  $x$ . Node  $x$  is called the successor of the key. Searching a key is equal to routing the query message to the successor of the key. Hence, the performance of search algorithms is similar to that of routing algorithms, which means a key is always located in  $O(\log n)$  hops with the overhead of  $O(\log n)$  messages, where  $n$  is the number of nodes in the application.

Search in applications based on unstructured P2P overlay networks is more complicated. Basically, the search algorithms can be classified as blind search and informed search. Blind search algorithms are for applications where nodes do not maintain the metadata of files shared by others, while informed search algorithms are for applications where nodes store some information of the files shared by others to facilitate the search. Blind search algorithms are often referred to as flooding algorithms.

In flooding algorithms, when a query message is first received, a node forwards it to  $k$  randomly selected neighbors, until the TTL of the message shrinks to 0. The query messages are forwarded round by round like a wave spreading from the source. Flooding algorithms inevitably bring considerable message overhead. A simple method to reduce overhead is to limit the TTLs of query messages. However, this would increase the probability that objects far away from the query nodes cannot be located. Various

algorithms have been proposed to reduce message overhead, such as expanding ring and random walk.

In expanding ring algorithms, a query node is associated with a schedule. Initially, the TTL of a query message is set to  $t_1$  by the query node. If the query cannot be satisfied, the TTL grows to  $t_2$ . Eventually, the TTL grows to the maximum value  $t_r$ . The successive retries facilitate overhead reduction at the cost of a slight increase in query path length.

Random walk algorithms reduce overhead from another aspect. A query node deploys  $k$  independently random walkers. Each walker carries a query message and walks on the overlay graph, until the desired object is located. Random walk algorithms suffer from the TTL selection problem just like flooding algorithms do. However, in random walk algorithms, each round only increases the number of visited nodes by  $k$ . Therefore, it is reasonable for walkers to periodically check with the query node to determine whether to continue or stop. Random walk algorithms generally outperform expanding ring algorithms and save on message overhead by two orders of magnitude at a cost of a slightly increase in query path length compared to flooding algorithms.

The principles of scalable blind search algorithms for unstructured P2P overlays are summarized as follows. First, the algorithms should adopt adaptive termination. Simply limiting the TTL values does not work well as that would adversely affect search scope. Second, the number of visited nodes should grow slowly with the increase in search round. This is because only a small portion of the nodes are required to be visited. If the number of visited nodes grows quickly (e.g., exponentially in flooding algorithms), most of the messages generated at the last several hops are duplicated.

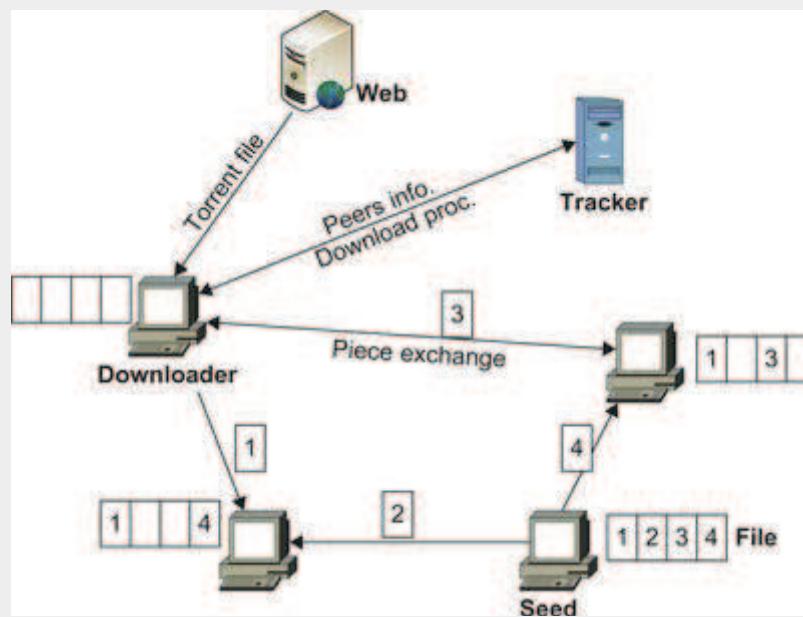
The message overhead incurred by blind search algorithms can be greatly reduced if nodes keep some information regarding the files shared by others. Informed search algorithms are designed for this purpose. Each node keeps some information regarding the files shared by others and the information is used to guide the forwarding of query messages. The idle case is that each node keeps all the files shared by others, and thus the query path is only one hop long. However, this is not feasible because it requires not only unaffordable storage, but also lots of synchronize overhead among peers.

Search performance can also be enhanced through overlay topology adaptation. In Gnutella, one can take heterogeneity of node capacity into account by assigning more loads on high-capacity nodes. High-capacity nodes adaptively build more overlay links and keep the file indices of the nodes one hop away. Since random walk is biased toward the high-degree node, most queries are forwarded to the high-capacity nodes and satisfied there. Another topology adaptation is to consider semantic interests of nodes. Semantically relevant nodes are organized into the same semantic clusters on overlays. Queries are forwarded to semantically relevant clusters for answers. The idea behind this is that if nodes are semantically relevant, it is likely that they are relevant to the same queries. BitTorrent is the most popular P2P content distribution application. It uses swarming content distribution.

## Example 8.15

### BitTorrent File-Sharing Network

BitTorrent is a P2P content distribution protocol that is widely used in the current Internet. It is reported by Ipoque that BitTorrent content distribution systems account for 27 percent to 55 percent of Internet traffic. A BitTorrent system leverages the upload bandwidth of participant peers to achieve high system capacity. The nodes downloading the same file constitute a torrent. There is no communication between different torrents. A typical BitTorrent system consists of four components as shown in Figure 8.29.



**FIGURE 8.29** The BitTorrent system architecture. Courtesy of “BitTorrent Protocol Specification”; [www.bittorrent.org/protocol.html](http://www.bittorrent.org/protocol.html), 2006

Seeds are the nodes that have already downloaded the whole file. There is at least one initial seed in the system at startup. A node that has not gotten the whole file is called a downloader. The tracker tracks the nodes' information, including the address and download process information. The tracker also knows the seeds in the system. The web server hosts the torrent file, which stores the metadata of the file to be distributed and the tracker's information.

The file is divided into pieces of equal size, ranging from 32 KB to 4 MB, with a typical size of 256 KB. The publisher of the file creates a hash for each piece using the SHA-1 hash function. The hash for each piece is used to verify the integrity of the piece during downloading and is stored in the torrent file. A file piece may be further divided into blocks. In this case, each piece is a trading unit, while a block is a request unit. Only after all the blocks of a piece have been downloaded can this piece be publicly shared with others.

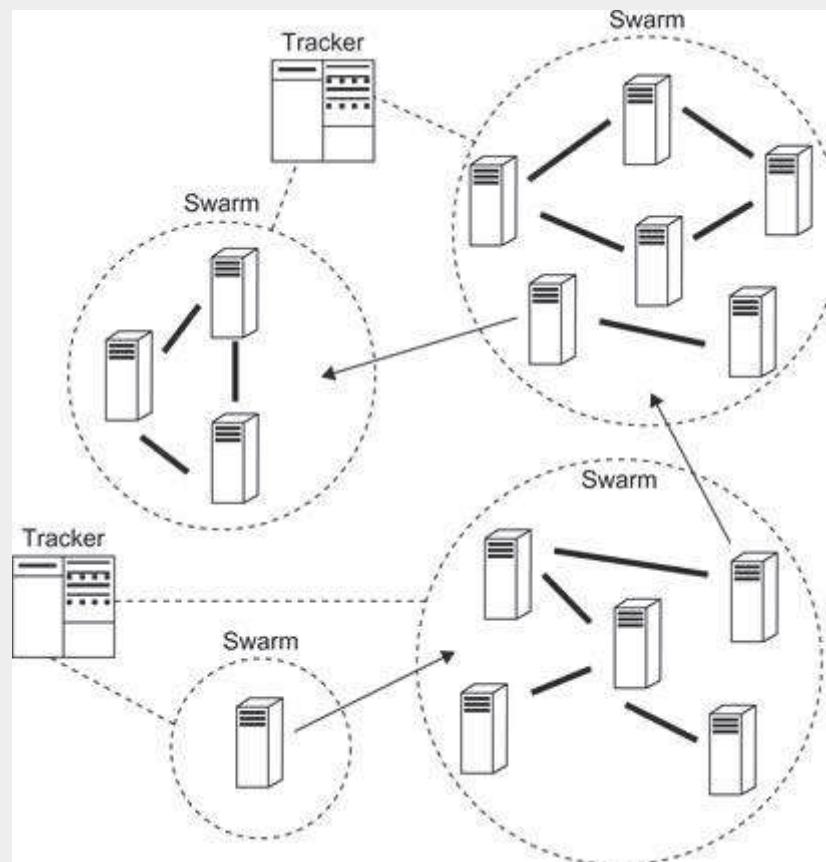
A new node first gets the torrent file from a web site. By parsing the file, it

gets the tracker's address and asks the tracker for peers' information. Usually, the tracker responds with a peer list containing 50 randomly selected peers. The number of TCP connections initialized by a peer is limited to 35 and the total number of connections of a peer is limited to 55. If ever a client fails to maintain at least 20 connections initialized by it, it recontacts the tracker to obtain additional peers. To update the tracker's global view of the system, active nodes periodically (every 30 minutes) report their states to the tracker or at the time of joining or departure.

Each peer has a bit-vector indicating the pieces' availabilities on it, one bit for each piece. The bit-vectors are exchanged between neighbors. If peer  $y$ 's neighbor  $x$  finds that  $y$  has a piece that it does not have,  $x$  sends an "interest" message to  $y$  to request that piece. The decision by  $y$  of whether and when to send the requested piece to  $x$  is based on an incentive strategy called *Tit-for-Tat (TFT)*. All connections are choked by default. Each peer periodically (every 10 seconds) updates its download rates from all of its neighbors, and chooses four neighbors.

The TFT is an incentive strategy to combat against free riders, the peers that contribute nothing or very limited upload bandwidth. A peer also uses an optimistic unchoking strategy to unchoke a uniformly chosen peer at random every 30 seconds, which enables the nodes that have no or very limited file pieces to start up. When unchoked by a neighboring peer, a node selects the file piece that is the least replicated among its neighbors. This selection is called *rarest first (RF)* (locally) piece selection. However, for a new peer that has no file piece, it randomly chooses a piece to download. The RF piece selection scheme is used to enable random distribution of file pieces on overlay networks, which facilitates the diversities of pieces on neighboring peers.

Figure 8.30 shows the hierarchical structure of multiple swarms built up in a BitTorrent network. A swarm is a collection of peer machines that form a virtual cluster under the coordination of special trackers. Interswarm traffic is possible if the local swarm cannot satisfy a user request. Multiple trackers exchange updated directory information to the global view consistently. The idea is quite similar to that of the KaZaA architecture shown in Figure 8.11. The purpose is to upgrade management efficiency and shorten search time for users.



**FIGURE 8.30** An operational view of a BitTorrent system built with multiple swarms of resource nodes under the coordination of special trackers.

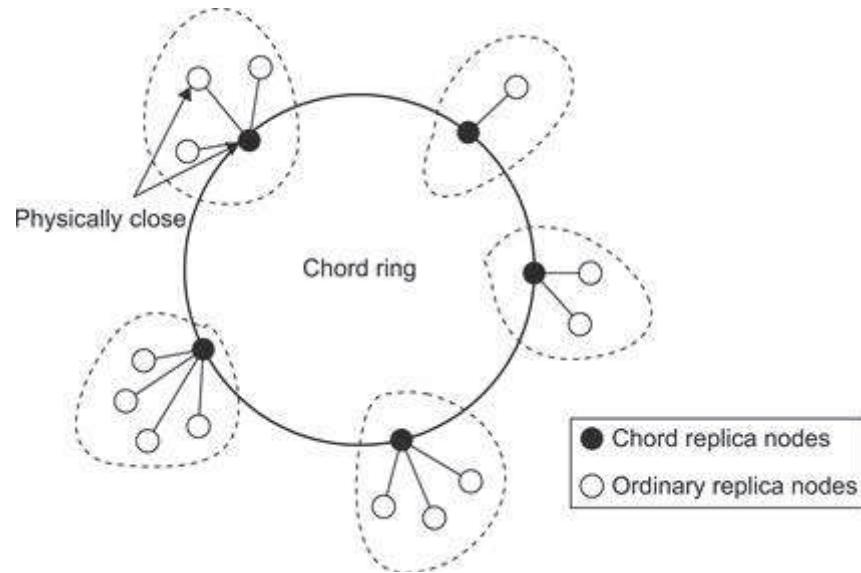
By convention, the name of a torrent file has the suffix (.torrent). Torrent files have an “announce” section, which specifies the URL of the tracker, and an “info” section, containing the suggested names for the files, their lengths, the piece length used, and an SHA-1 hash code for each piece, all of which are used by clients to verify the integrity of the data they receive. The tracker maintains lists of the clients currently participating in the torrent. Alternatively, in a decentralized tracking which works without a tracker, every peer acts as a tracker. Azureus was the first BitTorrent client to implement such a system through the DHT. An alternative and incompatible DHT system, known as Mainline DHT, was later developed and adopted by BitTorrent, µTorrent, and others.

### 8.5.1.1 Replica and Consistency

Replication is critical to improving search performance in P2P file-sharing applications. The number of replicas for an object is affected by the popularity of that object. Intuitively, more replicas should be created for highly popular objects. The number of nodes that hold the desired objects has a direct impact on search performance. This is quite intuitive in that as a greater number of desired objects scatter over the overlays, it becomes easier to find them.

A question related to replication concerns maintaining consistency of replicas. If the data object is modified or updated by a replica node (i.e., the node maintains a replica of the object), the updated content should be propagated to all the other replica nodes. Consistency maintenance schemes provide support for applications dealing with frequently updated contents. Two key issues of consistency maintenance concern how to track the replicas of an object and how to propagate the updated contents to the replica nodes. The centralized scheme dedicates one node (often the root node of the object) to keep the information of all the replica nodes and relies on this node to forward the updated content. However, it lacks scalability.

In the locality-aware and distributed consistency maintenance scheme [25], the nodes that store the replicas of an object constitute a replica group. The centralized scheme is sufficient for the replica group with fewer replica nodes. Thus, our focus is on larger replica groups; for example, replica groups of popular objects. An auxiliary hierarchical overlay network is established for each replica group: The upper layer is DHT-based and consists of powerful and stable replica nodes of the key, and the ordinary replica nodes at the second layer attach to physically close upper-layer nodes. [Figure 8.31](#) shows the structure, where the upper layer is based on a Chord ring structure.



**FIGURE 8.31** Auxiliary structure for a replica group.

When an updated replica emerges on a node, the corresponding upper-layer node dynamically initializes a  $d$ -ary source-specific tree on top of the upper layer by continuously partitioning the DHT identifier space. Suppose upper-layer replica node  $x$  receives an update request from one of the ordinary replica nodes that attaches to it. Initially, node  $x$  holds the whole identifier space represented by the Chord ring structure. It divides the identifier space into  $d$  equal parts. The first node along the clockwise direction within each part is selected as the representative node of that part. All the representative nodes constitute the root node  $x$ 's child node set. Each identifier part is further partitioned into  $d$  subparts of equal size, and so on, until there is only one replica node in the identifier part.

The updated content is propagated along the tree structure in a top-down fashion and the upper-layer replica node takes charge of forwarding the updated content to the ordinary replica nodes that attach to it. Since the upper-layer node and the ordinary replica nodes that attach to it are physically close, the updated contents are transferred quickly and network bandwidth consumption is greatly reduced. The tree structure is destroyed from bottom to top after the update operation completes, in order to save on maintenance overhead. Now, let's briefly analyze the time (measured by logical hops on the overlays) and the control overhead required by an update operation.

## 8.5.2 P2P Content Delivery Networks

This section applies P2P technology to digital content networking and distribution. First, three generations of *content delivery networks* (CDNs) are reviewed. Then we discuss how P2P technology can enhance the QoS of CDNs.

### 8.5.2.1 Three Generations of CDNs

Table 8.6 compares three generations of CDNs in the past 30 years. The early days of online distribution systems follow the client/server architecture using FTP or HTTP services. This generation is limited by the processing power of a single server. For distributing small files, this approach is still acceptable. Good examples include the Apache, GetRight, and CuteFTP services. The second generation is marked by the use of multiple surrogate servers and large-scale distribution networks mainly for video streaming and software distribution.

**Table 8.6**

**Three Generations of Content Distribution Networks**

System Generations	Network Architecture	Download Bandwidth	Content Applications	Example Systems and Web Sites
First generation: client/server system	Clients access server using FTP and HTTP	Limited by single-server bandwidth	Small files with limited downloads	Apache ( <a href="http://www.apache.com/">www.apache.com/</a> ) GetRight ( <a href="http://www.getright.com/">www.getright.com/</a> ) CuteFTP ( <a href="http://www.cuteftp.com/">www.cuteftp.com/</a> )
Second generation: CDNs	Clients connect to multiple servers	Load balancing among servers	Video streaming, virus signatures	Akamai ( <a href="http://www.akamai.com/">www.akamai.com/</a> ) SyncCast ( <a href="http://www.synccast.com/">www.synccast.com/</a> )
Third generation: P2P file-sharing system	P2P overlay network	Low; peers are not secured and are unreliable	Large files, speed is not critical	BitTorrent ( <a href="http://www.bittorrent.org/">www.bittorrent.org/</a> ) eDonkey ( <a href="http://www.edonkey.com/">www.edonkey.com/</a> ) eMule ( <a href="http://www.emule.org/">www.emule.org/</a> )

Courtesy of Lou and Hwang [31].

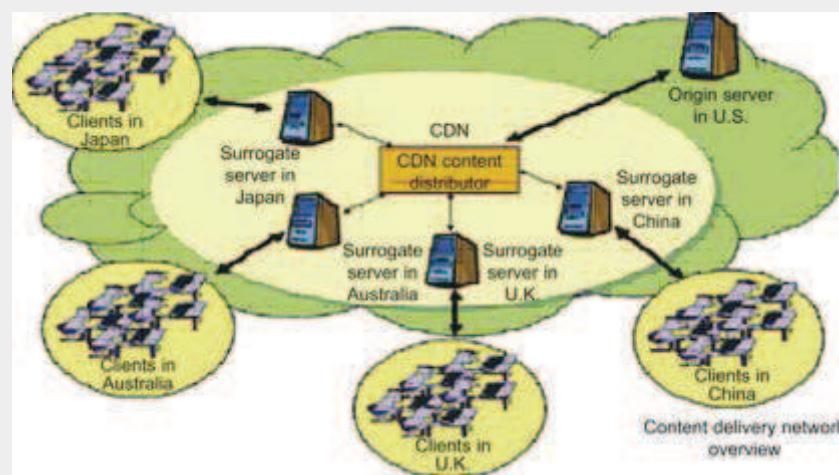
The Akamai CDN is a good example of a large-scale CDN as illustrated in Example 8.14. Finally, the P2P file-sharing networks mark the appearance of the third generation of CDNs. BitTorrent and eMule are good examples. P2P networks definitely are more cost-effective than the first two generations. However, severe abuses on copyright

violations in P2P file-sharing systems have hindered the use of open P2P networks for commercial content distribution. We discuss the online piracy problem in the next two subsections. Selective content poisoning poses a possible solution to this problem.

### Example 8.16

#### A Global Content Distribution Network

In 2004, more than 3,000 companies used CDNs. They spent more than \$20 million monthly. CDN providers have doubled their revenue almost every year since then. In 2005, worldwide CDN revenue grew to \$450 million in the delivery of news, film, sports, music, and video content. [Figure 8.32](#) shows the conceptual architecture of a global-scale CDN, modeled after the Akamai Technologies system. By 2006, the Akamai CDN provided 80 percent of overall content delivery services. At that time, more than 12,000 surrogate servers were used in various parts of the world in more than 1,000 networks in 62 countries.



**FIGURE 8.32** The concept of a global CDN using surrogate servers in major user regions or countries. Courtesy of Pallis and Vakali [37]

This number is now much greater. Surrogate servers are used as cache centers for distributing digital content more cost-effectively in terms of transmission, storage, access speed, and fault tolerance requirements. Each surrogate center is responsible for distributing the requested content to all end users in each country or region. The CDN content distribution center coordinates the traffic among the servers. For copyright protection reasons, open P2P technology is not used here for commercial content distribution.

A P2P network does not require many expensive servers to deliver content. Instead, content is distributed and shared among the peers. P2P networks feature improvements over conventional CDNs in terms of content availability and system scalability. Electronic publishing was hindered by the rapid growth of copyright violations. The major source of illegal P2P content distribution lies in peer collusion to share copyrighted content with other peers or pirates.

Digital watermarking is often considered for digital copyright protection. Digital watermarking is injected into a content file so that when a pirated copy is discovered, authorities can find the origin of the piracy via a unique watermark in each copy. In a P2P network, all peers are sharing exactly the same file (if it is not poisoned), which effectively defeats the purpose of watermarking. Thus, watermarking is not a suitable technology for P2P file sharing. By subdividing a large file into smaller chunks, P2P content delivery allows a peer to download multiple chunks from different sources. File chunking increases availability and decreases download time. [Table 8.7](#) summarizes three major P2P CDN families.

**Table 8.7**  
**File Chunking and Hashing Schemes in P2P Content Networks**

P2P Networks	BitTorrent Family	Gnutella Family	eMule Family
Chunking Scheme	Divides files into fix-sized chunks (256 KB), called pieces	Peers negotiate the chunk size at runtime, 64 KB/chunk	Divides into 9,500 KB parts, each has 53 (180 KB) chunks
Hash Distribution	SHA hashing at piece level, embedded in index file	SHA hashing applied to entire file to generate a unique file ID, no chunk-level hashing	MD-4 hashing at part level, peers exchange hash set to detect corrupted contents
Poison Resistance	Poison detected at piece level, each is handled independently	Poison detected only after downloading the entire file, heavy overhead if poisoned	Poison detected at part level, works if part hash set is not poisoned
Download Policy	Keeps clean and discards poisoned pieces, repeats download until all chunks are clean	Repeats downloading entire file until all chunks are clean, a time-consuming policy	Keeps clean and discards poisoned parts, downloads until all parts are clean
Example Networks	BitTorrent, Snark, BitComet, BNBT, BitTyrant, Azureus, JTorrent, etc.	Gnutella, KaZaA, LimeWire, Phex, Freenet, BareShare, Swapper, Ares, etc.	eMule, aMule, Shareaza, iMule, Morpheus, eDonkey, FastTrack, etc.

P2P networks in the same family have some common features. They are variants or descendants of their predecessors: *BitTorrent*, *Gnutella*, and *eMule*, respectively. These families are distinguished primarily by the file-chunking or hashing protocols applied. At the time of this writing, none of these P2P networks has been built with satisfactory support for copyright protection. The BitTorrent family applies the strongest hashing at the individual *piece* or chunklevel, which is most resistant to poisoning. The Gnutella family applies file-level hashing, which is easily poisoned. The eMule family applies part-

level hashing with fixed chunking.

Our analytical and experimental results show that the eMule family demonstrates a moderate level of resistance to content poisoning. The ability to detect and identify poisoned chunks is different in the three P2P network families. The BitTorrent family keeps clean chunks and discards poisoned chunks. The Gnutella family must download the entire file before any poisoned chunks can be detected. Because of part-level hashing, the eMule family will either keep or discard the entire part of 53 chunks.

### **8.5.2.2 Content Delivery Methods**

P2P content distribution applications take advantage of the abundant computing resources at the edge networks to distribute content. Nodes which are interested in the same content form a P2P overlay network. The data content is sent out from sources, stored, and forwarded among peers on the overlay networks. Basically, there are three approaches to distributing data content: flooding-based, tree-based, and swarm-based. In flooding-based approaches, data content is flooded on the overlay networks. There are too many duplicated deliveries to be tolerated.

The tree-based approach follows a tree structure and delivers the content along the tree. Since peer nodes are likely to join, leave, and even fail at any time at will, it is a great challenge to maintain the tree structure. One can also build multiple trees. Data content is partitioned into several chunks. Each chunk is delivered along one tree. A node resides at all the trees to get the data objects. Multiple tree-based approaches enable fast content delivery and improve the robustness of data distribution. However, vulnerability to node churn is still the most critical concern for this kind of approach.

Swarm-based approaches are built on top of mesh overlay networks. They are also called data-driven approaches. Data content is pulled by the participants, rather than distributed actively by the data sources. Data content is also partitioned into small chunks. Each node keeps two “windows” indicating what it has and which chunk it requires, respectively. Nodes periodically exchange their “windows” with their neighbors and get the missing chunks actively.

Although swarming content distribution requires more time than the tree-based approach, it is widely used and deployed in current applications (e.g., PP live, BT, etc.) because of its scalability and ease of implementation, among other features. The order of the chunks that a node gets is random, that is, no specified schedule is imposed. This is reasonable for applications that do not care about the arrival order of the chunks (e.g., BitTorrent). However, other applications require control of orders (e.g., the live streaming applications). In these applications, a chunk must appear on a node before its playback deadline, or it is useless. Basically, the rule for a node is to get the chunks whose deadlines are approaching and the chunks which are rarely replicated among its neighbors.

Recently, P2P has become popular in the context of dynamic information dissemination. Nodes that are interesting in the same event are organized in a P2P overlay network. Data items emerge at nodes dynamically, which means the data sources are no longer fixed. Data items are replicated and forwarded among peers. The delivery

algorithms in this context are required to distribute the items appearing in any node to all the others as soon as possible with relative small network resource consumption. Swarm-based approaches are not suitable in this scenario because they suffer from a delay-versus-overhead trade-off. That is, to minimize data delivery delay, the intervals of “window” information exchange should be small.

However, this would increase control overhead. On the other hand, increasing the intervals would prolong the delivery delay. Building one tree for each potential data source is also not a good idea due to the considerable maintenance overhead involved. Building one undirected tree structure for all the sources is optimal for controlling overhead. However, if the data source is a leaf node, it would take as many as  $2h$  hops to transmit the data item to all members, where  $h$  is the height of tree. Another concern is how to maintain the tree structure under node churn.

### 8.5.3 Copyright Protection Issues and Solutions

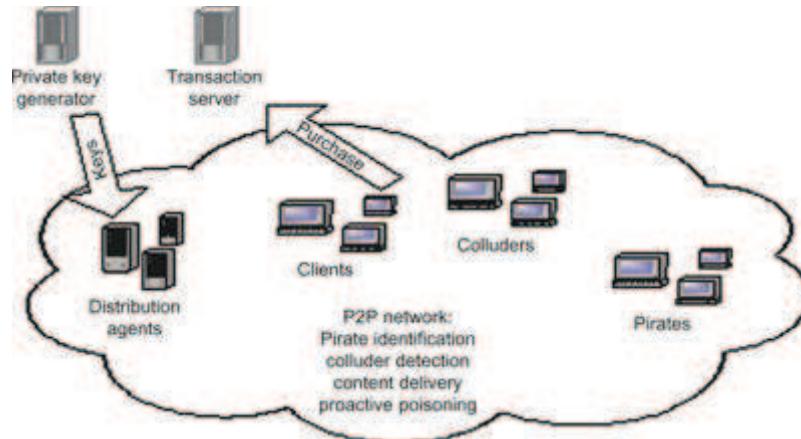
P2P networks are cost-effective in delivering large files to massive numbers of users. Unfortunately, today’s P2P networks are grossly abused by illegal distribution of music, games, video streams, and popular software. These abuses have not only resulted in heavy financial loss in the media and content industries, but also hindered the legal commercial use of P2P technology. The main sources of illegal file sharing are peers who ignore copyright laws and collude with pirates. To solve this peer collusion problem, a copyright-compliant system for legalized P2P content delivery is proposed [31].

The goal is to stop collusive piracy within the boundary of a P2P CDN. In particular, the scheme appeals to protecting large-scale perishable contents that diminish in value as time elapses. Traditional CDNs use a large number of surrogate content servers over many globally distributed WANs. The content distributors need to replicate or cache contents on many servers. The bandwidth demand and resources needed to maintain these CDNs are very expensive. A P2P content network significantly reduces the distribution cost, since many content servers are eliminated and open networks are used. P2P networks improve content availability, as any peer can serve as a content provider. P2P networks are inherently scalable, because more providers lead to faster content delivery.

Honest or legitimate *clients* are those that comply with the copyright law not to share contents freely. *Pirates* are peers attempting to download some content file without paying or authorization. The *colluders* are those paid clients who share the content with pirates. Pirates and colluders coexist with law-abiding clients. *Content poisoning* is implemented by deliberate falsifying the file requested by the pirate. The media industry backed by the *Recording Industry Association of America* (RIAA) and *Motion Picture Association of America* (MPAA) has applied unscreened brute force content poisoning to deter piracy in open P2P file-sharing networks. However, their prevention results are mixed and controversial.

The media industry fight against all P2P file-sharing services by brute force. Legitimate clients can still enjoy the flexibility and convenience provided by an open P2P network.

The scheme stops pirates from downloading copyrighted files, even in the presence of colluding peers. A reputation scheme is used to detect these colluders. A copyright-protected P2P network should benefit both the media industry and Internet user communities [6]. Figure 8.33 suggests a copyright-protected P2P network. As originally developed by Lou and Hwang [31], the network is built over a large number of peers. Four types of peers coexist in the P2P network: clients (honest or legitimate peers), colluders (paid peers sharing content with others without authorization), distribution agents (trusted peers operated by content owners for file distribution), and pirates (unpaid clients downloading content files illegally).



**FIGURE 8.33** A secure P2P platform for copyright-protected content delivery. Courtesy of Lou and Hwang [31]

To join the system, clients submit requests to a *transaction server* that handles purchasing and billing matters. A *private key generator* (PKG) is installed to generate private keys with *identity-based signatures* (IBSes) for securing communication among the peers. The PKG's role is similar to a certificate authority (CA) in PKI services. The difference lies in the fact that the CA generates the public/private key pairs, while PKG only generates the private key.

The transaction server and PKG are only used initially when peers are joining the P2P network. With IBS, the communication between peers does not require an explicit public key, because the identity of each party is used as the public key. In our system, file distribution and copyright protection are completely distributed. Based on past experience, the number of peers sharing or requesting the same file at any point of time is in the hundreds. Depending on the variation of the swarm size, only a handful of distribution agents are needed. For example, it is sufficient to use 10 PC-based distribution agents to handle a swarm size of 2,000 peers. These agents authorize peers to download and prevent piracy by unpaid peers.

Paid clients, colluders, and pirates are all mixed up without visible labels. The copyright-protection network is designed to distinguish them automatically. Each client is assigned a *bootstrap agent*, selected from one of the distribution agents, as its entry point. In current P2P networks, a peer can self-assert its username without verification. Therefore, the peer endpoint address (IP address + port number) instead of the

username is used to identify a peer. A peer is considered fully connected if it is reachable via a listening port on its host.

Table 8.8 summarizes the key protocols and mechanisms used to construct a trusted P2P system. In this approach, the modified file index format enables pirate detection. Peer Authorization Protocol (PAP) authorizes legitimate download privileges to clients. The content distributor applies content poisoning to disrupt illegal file distribution to unpaid clients. The system is enhanced by randomized collusion detection. In the system, a content file must be downloaded fully to be useful. Such a restraint is easily achievable by compressing and encrypting the file with a trivial password that is known to every peer. This encryption does not offer any protection of the content, except to package the entire file for distribution.

---

### Table 8.8

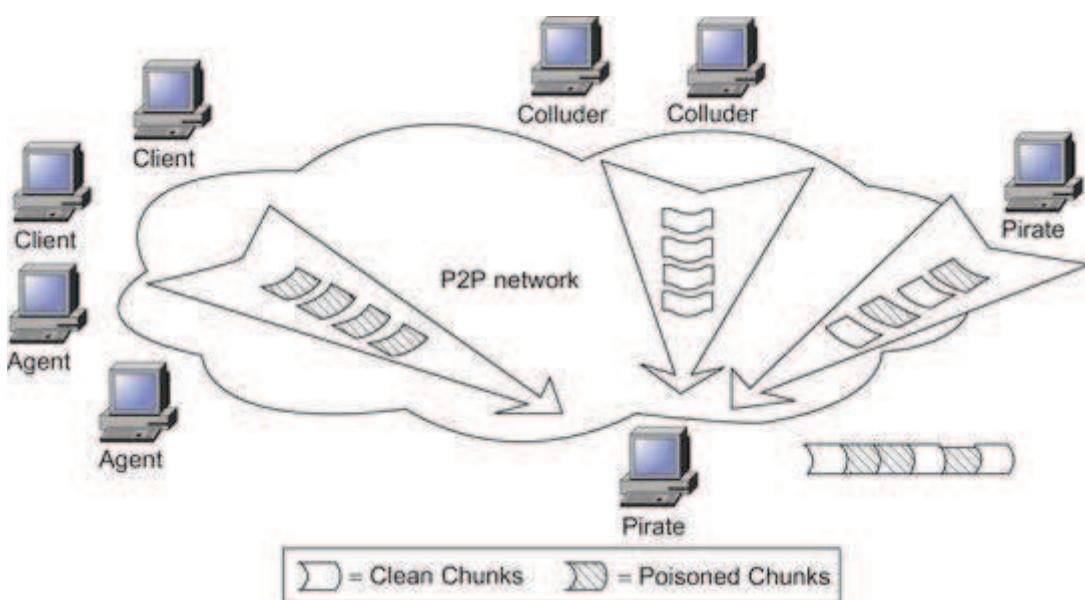
#### Mechanisms for Copyright Protection in P2P Networks

---

Mechanism	Protocol Requirements
Secure file indexing	File index format is modified to include token and IBS signature.
PAP	Peer sends digital receipt to bootstrap agent and obtains an authorization token. The token must be refreshed periodically.
Proactive content poisoning	The token and IBS signature check all download requests and responses, sending clean or poisoned contents, accordingly.
Random collusion prevention	Distribution agents randomly recruit decoys to probe for colluders. Collusion reports are weighted against client trust rates.

### 8.5.4 Collusive Piracy Prevention in P2P Networks

Figure 8.34 illustrates the concept of proactive content poisoning. If a pirate sends a download request to a distribution agent or a client, based on the protocol definition it will receive poisoned file chunks. If the download request was sent to a colluder, it will receive clean file chunks. If a pirate shares the file chunks with another pirate, it could spread the poison. Therefore, it is critical to send poisoned chunks to pirates to deny their requests. Otherwise, the pirate still can assemble a clean copy from those colluders who have responded with clean chunks. With poisoning, we exploit the limited poison detection capability of P2P networks and force a pirate to discard the clean chunks downloaded with the poisoned chunks. The rationale behind such poisoning is that if a pirate keeps downloading a corrupted file, the pirates will eventually give up their attempts out of frustration.

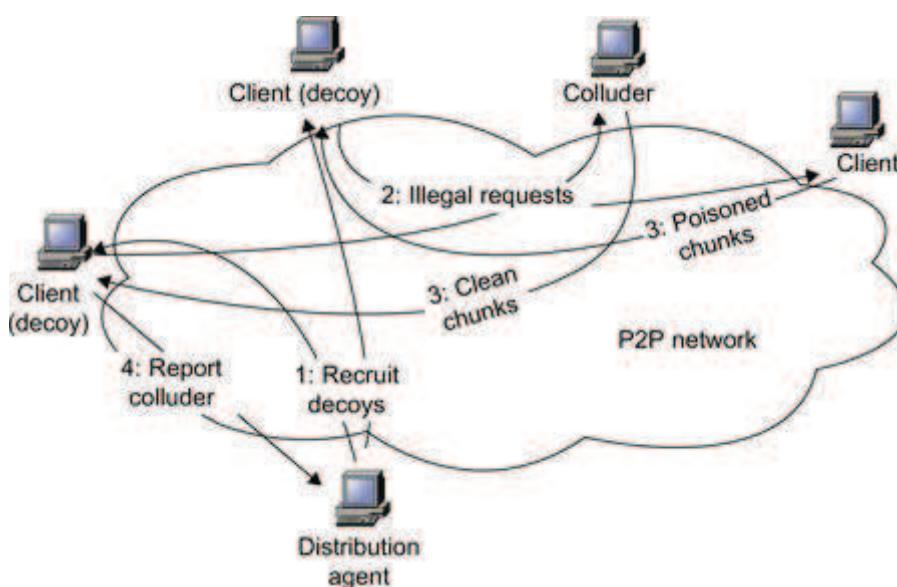


**FIGURE 8.34** Proactive poisoning in a trusted P2P network, where clean chunks (white) are received by legitimate clients and poisoned chunks (shaded) by detected pirates. Courtesy of Lou and Hwang [31]

#### 8.5.4.1 Randomized Colluder Detection

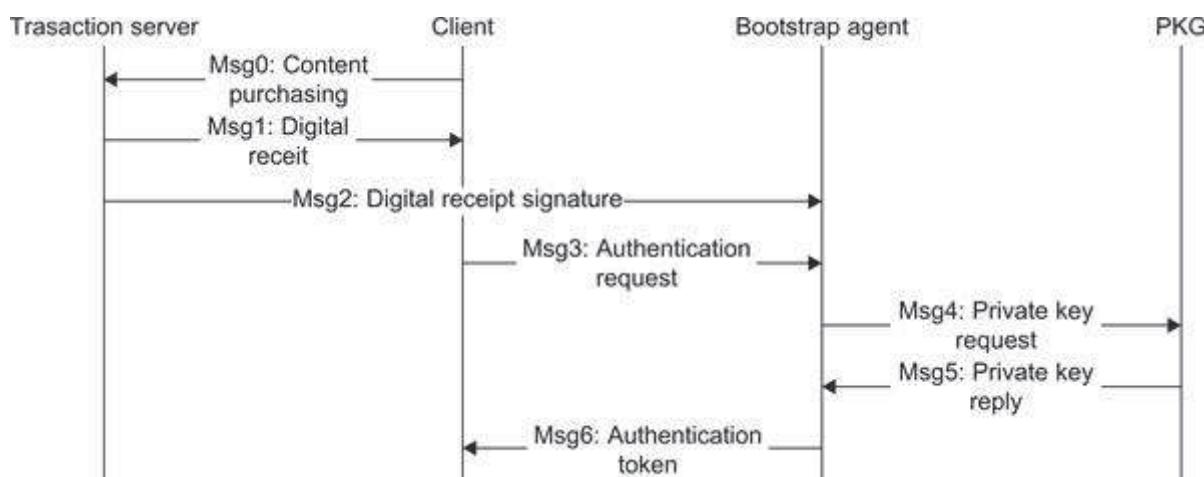
As reported in the GossipTrust paper [58], the gossip protocol and power nodes play a crucial role in speeding up the reputation aggregation process in a P2P network. Randomized gossiping can reach consensus among all peers in a distributed manner. This approach exploits massive concurrency among millions of active nodes in a very large P2P network. Figure 8.34 shows a simplified GossipTrust system to identify colluders.

The idea is to associate each  $\{peer, file\}$  pair with a *collusion rate*. The “0” rate means the peer was never reported as a colluder. Otherwise, the peer is getting a collusion report of “1,” meaning it has shared clean content with illegal download requesters. This collusion rate is accumulative in the same way eBay collects peers’ reputation scores. Figure 8.35 illustrates the collusion detection process. Distribution agents randomly recruit clients, called *decoys*, to send illegal download requests to suspected peers. If an illegal request is returned with a clean file chunk, the decoy reports the collusion event. Since the decoy is randomly chosen, there exists a risk that the report is not trustworthy either by error or by cheating. Thus, a reputation system is needed to screen the peers.



**FIGURE 8.35** Distribution agent randomly recruiting some clients to probe suspected peers; peer collusion is reported when a peer supplies a copyrighted file to an illegal requestor. Courtesy of Lou and Hwang [31]

A new PAP was developed to distinguish pirates from legitimate clients. Detected pirates will receive poisoned chunks in their repeated attempts. Pirates are thus severely penalized with no chance to download successfully in a tolerable amount of time. Based on simulation results, a 99.9 percent prevention rate was found in Gnutella, KaZaA, and Freenet, while for eMule, eDonkey, Morpheus, and others the rate is between 85 percent and 98 percent. This prevention system is shown to be less effective in protecting poison-resilient networks such as BitTorrent and Azureus. However, the poisoning approach opens up low-cost P2P technology for copyrighted content delivery. The advantage lies mainly in minimum delivery speed, higher content availability, and copyright compliance (see Figure 8.36).



**FIGURE 8.36** The protected peer joining process for copyrighted P2P content delivery using seven messages to secure communications among four parties involved. Courtesy of Lou and Hwang [31]

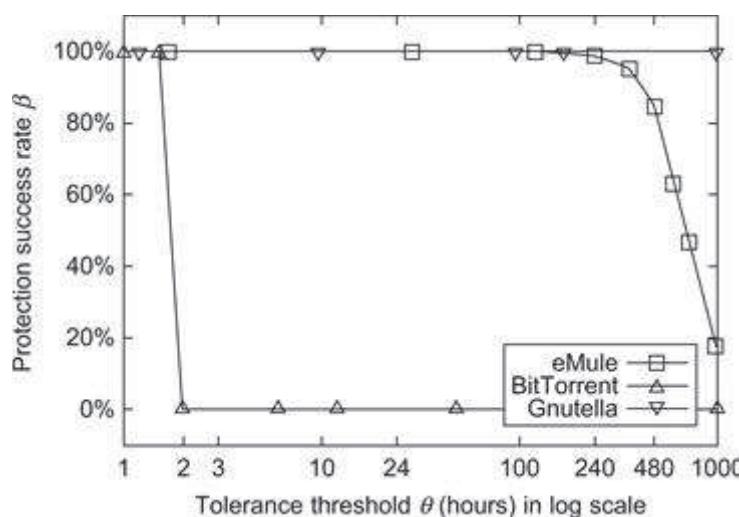
#### 8.5.4.2 Pirate Download Time and Success Rate

To explore the limit of the trusted P2P system, various file sizes have been experimented. The *protection success rate*  $\beta$  is defined by the failure rate of pirates to download a file within a *tolerance threshold*  $\theta$  of time. The  $\theta$  is set at 20 days for a 700 MB CD-ROM image file and 30 days for a 4.5 GB movie file.

Figure 8.36 simulates all three P2P network families with 100 paid clients, 900 colluders, and 1,000 pirates. This scenario of 90 percent colluders is very unlikely in a real-life P2P network. This experiment is designed to approximate a worst-case scenario that all pirates adopt an aggressive peer selection policy. The percentage of pirates that fail to download a clean copy within the time frame  $\theta$  as an approximated success rate  $\beta$  is measured. All  $\beta$  curves start with 100 percent initially.

Without poisoning, the average download time for a client is 1.5 hours. The Gnutella family, including KaZaA and LimeWire, has a near-perfect success rate (higher than 99.9 percent). The eMule network has an average of 85 percent after tolerating up to 20 days. These success rates are satisfactory, because most pirates will give up trying after a few days without success. The Gnutella family including the KaZaA and LimeWire network has the highest success rate, close to 99.9 percent, for both file sizes.

With a tolerance threshold of 20 days in Figure 8.37, the eMule network has an average 98 percent success rate for the 4.5 GB file and 85 percent success rate for the 700 MB file. In comparison, an average pirate in a BitTorrent network takes about 100 minutes to download the 700 MB file, only marginally longer than a paid client. Hence the success rate drops rapidly before two hours. This implies that our system does not protect BitTorrent well due to its strong resistance to content poisoning.



**FIGURE 8.37** Protection success rate of a simulated eMule network by prolonging the delivery delay to an intolerable magnitude. Courtesy of Lou and Hwang [31]

## 8.6 Bibliographic Notes and Homework Problems

A good tutorial on P2P systems was given by Ross and Rubenstein [38] at IEEE Infocom 2004. The key concepts of P2P are introduced in Wikipedia [50]. Buford, et al. have treated

P2P networking and applications in a 2008 book [6]. The problem of building overlays for unstructured P2P networks has been studied in [13,28,52]. Enabling techniques for P2P computing are discussed in [5]. NetTube [8] leverages the high clustering property of videos and P2P techniques for improving YouTube services. Keromytis, et al. [21] reported the SOS architecture and operational requirements. The Skype protocol was analyzed in [3] and measured in [15].

Reported work on consistent hashing can be found in [21]. Chord was first proposed by Stoica, et al. [45] and extended by a large number of researchers [12,16]. CAN was proposed by Ratnasamy, et al. [40]. Pastry was treated by Rowstron and Drusche [39]. Kademlia was proposed in [35] and measured in [44]. Gnutella was studied in [60] and measured in [17], and Freenet in [10]. Hybrid overlays are studied in [32]. Gnutella and Napster were also measured and compared by Sarioiu, et al. in [41]. Network proximity for structured P2P networks was studied in [7,16,54], while for unstructured P2P networks, studies can be found in [28,52]. The small-world networks is studied in Kleinberg [22].

P2P churn resilience and fault tolerance are treated in [12,14,23,24,51]. Trust models for P2P computing have been studied in [19,27,43,49]. Three P2P reputation systems—EigenTrust, PeerTrust, and PowerTrust—are reported in [20,53,57], respectively. P2P file-sharing and content distribution technologies can be found in [1,25,37,31,37,41,46]. Data replication and network proximity are studied in [11,33]. Overlay multicasts were extensively discussed in [9,47,48,56]. P2P video streaming is studied in [8,18,29,36]. BitTorrent has been studied in [4,26,30,59].

Ross and Rubestien [38] have suggested four important research areas for P2P systems: (1) Locality in DHT-based systems, (2) Using DH to hash or not to hash, (3) Building a trusted systems from autonomous and untrusted collections, and (4). Building P2P systems to operate in high-churn environments. These challenges are partially solved in some recent work on churn-resilient load balancing [42], massive data dissemination [24], P2P trust management [43,57,58], and copyright protection [31]. A reputation system for social networking is reported in [55].

## Acknowledgements

This chapter was jointly written by Zhenyu Li of the Chinese Academy of Sciences (CAS) and by Kai Hwang of the University of Southern California. The authors would like to acknowledge the technical support from Gaogang Xie of CAS Institute of Computing Technology.

## References

1. Androulidakis S, Spinellis D. A survey of P2P content distribution technologies. *ACM Comput Surv.* 2004;December.
2. Amaral L, Scala A, Barthelemy M, Stanley M. Classes of small-world networks. *Natl Acad Sci.* 2000;97.

3. Baset SA, Schulzrinne H. An analysis of the Skype peer-to-peer Internet telephony protocol. In: *Proceedings of IEEE INFOCOM*. April 2006.
4. Bindal R, Cao P, Chan W, et al. Improving traffic locality in BitTorrent via biased neighbor selection. In: *Proceedings of IEEE ICDCS*. 2006.
5. Bloom B. Space/time tradeoffs in hash coding with allowable errors. *Commun ACM*. 1970;13(7):422–426.
6. Buford J, Yu H, Lua E. *P2P Networking and Applications* Morgan Kaufmann December 2008; Also [www.p2pna.com](http://www.p2pna.com).
7. Castro M, Druschel P, Hu YC, Rowstron A. Exploiting network proximity in distributed hash tables. In: *Proceedings of the International Workshop on Future Directions in Distributed Computing*. June 2002.
8. Cheng X, Liu J. NetTube: exploring social networks for peer-to-peer short video sharing. In: *Proceedings of IEEE Infocom*. March 2009.
9. Chen S, Shi B, Chen S. ACOM: any-source capacity-constrained overlay multicast in non-DHT P2P networks. In: *IEEE Transactions on Parallel and Distributed Systems*. September 2007;1188–1201.
10. Clarke I, Sandberg O, Wiley B, Hong TW. Freenet: a distributed anonymous information storage and retrieval system. In: *ICSI Workshop on Design Issues in Anonymity and Unobservability*. June 2000.
11. Cohen E, Shenker S. Replication strategies in unstructured peer-to-peer networks. In: *ACM SIGCOMM*. 2002.
12. Fiat A, Saia J, Young M. Making chord robust to Byzantine attacks. In: *Proceedings of the European Symposium on Algorithms (ESA)*. 2005.
13. Ganesh AJ, Kermarrec AM, Massoulié L. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans Computers*. 2003;52(2):139–149.
14. Godfrey PB, Shenker S, Stoica I. Minimizing churn in distributed systems. In: *Proceedings of ACM SIGCOMM*. 2006.
15. Guha S, Daswani N, Jain R. An experimental study of the Skype peer-to-peer VoIP system. In: *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*. February 2006.
16. Gummadi KP, Gummadi R, Gribble SD, et al. The impact of DHT routing geometry on resilience and proximity. In: *Proceedings of ACM SIGCOMM*. 2003.
17. Hefeeda M, Saleh O. Traffic modeling and proportional partial caching for peer-to-peer systems. In: *IEEE/ACM Transactions on Networking*. December 2008;1447–1460.
18. Huang Y, et al. Challenges, design and analysis of a large-scale P2P VOD system. In: *Proceedings of ACM SIGCOMM 2008, Seattle*. August 2008.
19. Hwang K, Li D. Trusted cloud computing with secure resources and data coloring. *IEEE Inte Comput*. 2010;September:14–22.
20. Kamvar S, Schlosser M, Garcia-Molina H. *The Eigentrust algorithm for reputation management in P2P networks*. ACM WWW '03 Hungary: Budapest; 2003.
21. Keromytis A, Misra V, Rubenstein D. SOS: secure overlay services. In: *Proceedings of ANM SIGCOMM'02*. Pittsburg, PA. August 2002.

22. Kleinberg J. The small-world phenomenon: an algorithmic perspective. In: *Proceedings 32nd ACM Symposium on Theory of Computing*. Portland, OR. May 2000.
23. Leitao J, Pereira J, Rodrigues L. Epidemic broadcast trees. In: *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*. October 2007.
24. Z. Li, G. Xie, K. Hwang, Z. Li, Churn-resilient protocol for massive data dissemination in P2P networks, in: *IEEE Transactions on Parallel and Distributed Systems*, accepted to appear 2011.
25. Li Z, Xie G, Li Z. Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems. In: *IEEE Transactions on Parallel and Distributed Systems*. December 2008;1695–1708.
26. Li Z, Xie G. Enhancing content distribution performance of locality-aware BitTorrent systems. In: *Proceedings of IEEE Globecom*. December 2010.
27. Liu L, Shi W. Trust and reputation management. *IEEE Internet Computing* September 2010;10–13 (special issue).
28. Liu Y, Xiao L, Liu X, M Ni L, Zhang X. Location awareness in unstructured peer-to-peer systems. In: *IEEE Transactions on Parallel and Distributed Systems*. February 2005;163–174.
29. Liu J, Rao SG, Li B, Zhang H. Opportunities and challenges of peer-to-peer internet video broadcast. In: *Proceedings of the IEEE Special Issue on Recent Advances in Distributed Multimedia Communications*, Vol 96 (1). 2008;11–24.
30. Locher T, Moor P, Schmid S, Wattenhofer R. Free riding in BitTorrent is cheap. In: *Proceedings of ACM HotNets*. November 2006.
31. Lou X, Hwang K. Collusive piracy prevention in P2P content delivery networks. *IEEE Transactions on Computers*. 2009;Vol. 58(7):970–983.
32. B.T. Loo, R. Huebsch, I. Stoica, J. M. Hellerstein, The Case for a Hybrid P2P Search Infrastructure, 3rd International Workshop on Peer-to-Peer Systems, February 2004.
33. Lv Q, Cao P, Cohen E, Li K, Shenker S. Search and replication in unstructured peer-to-peer networks. In: *Proceedings of the ACM International Conference on Supercomputing*. June 2002.
34. Manimaran B, Bertier M, Kermarrec A-M. Build one, get one free: leveraging the coexistence of multiple P2P overlay networks. In: *Proceedings of the International Conference on Distributed Computing Systems*. Toronto. June 2007.
35. Maymounkov P, Mazières D. Kademlia: a peer-to-peer information system based on the XOR metric. *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)* March 2002.
36. Magharei N, Rejaie R, Guo Y. Mesh or multiple-tree: a comparative study of live P2P streaming approaches. In: *Proceedings of IEEE INFOCOM*. Alaska. May 2007.
37. Pallis G, Vakali A. Insight and perspectives for content delivery networks. *Commun ACM*. 2006;49(1):101–106.
38. Ross K, Rubenstein D. Peer-to-peer systems. In: *IEEE Infocom*. Hong Kong. 2004; (Tutorial slides).
39. Rowstron A, Druschel P. Pastry: scalable, decentralized object location and

- routing for large-scale peer-to-peer systems. In: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. Heidelberg, Germany. November 2001;329–350.
- 40. Ratnasamy S, Francis P, Handley M, Karp R. A scalable content-addressable network. In: *Proceedings of ACM SIGCOMM*. August 2001.
  - 41. Saroiu S, Gummadi P, Gribble S. A measurement study of peer-to-peer file sharing systems. In: *Multimedia Computing and Networking (MMCN '02)*. January 2002.
  - 42. Shen H, Xu C. Locality-aware and churn-resilient load balancing algorithms in structured peer-to-peer networks. In: *IEEE Transactions on Parallel and Distributed Systems*, Vol 18 (6). June 2007;849–862.
  - 43. Song S, Hwang K, Zhou R, Kwok YK. Trusted P2P transactions with fuzzy reputation aggregation. In: *IEEE Internet Computing*. November–December 2005;18–28.
  - 44. M. Steiner, T. En-Najjary, E. Biersack, Long term study of peer behavior in the KAD DHT, in: *IEEE/ACM Transactions on Networking*, Vol. 17 (6) 2009.
  - 45. I. Stoica, R. Morris, D. Liben-Nowell, et al., Chord: a scalable peer-to-peer lookup service for Internet applications, in: *IEEE/ACM Transactions on Networking*, Vol. 11 (1) February 2003, pp. 17–32.
  - 46. Spripanidkulchai K, Maggs B, Zhang H. Efficient content location using interest-based locality in peer-to-peer systems. *Proceedings of IEEE INFOCOM* 2003.
  - 47. Tang C, Chang RN, Ward C. GoCast: gossip-enhanced overlay multicast for fast and dependable group communication. In: *Proceedings of the International Conference on Dependable Systems and Networks*. Yokohama, Japan. June 2005;140–149.
  - 48. Venkataraman V, Yoshida K, Francis P. Chunkyspread: heterogeneous unstructured tree-based peer to peer multicast. In: *14th IEEE International Conference on Network Protocols*. November 2006;2–11.
  - 49. Wang Y, Vassileva J. Trust and reputation model in peer-to-peer networks. In: *Third International Conference on Peer-to-Peer Computing*. August 2003.
  - 50. Wikipedia, Peer to Peer. <http://en.wikipedia.org/wiki/Peer-to-peer>, 2010 (accessed 14.08.2010).
  - 51. Wouhaybi RH, Campbell AT. Phoenix: supporting resilient low-diameter peer-to-peer topologies. In: *IEEE INFOCOM*. 2004.
  - 52. L. Xiao, Y. Liu, L.M. Ni, Improving unstructured peer-to-peer systems by adaptive connection establishment, in: *IEEE Transactions on Computers*, Vol. 54 (9), September 2005, pp. 1091–1103.
  - 53. L. Xiong, L. Liu, Peertrust: supporting reputation-based trust for peer-to-peer electronic communities, in: *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16 (7), 2004, pp. 843–857.
  - 54. Xu Z, Tang C, Zhang Z. Building topology-aware overlays using global soft-state. In: *Proceedings on the International Conference on Distributed Computing Systems*. 2003.

55. Yang M, Dai Y, Li X. Bring reputation system to social network in the maze P2P file-sharing system. In: *IEEE 2006 International Symposium on Collaborative Technologies and Systems (CTS 2006)*. Las Vegas. 14–17 May 2006.
56. Zhang Z, Chen S, Ling Y, Chow R. Capacity-aware multicast algorithms in heterogeneous overlay networks. In: *IEEE Transactions on Parallel and Distributed Systems*. February 2006;135–147.
57. Zhou R, Hwang K. PowerTrust for fast reputation aggregation in peer-to-peer networks. In: *IEEE Transactions on Parallel and Distributed Systems*. April 2007;460–473.
58. Zhou R, Hwang K, Cai M. GossipTrust for fast reputation aggregation in peer-to-peer networks. In: *IEEE Transactions on Knowledge and Data Engineering*. September 2008;1282–1295.
59. Zhou Z, Li Z, Xie G. ACNS: Adaptive complementary neighbor selection in BitTorrent-like applications. In: *Proceedings of IEEE ICC 2009*. Germany. 2009.
60. Y. Zhu, Y. Hu, Enhancing search performance on Gnutella-Like P2P systems, in: *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17 (12), December 2006, pp. 1482–1495.

## Homework Problems

### Problem 8.1

You studied the basic idea of DHT in [Section 8.3.1](#). One important DHT-based protocol is Kademlia [35]. The unique feature of Kademlia is that it has been successfully used in eDonkey/eMule file-sharing systems. Study the Kademlia paper and compare it with Chord and CAN in terms of routing complexity, maintenance cost, and scalability. Finally, download the source code of eMule at [www.emule-project.net](http://www.emule-project.net) and study how it works. If possible, modify the client protocol to measure the eMule system. For measurement, please refer to the paper authored by M. Steiner [44].

### Problem 8.2

P2P VoD (video on demand) systems are different from P2P live streaming systems since P2P-VoD has less synchrony in the users sharing video content, and thus it is very difficult to leverage end users' resources. Please read the paper [18] authored by Huang, et al. Then list the challenges of designing P2P VoD systems and explain how PPlive solves these challenges.

### Problem 8.3

List the three types of approaches to exploit network proximity in structured P2P overlays and their advantages and disadvantages. [Section 8.3.3](#) points out that, in Chord overlay networks, a node  $x$  can take the physically closest node whose identifier falls in the range as its  $i$ -th figure entry, which does not affect routing complexity. Why does this

hold?

## Problem 8.4

Building network proximity-aware overlays would degrade the performance of search algorithms such as flood-based algorithms and random walk-based algorithms since nodes are clustered into groups and fewer links are built among the groups. For example, a random walk query message may be forwarded within a cluster for many hops before reaching the cluster where the resource is located. You are asked to design a search algorithm for clustered overlay networks.

## Problem 8.5

Data delivery can also be implemented on top of a DHT-based structure. The representative approach is CAM-Chord [56], which builds a capacity-aware Chord overlay and delivers data items on top of it. CAM-Chord brings no redundant message, that is, to deliver an item to  $n$  nodes, only  $n$  message is required. However, maintaining such a structure is nontrivial. Read the paper and compare CAM-Chord with the ACOM [9] and CRP approaches [24]. Please focus on following performance metrics: hop complexity, proximity awareness, overlay maintenance overhead, and data replication ratio.

## Problem 8.6

In Table 8.9, compare the routing algorithms of Chord, Pastry, and CAN by briefly filling in the missing table entries. Let  $n$  be the network size (number of peer nodes) and  $d$  be the dimension size. A few of the entries are already given as examples.

**Table 8.9**

**Comparison of Structured P2P Networks**

Features	Chord	Pastry	CAN	Kademlia
Route state			$O(d)$	
Network structure	Chordal ring			
Routing complexity		$O(\log n)$		
Fault tolerance				

## Problem 8.7

P2P traffic locality can also be improved by deploying cache at the gateway of the network provider. However, different from the HTTP access pattern, the popularity distribution of objects (files) does not follow Zipf law. Read the paper [17] and explain how the popularity distribution affects the cache algorithm. Furthermore, discuss the

advantages and shortcomings of different methods to preserve locality.

## Problem 8.8

In Section 8.5.1, you studied the basic search algorithms that do not consider semantic information of shared content. It has been identified in [60] that building semantic overlays can significantly improve search performance. Read this paper and their related works. You are asked to describe the reasons why semantic overlays can enhance search performance.

## Problem 8.9

YouTube is one of the largest web sites for user-generated videos. Tens of thousands of videos are viewed each day by a large number of users. YouTube naturally leverages P2P techniques to alleviate server loading. However, unlike traditional videos, user-generated videos are small. Study the NetTube paper [8] and discuss the solution for using P2P techniques in YouTube content distribution.

## Problem 8.10

Analyze the fault tolerance of a Chord overlay with respect to the size of the successor list. Is it sufficient to guarantee a stable Chord ring with high probability in the case that the successor list size is  $O(\log n)$  ( $n$  is the number of nodes), when every peer fails with probability  $\frac{1}{2}$ ?

## Problem 8.11

BitTorrent is wildly used for file sharing. Files are divided into pieces. Nodes exchange the piece availability information with their neighbors. When a node finds there is a piece that its neighbor has but it has not downloaded yet, it sends a request for the piece to that neighbor. By default, the piece size is 256 KB. Perform simulations using the code in <http://theory.stanford.edu/~cao/biased-bt.html> to evaluate how piece size affects content distribution performance for files of various sizes.

## Problem 8.12

Skype is a P2P-based VoIP system. It has been widely used and studied. Read the papers on Skype [3]. Then describe the overlay structure of Skype, the bandwidth consumption characteristic, and the impact of access capacity constraint on VoIP quality.

## Problem 8.13

In mesh-based pull methods, each node advertises to its neighbors which data block it has received. The neighbors explicitly request messages if needed. Tree-based push

methods, on the other hand, deliver data blocks along one or multiple tree structures. It is considered that mesh-based methods suffer from a trade-off between control overhead and delay, while tree-based methods have a trade-off between continuity and delay [48]. Read a survey paper on content delivery algorithms [36] and a comparison study on mesh- and tree-based methods [29]. You may consider combining these two methods by leveraging network coding in tree-based systems.

## Problem 8.14

BitTorrent systems use the so-called “Tit-for-Tat” incentive scheme to enable contribution awareness in content delivery. Some researchers point out that this mechanism is sufficient to ensure fairness, but others are doubtful of this and free-riding is still cheap. Study these three papers in [20,29,37]. Then answer the following two questions: (1) Which factor makes free riding so easy? (2) Do you think “Tit-for-Tat” is sufficient to solve the problem? Justify your answers with reasoning.

## Problem 8.15

Match the overlay networks on the left with their best-match descriptions on the right:

<input type="checkbox"/> eMule	a) A structured overlay using a coordinated multi-dimensional search space
<input type="checkbox"/> BitTorrent	b) An unstructured overlay for anonymous and decentralized storage services
<input type="checkbox"/> Gnutella	c) A P2P network for video telephony applications
<input type="checkbox"/> Napster	d) An overlay coded with high-radix nested groups for proximity routing
<input type="checkbox"/> SETI@Home	e) A P2P file-sharing network using multiple file index trackers
<input type="checkbox"/> KaZaA	f) The first P2P network for MP3 music delivery using a centralized control
<input type="checkbox"/> Chord	g) A secure overlay service for preventing DDoS or network attacks
<input type="checkbox"/> Pastry	h) An unstructured overlay using flooding search for file providers
<input type="checkbox"/> Freenet	i) A decentralized P2P network built with multiple super nodes
<input type="checkbox"/> CAN	j) A P2P file-sharing network applying hashing at a 9.5 MB part level
<input type="checkbox"/> Skype	k) A DHT-based overlay structure with a $O(\log n)$ lookup time
<input type="checkbox"/> SOS	l) A P2P network for distributed processing of outer space signals

## Problem 8.16

Consider a Chord overlay network with 64 keys identified by 6-bit key identifiers. Assume six nodes (0, 4, 7, 12, 32, 50) are populated at present. The rest are not assigned yet. Answer the following questions using this Chord graph:

- What are the data keys and node keys stored in node 0?
- Work out the finger table for node 7. Describe how node 7 uses the finger table to find the shortest routing path to reach the home node of a document file where data key 3 is located.
- Describe how to insert a new node 45 into this Chord structure. You need to show the

shortest path on the Chord graph leading to the node holding key 45.

- d. Use the finger table at the predecessor node to perform five lookup operations to establish the finger table for this new node 45.

# CHAPTER 9

# Ubiquitous Clouds and the Internet of Things

## CHAPTER OUTLINE

### Summary

#### 9.1 Cloud Trends in Supporting Ubiquitous Computing

- 9.1.1 Use of Clouds for HPC/HTC and Ubiquitous Computing
- 9.1.2 Large-Scale Private Clouds at NASA and CERN
- 9.1.3 Cloud Mashups for Agility and Scalability
- 9.1.4 Cloudlets for Mobile Cloud Computing

#### 9.2 Performance of Distributed Systems and the Cloud

- 9.2.1 Review of Science and Research Clouds
- 9.2.2 Data-Intensive Scalable Computing (DISC)
- 9.2.3 Performance Metrics for HPC/HTC Systems
- 9.2.4 Quality of Service in Cloud Computing
- 9.2.5 Benchmarking MPI, Azure, EC2, MapReduce, and Hadoop

#### 9.3 Enabling Technologies for the Internet of Things

- 9.3.1 The Internet of Things for Ubiquitous Computing
- 9.3.2 Radio-Frequency Identification (RFID)
- 9.3.3 Sensor Networks and ZigBee Technology
- 9.3.4 Global Positioning System (GPS)

#### 9.4 Innovative Applications of the Internet of Things

- 9.4.1 Applications of the Internet of Things
- 9.4.2 Retailing and Supply-Chain Management
- 9.4.3 Smart Power Grid and Smart Buildings
- 9.4.4 Cyber-Physical System (CPS)

#### 9.5 Online Social and Professional Networking

- 9.5.1 Online Social Networking Characteristics
- 9.5.2 Graph-Theoretic Analysis of Social Networks
- 9.5.3 Communities and Applications of Social Networks
- 9.5.4 Facebook: The World's Largest Social Network
- 9.5.5 Twitter for Microblogging, News, and Alert Services

## 9.6 Bibliographic Notes and Homework Problems

### Acknowledgements

### References

### Homework Problems

Data-intensive applications and future trends of Internet clouds are discussed along the lines of supporting mobile computing, ubiquitous computing, and social networking. First, public, private, and science clouds and grid projects are reviewed, including those built by IBM, SGI, [Salesforce.com](#), NASA, and CERN. Requirements for data-intensive scalable computing are reviewed. Then performance metrics are presented for high-performance computing grids and for high-throughput computing clouds. Cloud mashups, mobility support, and benchmark results on existing grids and clouds are reported. The emergence of the Internet of Things and cyber-physical systems is assessed. Finally, we analyze large-scale social networks. In particular, Facebook and Twitter are reviewed with innovative applications.

ubiquitous computing, Internet of Things (IoT), cloud mashups, data-intensive scalable computing (DISC), radio-frequency identification (RFID), social networking, cyber-physical systems

## Summary

Data-intensive applications and future trends of Internet clouds are discussed along the lines of supporting mobile computing, ubiquitous computing, and social networking. First, public, private, and science clouds and grid projects are reviewed, including those built by IBM, SGI, [Salesforce.com](#), NASA, and CERN. Requirements for data-intensive scalable computing are reviewed. Then performance metrics are presented for high-performance computing grids and for high-throughput computing clouds. Cloud mashups, mobility support, and benchmark results on existing grids and clouds are reported. The emergence of the Internet of Things and cyber-physical systems is assessed. Finally, we analyze large-scale social networks. In particular, Facebook and Twitter are reviewed with innovative applications.

## 9.1 Cloud Trends in Supporting Ubiquitous Computing

This chapter looks into the future of distributed and cloud computing systems and their innovative applications. Distributed computing has changed rapidly in system structure, scale of operation, and application domains in recent years. Clustered systems appeared first in the 1990s, which was evidenced by the increasing population of cluster-structured massively parallel processors (MPPs) in the Top 500 supercomputer list. When we crossed into the 2000s grids became popular, appearing as large-scale clusters of computer clusters. However, the general public has not embraced grids due to their lack

of flexibility in general-purpose applications.

Since 2005, many peer-to-peer (P2P) networks have appeared as file-sharing networks and online social networks. In fact, e-books, video streaming, movie downloads, and social networking largely dominate Internet traffic today. Computing clouds over data-center resources did not appear until 2007. Clouds are still in their infancy stage, even though all the underlying technologies are there. However, clouds do reflect the latest development trend that emphasizes elasticity and flexibility by scaling out of computing services to automated data centers through virtualization.

Data-intensive applications and future trends of Internet clouds are discussed in this section, along the lines of supporting mobile computing, ubiquitous computing, and social networking. We assess the development trends of both public and private clouds, as they appear in many high-performance or ubiquitous applications. The idea of cloud mashups is studied for agility and scalability. Future trends in clouds for both business and scientific applications are examined with illustrated examples.

We will evaluate the performance of clouds and distributed systems in [Section 9.2](#). We will devote Sections 3 and 4 to the emerging Internet of Things (IoT) that naturally extends the current Internet, as we are in transition from IPv4 (with a 32-bit address space) to IPv6 (with a 128-bit address space). The IoT may lead to a new wave of IT advances and its application potential may affect all walks of life and the global economy. Finally, we will study social networks and assess their impact on the future of a digital society that is wide open to billions of Internet of IoT users.

## 9.1.1 Use of Clouds for HPC/HTC and Ubiquitous Computing

Ubiquitous cloud computing refers to the use of Internet resources at any place and any time for any objectives. Today, people can access the Internet via a fixed wire or a mobile wireless connection. Science in the cloud has been impacted with a new research environment that leverages software and services being provided on user demand. New advances in research discovery are expected in high-energy physics, observatory astronomy, atmospheric modeling, and biomedicine. Dan Reed [69] of Microsoft Research has assessed the revolution of today's big-data, many-cores pervasive clouds and their impact on scientific research in the information age. Some observations by Reed are discussed below.

### 9.1.1.1 Herbert Simon on Information Consumption

Recall the statement by Herbert Simon: "What information consumes is rather obvious: it consumes the attention of its recipients. Hence a wealth of information creates a poverty of attention, and a need to allocate that attention efficiently among the overabundance of information sources that might consume it." This clearly outlines a supply and demand model in the information age. With ubiquitous sensors, clouds, and mobile devices, we can retrieve information faster and more effectively. We can consume information with much less cost in current and future data-explosive information age.

### **9.1.1.2 Vannevar Bush on Predicting Memex**

Another interesting insight was made by Vannevar Bush on the concept of “Memex.” He made his prediction in 1945, long before the creation of today’s digital computers. He predicted a dream device as follows: “*Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and to coin one at random, “memex” will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.*” What a coincidence ! Today, this “memex” device goes by the name of iPad, smartphone, cloud, IoT, virtual reality, and *cyber-physical system* (CPS), among many others.

### **9.1.1.3 Cloud Service Trends**

Table 9.1 summarizes cloud offerings from eight IT leaders and some startup cloud companies. We already studied the cloud offerings from Amazon, Google, Microsoft, GoGrid, and Rackspace in Chapters 4 and 6. Here we examine the remaining cloud service offerings from IBM, SGI, and [Salesforce.com](#). Other clouds are left for the readers to explore deeper.

**Table 9.1**

**Some IT Companies Providing Cloud Services or Products**

Company, Year	Major Cloud Offerings	Major User Groups
<b>Amazon</b> Seattle, 1994	Amazon Web Services (AWS); six Infrastructure-as-a-Service (IaaS) systems including EC2 for computing capacity and S3 for on-demand storage capacity	10,000 businesses and individual users including the <i>New York Times</i> , <i>The Washington Post</i> , and Eli Lilly
<b>Enomaly</b> Toronto, 2004	Elastic Computing Platform integrating enterprise data centers with commercial cloud offerings, managing internal and external resources with virtual machine (VM) migration	Business Objects, France Telecom, NBC, Deutsche Bank, Best Buy
<b>Google</b> Mountain View, CA, 1998	GAE offering Platform-as-a-Service (PaaS) capability plus office productivity tools including Gmail, calendar tools, the Postini web site creation tool, and security protection services	Small businesses, enterprises, and colleges including Arizona State University and Northwestern University
<b>GoGrid</b> San Francisco, 2008	Offers web-based storage and deploys Windows- and Linux-based virtual servers in the cloud with preinstalled software from Apache, PHP, and Microsoft SQL	Mostly startups, Web 2.0, and Software-as-a-Service (SaaS) companies, plus a few big names like SAP and Novell
<b>Microsoft</b> Seattle, 1975	Azure offering a Windows-as-a-service platform consisting of the OS and developer services that can be used to build and enhance web-hosted applications	Epicor, S3Edge, and Micro Focus using Azure to develop cloud applications
<b>NetSuite</b> San Mateo, CA, 1998	A business software suite including e-commerce, consumer relationship management (CRM), accounting, and enterprise resources planning (ERP) tools	Business customers such as Puck Coffee and Wrigleyville Sports
<b>Rackspace</b> San Antonio, TX, 1998	Mosso cloud offering a platform for building web sites; cloud files for a storage service; and Cloud Servers, an EC2-like service that provides access to virtualized server instances	Web developers and SaaS providers such as Zapproved, which uses Mosso to deliver an online productivity tool
<b>Salesforce.com</b> San	Offers CRM tools for sales force automation, analytics, marketing, and social networking; <a href="#">Force.com</a> offers a PaaS for building web	500,000 customers in financial services, communications, and media, energy, and

### 9.1.1.4 IBM Cloud Projects

IBM has gradually changed from a hardware/software and system company to a computing services company. The IBM cloud platforms are mostly built with IBM server clusters supported by IBM WebSphere and accumulated software assets over the years. Both the z-series and p-series servers at IBM are being upgraded with virtualization capabilities. In 2007, IBM began launching a number of cloud-related R&D projects. These projects are extended from IBM's previous work for on-demand and autonomous computing using service-oriented architecture (SOA).

IBM *Ensembles* offer a virtualized cloud system for IaaS services. This system can put together a large resource pool to simplify management complexity. The purpose is to offer application flexibility and efficient resource deployment through dynamic server, storage, and network ensembles. IBM also developed the *Tivoli Service Automation Manager* (TSAM) for rapid design, deployment, and management of service processes. *WebSphere CloudBurst* (WCA) is another platform for managing private clouds. IBM *LotusLive* offers an SaaS cloud for application service development. The services include an online conference service, coordinated office management, and e-mail services.

### 9.1.1.5 IBM RC2 Cloud

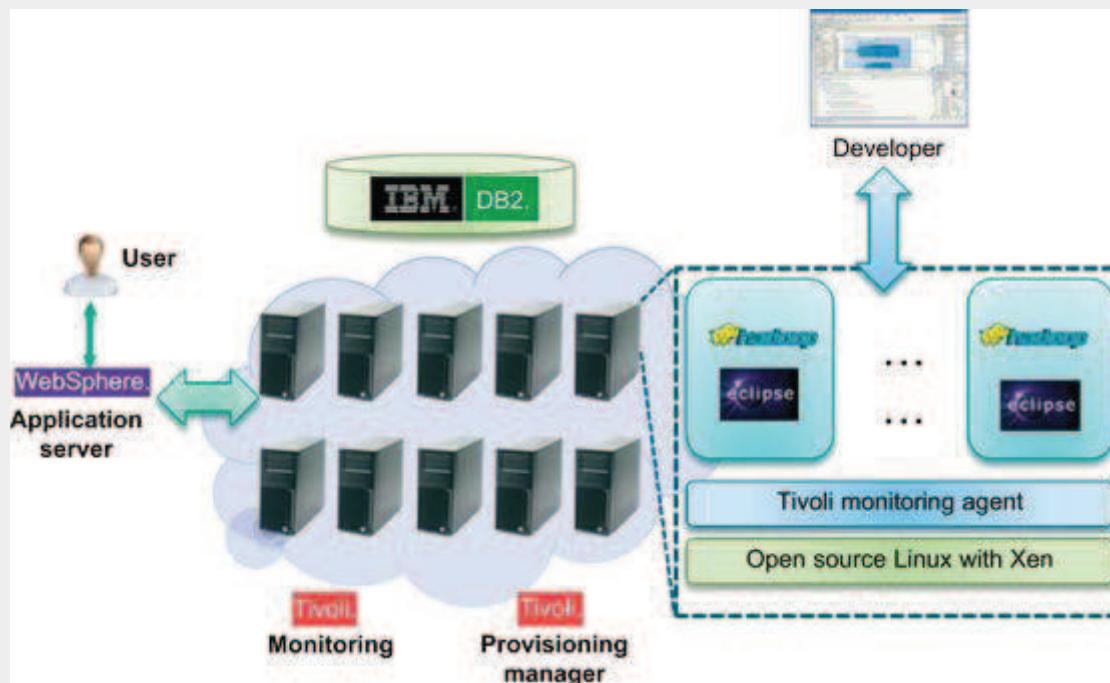
The computer and IT infrastructures of eight IBM Research Centers are now strongly connected to form a private cloud, called the Research Compute Cloud (RC2). RC2 is web-based, allowing more than 3,000 IBM researchers worldwide to share computing resources. It is also IBM's internal testbed for promotion of cloud technology. RC2 provides solutions to establish an autonomous computing environment on user demand. At the time of this writing, the RC2 cloud serves more than 3,000 IBM researchers and developers worldwide as a large testbed for private cloud services.

The system supports machine virtualization, service life-cycle management, and performance monitoring. The idea is to reduce computing costs by outsourcing jobs to best-fit sites for rapid resource deployment, and thus optimal execution. The project also evaluates green energy savings and dynamic resource optimization. In a typical RC2 service setting, the user uses the web service to submit a new order of RC2 resources such as VMs and storage. Then the user specifies the order with special service-level agreements (SLAs) regarding resource quantity and capacity classes. RC2 management approves the order with provisioned resources and services, and RC2 monitors the patterns of requested resources and handles billing at the completion of the service.

#### Example 9.1

#### IBM Blue Cloud System

In November 2007, IBM announced the Blue Cloud Project based on open standards and open source software. The project is supported by more than 200 IBM web-scale researchers worldwide. Blue Cloud combines several existing software and virtualization packages on a specifically designed IBM hardware server platform. Open source and private software is combined to form the cloud computing environment. The Blue Cloud is built with x-servers, that are very similar to x86 processors. (Figure 9.1). Linux runs on these servers supported by XEB-based virtualization software.



**FIGURE 9.1** Architecture of the BlueCloud system built with IBM hardware and software along with some open source Linux and Xen. Courtesy of IBM BlueCloud Project

Special software includes Tivoli for monitoring of virtual cluster operations (see [www-01.ibm.com/software/tivoli/products/monitor](http://www-01.ibm.com/software/tivoli/products/monitor) for details). DB2 manages structured data, and the Tivoli Provisioning Manager provisions virtual servers to physical nodes (see [www-03.ibm.com/systems/power/software/virtualization/index.html](http://www-03.ibm.com/systems/power/software/virtualization/index.html) for details). The WebSphere Application Server is used for building application portals. Open source software is used in the background for building the cloud application to process large-scale data. Xen is used as the container of running instances of cloud applications. Hadoop is used to process massive amounts of data based on the MapReduce paradigm.

### 9.1.1.6 Cloud System from SGI

In February 2010, SGI announced the Cyclone, a large-scale on-demand cloud computing

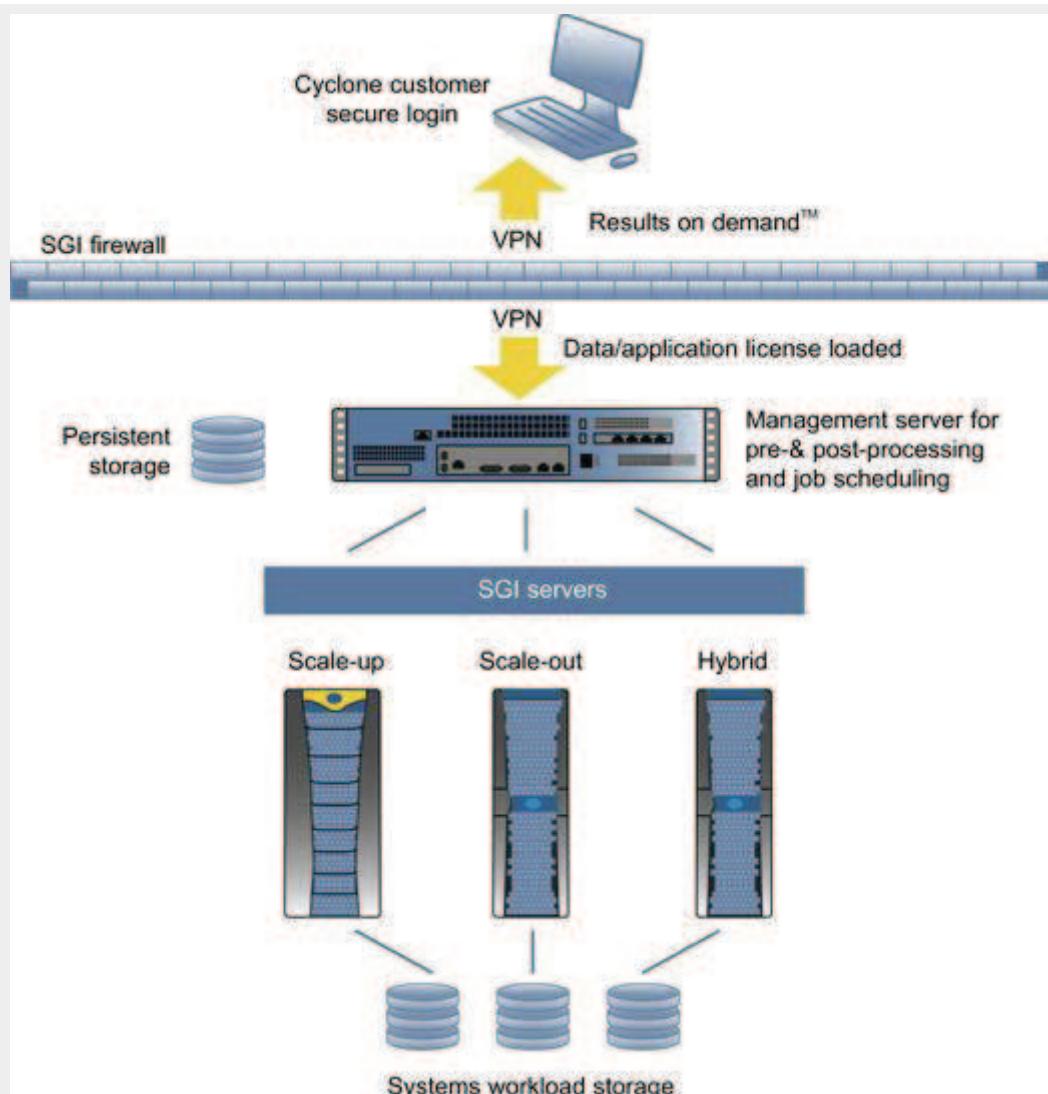
service for high-performance computing (HPC) applications. Both SaaS and IaaS models are offered on the Cyclone. The SaaS model offers users prepackaged technical applications covering a wide range of domains. For example, the system can be used to perform computational fluid dynamics (CFD) and finite element analysis in both the airplane and automobile industries. With the IaaS model, users can access SGI's fastest Altix servers and ICE clusters, which are networking and storage systems managed and optimized by SGI experts for specific user applications.

Companies today deal with increasingly difficult business problems, ever-growing amounts of data to manage and process, and pressure to deliver results in record time. Most attack this critical problem by expanding their IT infrastructure, which often comes with unforeseen administration and facility costs and lower-than-expected throughput due to data bottlenecks. The SGI Altix ICE integrated blade cluster was designed for today's data-intensive problems and avoids these pitfalls. This innovative new platform raises the efficiency bar, easily scaling to meet virtually any processing requirements without compromising ease of use, manageability, or price/performance.

### **Example 9.2**

#### **SGI Cyclone Cloud for High-Performance Computing**

Figure 9.2 shows the architecture of the SGI Cyclone HPC cloud. The SGI servers can scale up, scale out, or combine in a hybrid mode to match specific user application requirements. The Cyclone HPC cloud supports many well-known scientific and engineering applications including OpenFOAM, NUMECA, Acusolve, LS-Dyna, Gaussian, Gamess, NAMD, FASTA, and ClustalW. These cloud applications benefit many industrial and government sectors including green energy, manufacturing, digital media, entertainment, financial government, research, and higher education. Customers can expand their compute and storage capacity with no capital outlay. Advantages are demonstrated by the elimination of lengthy procurement and installation cycles. Users can request infrastructure as they see fit without worrying about infrastructure management. The Cyclone cloud platform is protected by SGI firewalls and virtual private network (VPN) access channels.



**FIGURE 9.2** SGI Cyclone HPC cloud for enabling SaaS and IaaS applications. Courtesy of M. Barrenechea, SGI CEO, [www.sgi.com/cyclone](http://www.sgi.com/cyclone), Feb. 2010

Cyclone is an on-demand cloud computing service specifically dedicated to technical applications. Cyclone supports a number of leading application partners and five technical domains, including computational fluid dynamics, finite element analysis, computational chemistry and materials, computational biology, and ontology. Cyclone is available in two service models: SaaS and IaaS. With the SaaS model, Cyclone customers can significantly reduce time to results by accessing leading-edge open source applications and best-of-breed commercial software platforms from top independent software vendors. The IaaS model enables customers to install and run their own applications.

SGI Altix provides scale-up, Altix ICE provides scale-out, and Altix XE hybrid clusters are used for hybrid management of cloud servers, all based on Intel Xeon or Itanium processors. Altix is a line of servers and supercomputers produced by Silicon Graphics, based on Intel processors. Altix servers have gone through many revisions, the latest being the SGI Altix 4700. This server offers blade configurability in an uncompromising data-centric architecture capable of sustaining terabytes of data throughput. It

features “plug and solve” blade configurability in data-centric architecture capable of sustaining terabytes of data throughput. Like its predecessors, the SGI Altix 4700 platform has been designed for technical users based on industry-standard CPUs, memory, and I/O.

### 9.1.1.7 **Salesforce.com’s Force.com Cloud**

Salesforce.com was established in 1999 to provide online solutions to SaaS, mainly in CRM applications. Initially, it used third-party cloud platforms to promote its software services. Gradually, Salesforce.com launched Force.com as a PaaS platform. By 2010, Salesforce.com subscribers exceeded 2.1 million. Example 9.3 describes the Force.com platform cloud.

#### **Example 9.3**

#### **The Force.com Cloud from Salesforce.com**

Salesforce.com offers its Force.com platform to support some SaaS and IaaS applications. The platform allows external developers to create add-on applications that integrate into the main Salesforce.com hosted applications. It is aimed at enterprise users in business computation applications. In addition to making CRM easier for users to apply in cloud services, Force.com provides Apex, a proprietary Java-like programming language, and the Visualforce to simplify the business development cycle. The AppExchange, is designed to enable multiple users to interact and perform coordinated work easily. Application services are mainly in the areas of CRM databases, application development, and customization.

In the security area, Salesforce.com provides not only some mechanisms to protect data integrity, but also some access control mechanisms to ensure administrative security and record security. The company introduced Chatter as a “Facebook for the enterprise” in June 2010. The packaging service helps users to release their application innovations. Users can customize their CRM applications in the Force.com platform. The system features “Contacts,” “Report,” and “Account” tabs. Users can add sets of customized /novel tabs for specific features in finance and human resources applications. In addition, the Force.com platform also provides SOAP web service APIs. Mobile support is provided for subscribers with BlackBerry, iPhone, and Windows mobile devices. The Salesforce.com SaaS and PaaS services can work in more than a dozen international languages.

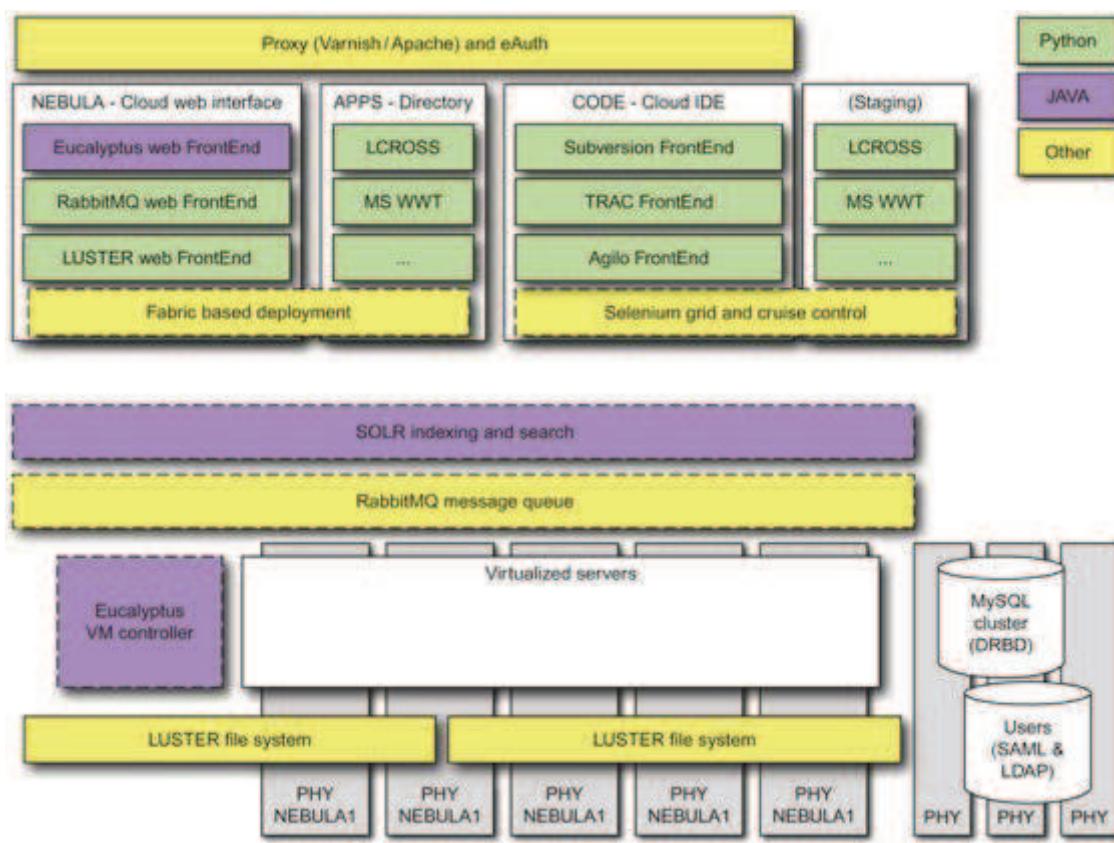
## 9.1.2 Large-Scale Private Clouds at NASA and CERN

In 2010, two large-scale private clouds were under construction in the United States and European Union. We discuss them in this section to demonstrate the scalable growth of cloud computing platforms. The U.S. cloud, called Nebula, is developed by NASA and is designed for NASA scientists to run climate models on remote systems provided by NASA. This can save thousands of NASA users from acquiring supercomputers at their local sites. Furthermore, it enables NASA to build the complex weather models around its data centers, which is more cost-effective.

The EU cloud is built by CERN in Geneva. This is a large private cloud for distributing data, applications, and computing resources to thousands of scientists around the world. CERN deals with large data sets and throughput with a global workforce and a finite budget. Many private clouds are also under development by the IT industry, large enterprises, and other government agencies. Initially, one can see that most clouds are developed with restricted user groups. Once these clouds become mature and more secure to use to prove their claimed advantages, they could be converted to public clouds in the future.

### 9.1.2.1 NASA's Nebula Cloud

Nebula is a cloud computing pilot under development at NASA Ames Research Center ([Figure 9.3](#)). It integrates a set of open source components into a seamless, self-service platform, providing high-capacity computing, storage, and network connectivity using a virtualized, scalable approach to achieve cost and energy efficiencies. Nebula is currently being used for education and public outreach, for collaboration and public input, and for mission support. Nebula enhances NASA's ability to collaborate with external scientists and researchers by providing high-speed data connections and consistent tool sets and open data APIs used by commercial cloud providers. Built from the ground up around principles of transparency, Nebula is also an open source project.



**FIGURE 9.3** The Nebula cloud under development at NASA (<http://nebula.nasa.gov>). Courtesy NASA Ames Research Center

When completed, the Nebula platform will offer a turnkey computing solution that can rapidly address the requirements of a large number of projects. Each component of the Nebula platform will be available individually, serving SaaS, PaaS, and IaaS needs across the agency. NASA will use Nebula for mission support, education, and public outreach and to encourage collaboration and public input. In line with the Obama administration's efforts to establish an open and transparent government, Nebula will provide NASA with an easy, efficient, and secure way to interact and share data with the public.

### 9.1.2.2 Nebula Development Status

Nebula is available to NASA's internal project groups and its research and academic partners. Nebula is not available for use by private industry or the general public. Nebula is also currently acting as a testbed for federal cloud computing technology for the Office of Management and Budget. Nebula uses a containerized data center to house servers on the NASA Ames Research Center campus. These containerized centers are proven to be the “greenest” form of data center because they are denser, and thus require less energy to keep them cool. Cloud computing can be three to five times more efficient than traditional networks or server centers. Another way Nebula is working to make cloud computing more efficient is to actually turn computers on the network off to save energy. Some estimates show Nebula to be 50 percent more energy efficient than traditional data centers.

### 9.1.2.3 CFRN Super Cloud

CERN has long been an influential leader in solving complex IT problems. In fact, the World Wide Web owes a lot to CERN for its usefulness in getting some early work done there. Now, the focus is on cloud computing, CERN scientists plan to smash two particle beams together in an attempt to re-create the “big bang” event to uncover the enigma of many fundamental physics phenomena. The CERN scientists realize the use of cloud computing over their large data centers may be more cost-effective in performing the required simulation experiments.

Tens of thousands of scientists around the world are feeding off the huge data sets and doing all kinds of research by themselves. CERN believes the cloud project will allow it to deliver increased computing performance and offer better infrastructure services to its 10,000 researchers from 85 countries. At CERN, massive amounts of scientific data are processed and must be distributed to researchers in near real time. As a result, CERN’s cloud infrastructure has to provide the capacity necessary to support production and analysis of more than 15 petabytes of data per year, processed by 60,000 CPU cores, allowing scientists to manage workloads themselves as opposed to a centralized IT management department at CERN’s laboratory near Geneva.

Because CERN uses the platform’s Load Sharing Facility (LSF) grid and workload management solution to enable extensive scalability to analyze its vast amount of research data, the laboratory chose to partner again with the platform to explore how to more effectively utilize their resources in a virtualized cloud environment. Platform LSF and its adaptive cluster provide an open, low-cost, common platform for CERN’s scientists, allowing management of both virtual and physical servers in the cloud. In addition, scientists can manage their own application environments and control projects dynamically for maximum flexibility and efficient workload processing.

### **9.1.2.4 CERN VM Projects**

Scientists working on the ALICE (A Large Ion Collider Experiment) collaboration are conducting heavy ion simulations at CERN. They have been developing and debugging compute jobs on a collection of internationally distributed resources, managed by a scheduler called AliEn, to integrate a cloud’s dynamically provisioned resources into an existing infrastructure such as the ALICE pool of computers. The purpose is to ensure that the various AliEn services have the same deployment-specific information. The CERN VM project developed a way to provide VMs that can be used as a base supporting the production environment for all large experiments at CERN.

The CERN VM technology was originally started with the intent to supply portable development environments that scientists could run on their laptops and desktops. A variety of virtual image formats are now supported, including the Xen images used by Amazon EC2 as well as science clouds. The challenge was to find a way to deploy these images so that they would dynamically and securely register with the AliEn scheduler, and thus join the ALICE resource pool.

### **9.1.2.5 Road Map of CERN Cloud Construction**

Nimbus is a cloud OS that allows a user to securely provide context-specific information to a VM deployed on remote resources. It places minimal compatibility requirements on the cloud provider and can orchestrate information exchange across many providers. Commercial cloud providers such as EC2 allow users to deploy groups of unconnected VMs, whereas scientists typically need a ready-to-use cluster whose nodes share a common configuration and security context. The Nimbus Context Broker bridges that gap.

Integration of the Nimbus Context Broker with the CERN VM technology has proved a success. The new system dynamically deploys a VM on the Nimbus cloud at the University of Chicago, which then joins the ALICE computer pool so that jobs can be scheduled on it. Moreover, with the addition of a queue sensor that deploys and terminates VMs based on demand, researchers can experiment with ways to balance the cost of additional resources against their need as evidenced by jobs in a queue.

In 2009, the CERN started an IaaS setup. The batch computing farm is a critical part of the CERN data center today. It has the ability to use the IaaS to provision a large number of virtual batch worker nodes. By doing so, both the VM provisioning system and the batch application itself have been tested on a large scale. Eventually, the cloud at CERN will be able to sustain 15,000 or more concurrent virtual batch worker nodes.

## 9.1.3 Cloud Mashups for Agility and Scalability

In web application development, a mashup is a web page or application that combines data, presentations, or functionality from two or more sources to create a new service. The main characteristics of the mashup are a combination of virtualization and aggregation. In cloud computing, the cloud has captured the computing market with dynamic resource allocation from a pool of VM resources. [Table 9.1](#) listed the diversity of cloud offerings; the resource pool can be employed for both parallel and distributed computing as demonstrated by the Amazon EC2 and S3 services. On the other hand, Google's App Engine (GAE) is primarily driven by web-based service offerings for storage of files, e-mailing, and messaging, among others.

Not only are AWS and GAE different in terms of their functionalities, but they also can complement each other. This has triggered the idea of mashing up different clouds to build an *intercloud* or *cloud of clouds* dynamically. We will demonstrate how to mash up GAE and AWS clouds to achieve the desired features of application agility and performance scalability. In fact, cloud mashups offer a more cost-effective solution to new startups that do not want to invest in cloud hardware and software to create their own enterprise-level data center or private cloud. A cloud mashup must be designed to support popular paradigms such as MapReduce on AWS, and yet be controllable via GAE through easy-to-operate web interfaces.

### 9.1.3.1 The Idea of a Cloud Mashup

Any startup in today's business world needs to keep its operational costs low in the initial years. This allows the business to recover its initial investments and post profits

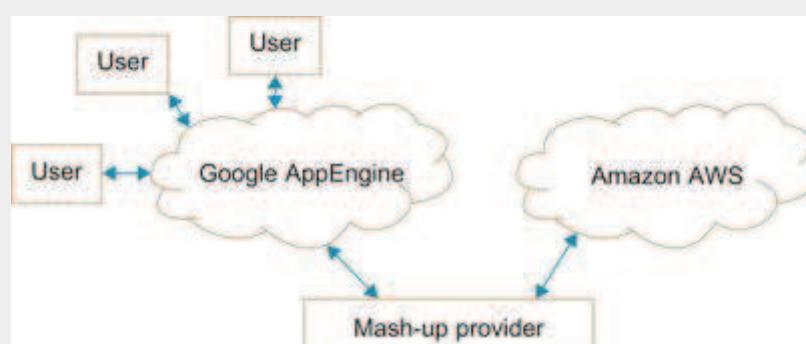
quickly. This, in turn, prompts venture capitalists to fund the company in hopes of a quick return on investment and the ability to supply more capital to allow the company to grow. Thus, it is critical that the company keep its initial costs low. Consider a startup company working as a social networking portal. It needs lots of data storage space and servers, as well as associated cooling requirements and infrastructure such as a physical building to house its equipment. This leads to a moderate to large initial investment on the part of the company.

One can use the pay-per-use models from public clouds such as AWS and GAE to quickly bring the business up to speed. This can not only ensure that the loss due to time-to-market is reduced, but it can also enhance the speed with which one can unleash a new idea into the business world. Business mashups are also popular today as more and more businesses can complement each other nicely. Today's mashups are typically dataflow systems with the interaction scripted in languages such as JavaScript or PHP. The ProgrammableWeb site, for example, reported more than 2,000 APIs and 5,000 mashups in September 2010 [84]. Commercial mashup examples include Yahoo! Pipes (<http://pipes.yahoo.com/pipes/>), Microsoft Popfly ([http://en.wikipedia.org/wiki/Microsoft\\_Popfly](http://en.wikipedia.org/wiki/Microsoft_Popfly)), and IBM's sMash-enhanced BPEL Web 2.0 [156].

#### Example 9.4

### A Mashup Cloud Designed on the AWS and GAE Platforms

Figure 9.4 shows the concept of mashing up the GAE and AWS platforms. Essentially, one gets user input files from GAE and uses the AWS infrastructure for MapReduce. To prove the effectiveness of mashups, the performance of MapReduce on the mashup platform is experimented with scalable EC2 resources. We demonstrate the results of AWS-GAE mashup experiments conducted at the University of Southern California [18]. The mashup provider is crucial to link the two cloud platforms together.



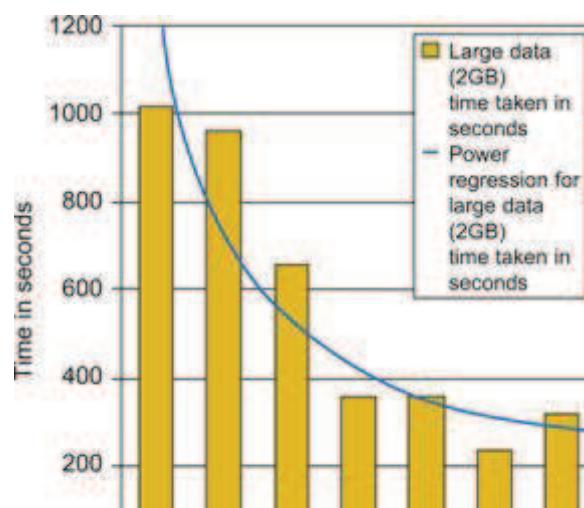
**FIGURE 9.4** The concept of a mashup between the GAE and AWS cloud platforms. Courtesy of V. Dixit of University of Southern California [18]

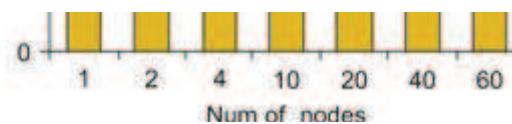
The mashup design leverages Google's web agility with the scaling power of AWS EC2. This mashup helps a user to write agile software on GAE and use the user inputs to perform parallel computing operations on AWS. Also, the creator can apply parallel computing operations to a user-owned cluster. AWS is not very handy as a scalable web interface because the size of EC2 has to be increased by subscriber request. Also, the name server has to be built by the user to isolate the end server from becoming directly visible on the network. On the other hand, Google is more cost-effective in this aspect.

A typical online business such as eBay requires computing power that generates relevant results for its consumers to increase sales. This is beyond what a web-based architecture such as GAE can deliver. So, such a business would need a larger computing "farm," available from Amazon through its AWS offerings. This model allows users to perform a number of computations on a VM. Consider a business running information located on an Excel spreadsheet in a Windows host and computation code written on a UNIX-based system. In this case, GAE, which has no capability to "execute" programs, becomes useless. In fact, any executable code needed by a business cannot use the Google model of service. AWS then becomes the preferred way to accomplish this.

### 9.1.3.2 Mashup Experimental Results

We tested a word counting program written in Python over a large data set of 2 GB. The job was submitted from GAE and was executed on an Amazon EC2 platform. Power regression of data was assumed. The MapReduce scheme was applied for parallel executions on the EC2 cluster of varying numbers of VM nodes from 1 to 60. The results are plotted in [Figure 9.5](#) after a curve fitting exercise. One can clearly see scalable performance with reduced execution time, as EC2 grows from 1 to 60 nodes. It takes 1,000 seconds to process 2 GB of data on one node and the time is reduced to 300 seconds over 60 nodes.





**FIGURE 9.5** Scalable performance on the mashup cloud with decreasing execution time of a 2 GB dataset on EC2 up to 60 VM instances. Courtesy of V. Dixit of University of Southern California [18]

### 9.1.3.3 Advantages of Cloud Mashups

A startup business that offers web-based applications involving large amounts of storage and that depends critically on its web page being up all the time should consider a cloud provider offering a dynamically scaled (load-based scaling) mechanism, as this would lead to increased availability in the face of a sudden increase in demand, which is common to startups. Also, the ability to quickly and automatically scale down allows the business to keep operational costs low. The business computing platform needs to deliver not only scalable services, but also agility to survive and compete. A mashup can combine several platforms or APIs to provide a wider scope of services that allows more flexibility and agility to users:

- **Scalability on the EC2 virtual clusters** Scaling the implementation of MapReduce implementation can be scaled in terms of *problem size* and *virtual cluster size*. Regarding cluster size, the USC experiments run MapReduce from 1 VM node to 40 VM nodes on the EC2 infrastructure. Thus, scaling is well explored on EC2 to yield scalable performance.
- **Agility in using GAE interfaces** A mashup platform is built by combining GAE and AWS, essentially using Google's web interfaces and Amazon's EC2 computing power. We use GAE to create a web page that is capable of uploading documents to GAE and running MapReduce on EC2. The mashup then sends the results back to Google users.

### 9.1.3.4 Lessons Learned about Cloud Mashups

In creating a scalable and agile mashup, it has been demonstrated that the most popular cloud platforms hold significant promise for entrepreneurs and established companies alike. The various synergies explored provide insight into what a company would need to do in order to be more scalable with increasing users and data and what is required to be agile in such an environment. By not being tied down to any single platform or implementation, the idea of agility within a mashup has been demonstrated [20].

SLAs must be established among cloud companies so that a seamless setup can be constructed. The mashup provider has to provide the functionality manually. Lack of security among cloud providers could become a bottleneck for scaling applications beyond a few hundred users. However, this is exactly where the agility of this approach comes into play by allowing the arrangement to be migrated easily across platform boundaries.

Building a private cloud platform may become even more effective to explore the scalability and agility of a mashup system, especially when the number of users or data

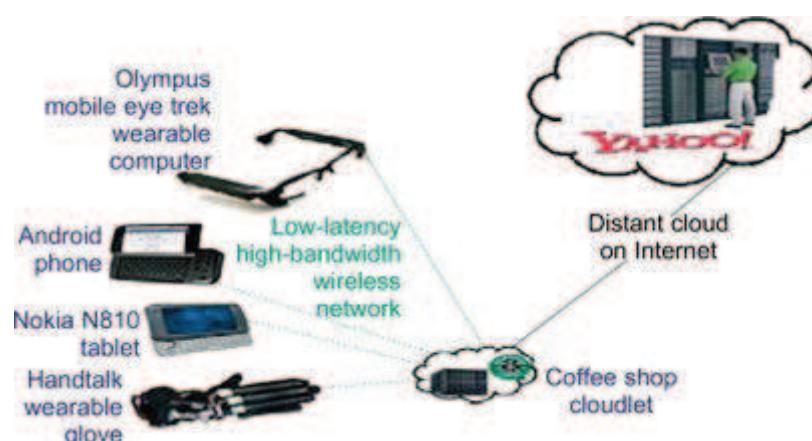
sets grows rapidly. By employing modular data centers, the costs of uploading data across geographically dispersed cloud providers can be alleviated. Wide-range benchmark work is still required to validate the effectiveness of the mashup synergies. MapReduce results demonstrate the feasibility of this approach. The other synergies for different models, such as Dryad and Pig Latin, may produce contradictory results due to differences in computing structure and languages used.

## 9.1.4 Cloudlets for Mobile Cloud Computing

Recently, researchers at Carnegie Mellon University, Microsoft, AT&T, and Lancaster University [74] proposed a low-cost infrastructure to enable cloud computing using mobile devices. The idea is called Cloudlet, and it offers a resource-rich portal for upgrading mobile devices with cognitive abilities to access distant clouds. This portal should be designed to be trustworthy and use VMs to explore location-aware cloud applications. The idea can be applied to opportunity discovery, fast information processing, and intelligent decision making on the road. Cloudlet makes it possible for mobile devices to access the Internet cloud easily in cost-effective mobile computing services.

### 9.1.4.1 The Cloudlet Idea

Figure 9.6 illustrates the idea of using cloudlets for mobile cloud computing. Both mobile devices and centralized clouds or data centers have shortcomings in terms of supporting mobile computing. Mobile handsets face problems of limited CPU power, storage capacity, and network bandwidth on smartphones and tablet computers. Mobile devices cannot be used to handle large data sets. However, the distant cloud on the Internet faces the WAN latency problem. Clouds have to solve this collision problem of too many (millions) clients logging in to the cloud simultaneously.



**FIGURE 9.6** VM-based cloudlets for mobile cloud computing applications. Courtesy of Satyanarayanan, et al. © IEEE [74]

To solve this two-sided problem cloudlets are being deployed at convenient sites, such as coffee shops and bookstores, similar to access points for WiFi services for connecting

to the Internet. Widely deployed cloudlets enable distributed cloud computing and extended resource handling for these users. The idea is to use the cloudlet as a flexible gateway or portal to access the distant cloud. The cloudlet can be implemented on PCs, workstations, or low-cost servers. The major innovation lies in using VM-based flexibility to handle requests from different mobile devices.

### 9.1.4.2 Differences between Cloudlets and Clouds

As Table 9.2 shows, cloudlets are decentralized and self-managed, like a “virtual data center in a box.” Cloudlets operate with LAN latency and bandwidth with only a few users at a time. Clouds are centralized at large-scale data centers. They are professionally administrated all the time and they need large machine rooms with uninterrupted power and cooling. The main problem for a distant cloud to be accessed by mobile devices lies in the long latency experienced with the Internet or WANs. The problem becomes more complicated when thousands or millions of users simultaneously demand services.

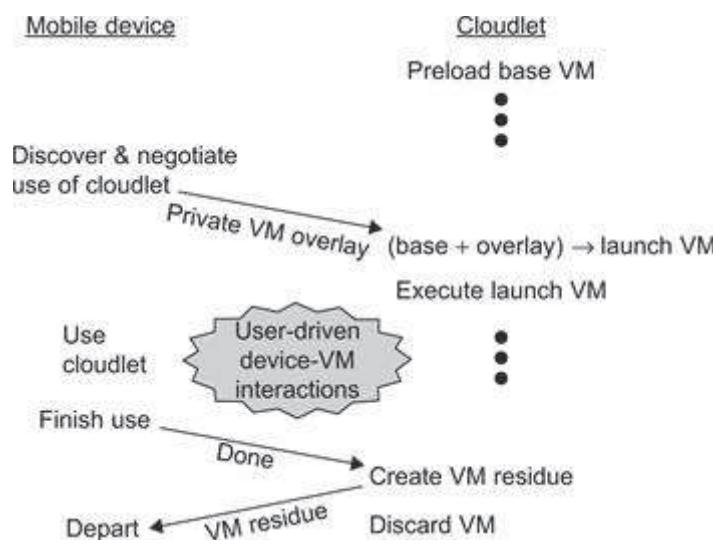
**Table 9.2**

#### Differences between Local Cloudlets and Distant Clouds

	Cloudlet	Cloud
<i>State</i>	Only soft state	Hard and soft state
<i>Management</i>	Self-managed; little to no professional attention	Professionally administered, 24×7 operator
<i>Environment</i>	“Data center in a box” at business premises	Machine room with power conditioning and cooling
<i>Ownership</i>	Decentralized ownership by local business	Centralized ownership by Amazon, Yahoo!, etc.
<i>Network</i>	LAN latency/bandwidth	Internet latency/bandwidth
<i>Sharing</i>	Few users at a time	100s–1000s of users at a time

### 9.1.4.3 Fast VM Synthesis in Cloudlets

A cloudlet prototype named Kimberley was built at Carnegie Mellon University. This prototype synthesizes a VM overlay in the cloudlet host. CMU has reported fast VM synthesis times of less than 100 seconds. In other words, it has created a VM overlay in transient cloudlets that is customized to bind cloud resources at a distance to satisfy user needs. Figure 9.7 shows the timeline for dynamic VM synthesis in Kimberley. A small VM overlay is delivered by a mobile device to the cloudlet that already possesses a base VM. The VM overlay plus the base VM create a special execution environment for the mobile device to launch its cloud applications through the cloudlet portal. Trust and security issues are also major factors in cloudlet deployment.



**FIGURE 9.7** Fast VM synthesis in Kimberley prototype cloudlet built at CMU. Courtesy of Satyanarayanan, et al. © IEEE [74]

#### 9.1.4.4 Mobility and Security Support in IoT Applications

Table 9.3 summarizes the mobility support, data protection, security infrastructure, and trust management in mobile computing and cloud-supported IoT applications. The purpose is to safeguard these cloud computing services in a fixed or mobile distributed computing environment for application listed in Table 9.3. Mobility support includes special air interface and mobile API designs and the use of wireless Public Key Infrastructure (PKI) for mobile access of cloud platforms. VPNs can be also used to secure cloud platforms. Data protection includes file/log access control, data coloring, and copyright compliance. Disaster recovery is also needed in case of hardware/software failures. Cloud security can be enforced by establishing the root of trust, securing the VM provisioning process, using software watermarking, and using firewalls and intrusion detection systems (IDSe) at the host and network levels. Recently, trust overlay networks and reputation systems [43] were suggested to protect data centers in trusted cloud computing.

**Table 9.3**

#### Mobility and Security Support for IoT Applications

IOT Service Level	Mobility Support and Data Protection Methods	Hardware and Software Measures for Cloud Security
Data sensing and networking support	<ul style="list-style-type: none"> <li>Special air interfaces</li> <li>Mobile API design</li> <li>File/log access control</li> <li>Data coloring</li> </ul>	<ul style="list-style-type: none"> <li>Hardware/software root of trust</li> <li>Secure provisioning of VMs</li> <li>Software watermarking</li> <li>Host-based firewalls and IDSe</li> </ul>
Support of cloud platforms for processing sensed data	<ul style="list-style-type: none"> <li>Use of wireless PKI</li> <li>User authentication</li> <li>Copyright protection</li> <li>VPN</li> </ul>	<ul style="list-style-type: none"> <li>Network-based firewalls and IDSe</li> <li>Trust overlay network</li> <li>Reputation systems</li> <li>OS patch management</li> </ul>

## 9.2 Performance of Distributed Systems and the Cloud

In this section, we will study the performance issues of both distributed HPC systems and clouds or data-center-based HTC systems. First, we review the *data-intensive scalable computing (DISC)* paradigm introduced by Randal Bryant [5]. HPC systems are advancing from Pflops performance in current supercomputers to exascale systems in the future. HTC systems are primarily measured by the number of user tasks that can be processed per unit time (i.e., throughput).

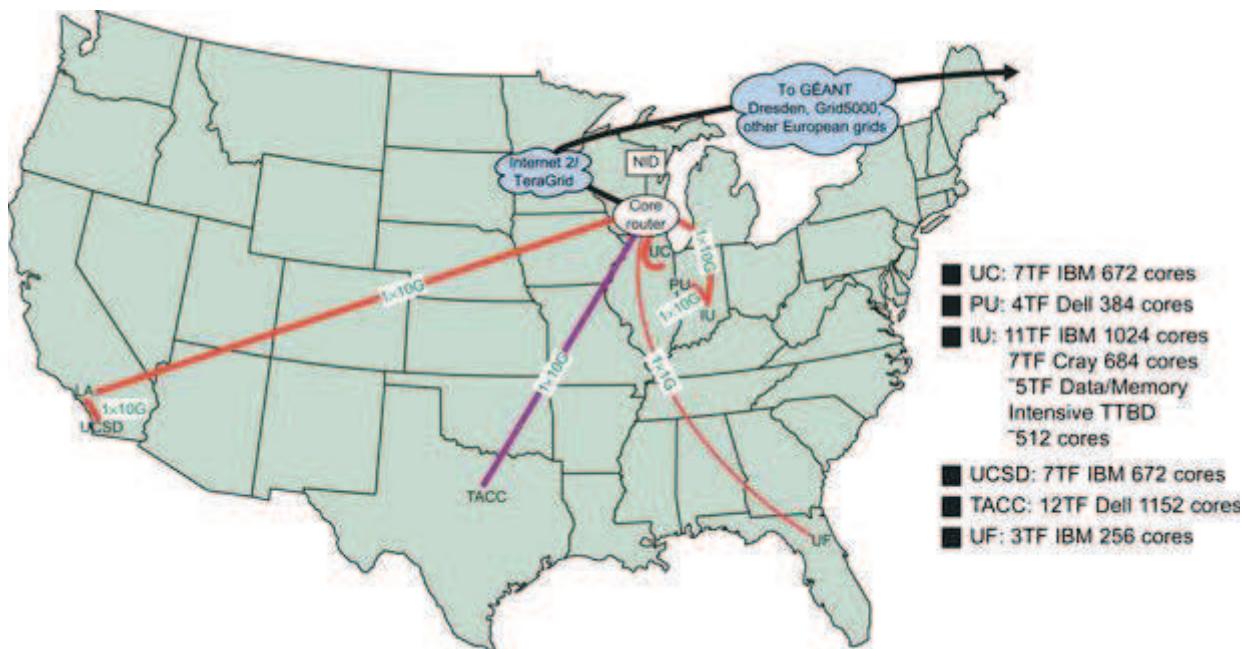
In this section, we identify major performance attributes for evaluating HPC systems. Then we suggest attributes such as sustained speed, system efficiency, and availability for evaluating HPC systems. HTC systems are evaluated in terms of both throughput and Quality of Service (QoS). We study the workload characterization method for performance prediction of grid or cloud systems. Finally, we present some benchmark performance results based on trace experiments conducted at Tsinghua University in China and cloud benchmark experiments at Indiana University.

### 9.2.1 Review of Science and Research Clouds

There has been substantial investigation of clouds for research applications and we surveyed the types of applications that were suitable in [Section 6.2.6](#). We have reviewed some of the research cloud software being developed. Here we focus on infrastructure used for research in and use of cloud computing. Interestingly, although commercial clouds have expanded rapidly, this has not occurred for clouds used for research. These remain smaller than both their commercial counterparts and the large supercomputers found in the TeraGrid introduced in [Chapter 7](#).

#### 9.2.1.1 FutureGrid [30]

FutureGrid, shown in [Figure 9.8](#), is a NSF project funded as an experimental component of TeraGrid and designed around the concept of Grid'5000. FutureGrid is led by Indiana University (hardware, architecture, core software, support) in partnership with Purdue University (HTC hardware), the San Diego Supercomputer Center at the University of California, San Diego (hardware, INCA, monitoring), the University of Chicago/Argonne National Labs (hardware, Nimbus), the University of Florida (hardware, VINE, education, and outreach), the University of Southern California Information Science Institute (Pegasus workflow software to manage experiments), the University of Tennessee, Knoxville (benchmarking), the University of Texas at Austin/Texas Advanced Computing Center (hardware, portal), the University of Virginia (interoperability, grid software), and the Center for Information Services and GWT-TUD from Technische Universität Dresden (VAMPIR software).

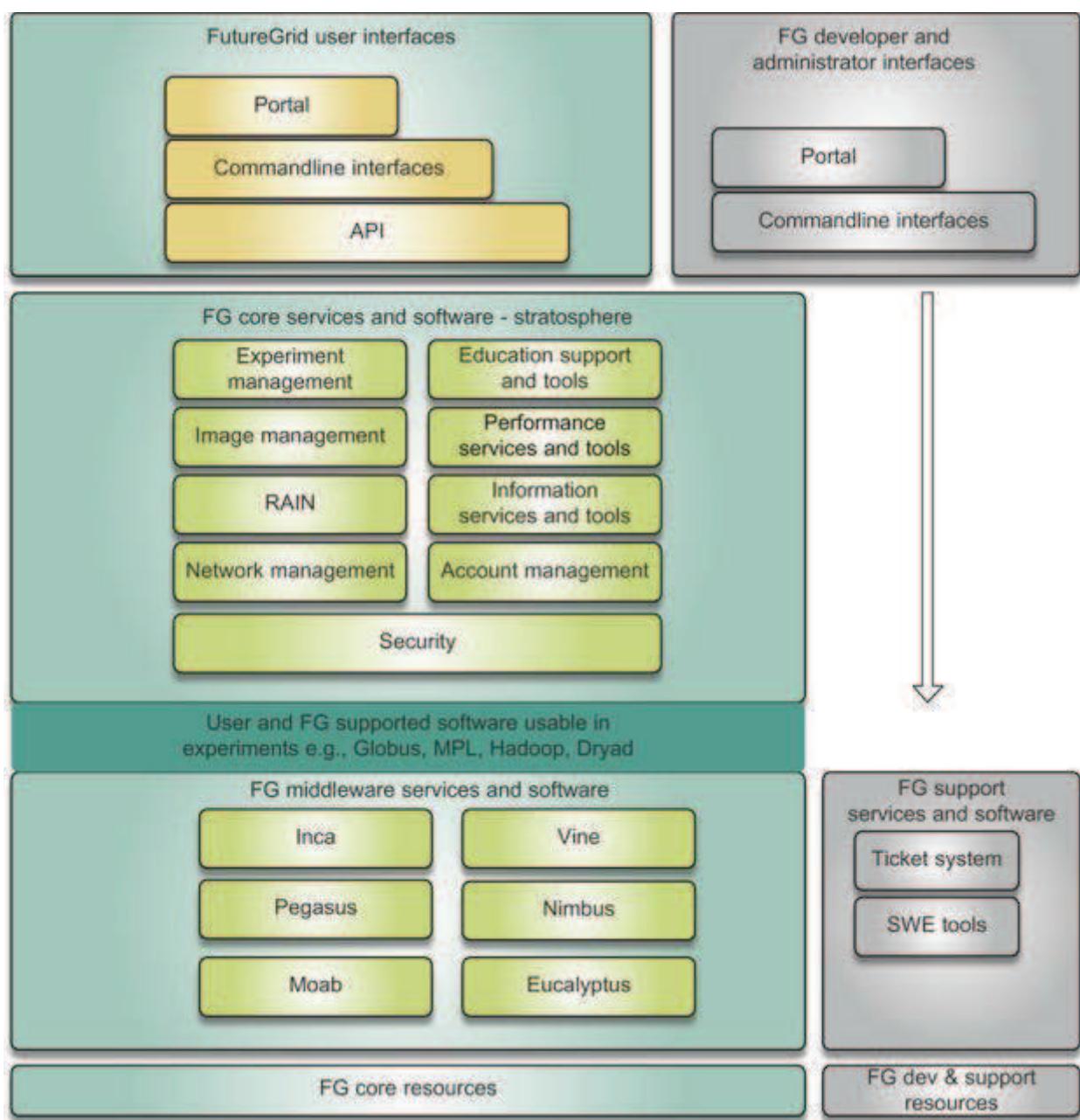


**FIGURE 9.8** FutureGrid distributed system testbed across the U.S. continent [30]. Courtesy of Judy Qiu [30]

Although it is an infrastructure project, it involves significant system development to support its goals. FutureGrid is not a production system, but rather an environment supporting a flexible development and testing platform for middleware and application users looking at interoperability, functionality, and performance issues. FutureGrid will make it possible for researchers to conduct experiments by submitting an experiment plan that is then executed via a sophisticated workflow engine, preserving the provenance and state information necessary to allow reproducibility.

The testbed includes a geographically distributed set of heterogeneous computing systems, a data management system that will hold both metadata and a growing library of software images, and a dedicated network allowing isolatable, secure experiments. The testbed will support VM-based environments, as well as native operating systems for experiments aimed at minimizing overhead and maximizing performance. The project partners are integrating existing open source software packages to create an easy-to-use software environment that supports the instantiation, execution, and recording of grid and cloud computing experiments.

One of the goals of FutureGrid is to understand the behavior and utility of cloud computing approaches. Researchers can measure the overhead of cloud technology by requesting linked experiments on both virtual and bare-metal systems. Further, it offers a rich education and teaching platform for advanced cyberinfrastructure classes. In this regard, it will build tutorials on supported software and offer a suite of appliances (customized VM-based images) supporting online laboratories such as basic grid and parallel computing. Figure 9.9 shows the software stack developed in the FutureGrid project.

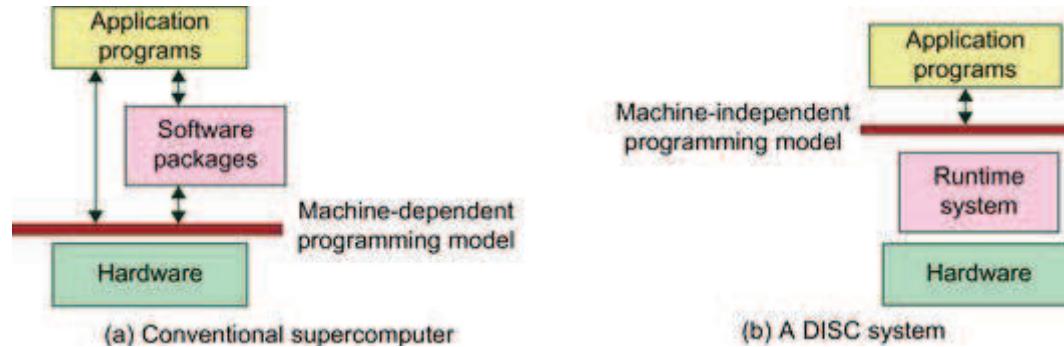


**FIGURE 9.9** The FutureGrid software stack. Courtesy of Judy Qiu [30]

Further, particular classes can be granted time on FutureGrid for development of curricula requiring privileges and system destruction capabilities that are hard to grant on conventional production systems. FutureGrid offers seven distinct systems (clusters) of about 5,000 cores with different optimizations. It also has a dedicated network (except to the Alamo system in Texas) allowing security isolation and use of a dedicated network impairment device. Clouds provide great flexibility by building environments on top of hypervisors. FutureGrid follows a different route by dynamically provisioning software as needed onto “bare metal.”

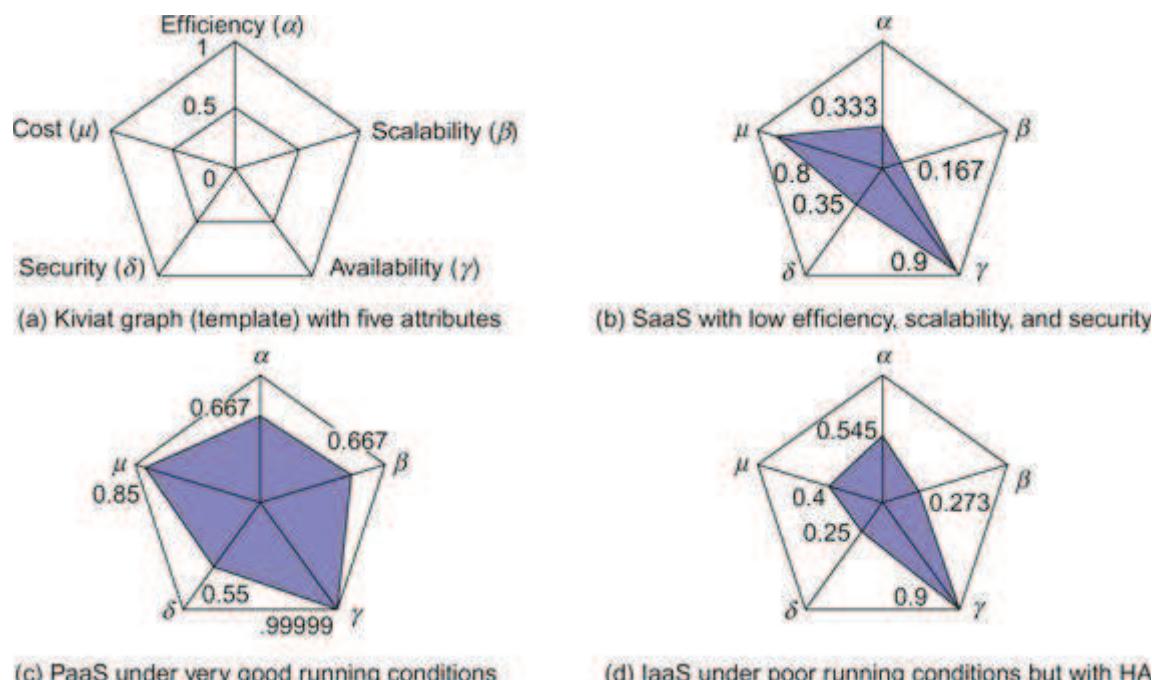
As of November 2010, “rebooting” a node of FutureGrid takes about four minutes [15]. The user can choose from a variety of different images to provision her test environment. Figure 9.9 shows the software stack developed in the FutureGrid project. The images need careful security review and currently include choices at different hypervisor levels (Xen, KVM), operating systems (Linux, Windows), cloud environments (Nimbus, Eucalyptus, OpenNebula), middleware (gLite, Unicore, Globus), programming

paradigms (MPI, OpenMP, Hadoop, Dryad), and specialized capability such as distributed shared memory from ScaleMP.



**FIGURE 9.10** Programmer's perspectives of DISC systems against conventional supercomputers. Courtesy of Randal Bryant [5], Reprinted with Permission

FutureGrid expects available images to increase as users build new and interesting environments and deposit them in the library. Figure 9.9 shows the FutureGrid software stack. Key components include portal interfaces; monitoring capability (INCA, Power [greenIT]); the Experiment Manager, a new feature allowing reproducibility of execution scenarios; image generation and repository feature; VINE intercloud networking (virtual clusters built with virtual networks); a performance library; RAIN or Runtime Adaptable InsertioN Service which schedules and deploys images; and security features including use of isolated network as well as authentication and authorization.



**FIGURE 9.11** Kiviat graphs showing the QoS of three cloud service models under assumed running conditions on five attribute dimensions. Courtesy of K. Hwang and S. Kulkarni, USC 2009

### 9.2.1.2 Grid'5000 [35]

The Grid'5000 distributed system links nine sites in France as shown in [Figure 7.8](#). Recently, Porto Alegre, Brazil, was added as a tenth site. Grid'5000 has more than 5,000 cores spread over 1,500 nodes. It has been used in a variety of computer science research projects including cloud and grid computing and green IT with both software systems and performance goals. Grid'5000 aims at providing a highly reconfigurable, controllable, and monitored experimental platform to its users. Work explores all levels, including application, programming environment, application runtime, grid, cloud, P2P middleware, operating system, and network.

### **9.2.1.3 Magellan [7]**

Magellan is the largest of the research clouds and is situated at two U.S. Department of Energy sites: Argonne [\[6\]](#) and the National Energy Research Scientific Computing Center (NERSC) in California. Each consists of a medium-sized cluster and storage facilities. The NERSC system has more than 700 nodes and about 6,000 cores, while Argonne has more than 500 nodes and 4,000 cores. The systems are addressing a wide variety of research issues, including the following:

- Comparing cloud performance and traditional cluster and supercomputing environments for scientific applications. Of particular interest are applications currently running on local/departmental clusters, or midrange applications that do not require more than 1,000 cores.
- Testing solid state (flash) storage for data-intensive applications. Flash storage offers substantially increased bandwidth and IOPS (I/O operation rate), and decreased latency, especially for read-intensive applications.
- Exploring the portability of applications to cloud systems, and in particular, looking at MapReduce.
- Providing portals on clouds to offer easier access to applications, databases, or automated workflows.
- Testing alternative resourcing models. For example, where availability is crucial, a virtual private cluster could guarantee access to particular research groups for specific periods of time.
- Understanding which scientific applications and user communities are best suited for cloud computing.
- Understanding the deployment and support issues required to build large science clouds. Is it cost-effective and practical to operate science clouds? How could commercial clouds be leveraged?
- Determining how existing cloud software meets the needs of science and whether extending or enhancing current cloud software improves utility.
- Determining how well cloud computing supports data-intensive scientific applications.
- Determining the challenges in addressing security for a virtualized cloud environment.

### **9.2.1.4 Open Cirrus [64]**

This is a broad consortium of some 14 partners exploring cloud computing. The quite powerful testbed consists of more than 600 nodes at four major sites: KIT (Karlsruhe Institute of Technology), the University of Illinois at Urbana Champaign, HP Labs in Palo Alto, California, and the BigData cluster at Intel Research in Pittsburgh. The Open Cirrus testbed is designed to support research into the design, provisioning, and management of services at a global, multidata-center scale.

The open nature of the testbed is designed to encourage research into all aspects of service and data-center management. In addition, Open Cirrus aims to foster a collaborative community around the testbed, providing ways to share tools, lessons, and best practices, and ways to benchmark and compare alternative approaches to service management at data-center scale.

### **9.2.1.5 Open Cloud Testbed [36]**

The Open Cloud Consortium (OCC) is a member-driven organization that supports the development of standards for cloud computing and frameworks for interoperating between clouds, develops benchmarks for cloud computing, and supports reference implementations for cloud computing. The OCC also manages a testbed for cloud computing, the Open Cloud Testbed, and operates cloud computing infrastructure to support scientific research, called Open Science Data Cloud, with an emphasis on data-intensive computing. As of November 2010, the Open Cloud Testbed includes sites at Chicago (two), California (La Jolla, California), and John Hopkins University, with a total of 250 nodes and 1,000 cores.

### **9.2.1.6 Science Clouds [76]**

Science Clouds is an open cloud federation providing compute cycles in the cloud for scientific communities exploring the use of Nimbus. The infrastructure federates sites across Europe and the United States. As of November 2010, it consists of the Nimbus Cloud at the University of Chicago, Stratus at the University of Florida, Wispy at Purdue University, and some machines in the Czech Republic.

### **9.2.1.7 Sky Computing [43]**

This project is aligned with Science Clouds and, like the European Reservoir project, is designed to federate multiple clouds into a single resource. Linking separate clouds together can help scalability (increasing the number of resources), but also improves fault tolerance. The federation also supports the special case of a hybrid cloud—two clouds, one private and one public. Sky Computing has been developed to extend Nimbus (see Section 6.6.1) to support multiple distinct clouds.

One good example of the Sky Computing project is a demonstration that showed the bioinformatics program BLAST controlled by Hadoop running on 1,000 cores in six different sites: three in the United States on FutureGrid (discussed earlier) and three on Grid'5000 [5] in Europe. This distributed federation was enabled by Nimbus for cloud management, offering VM provisioning and contextualization services, from the

University of Florida to enable all-to-all communication among multiple clouds, and Hadoop for parallel fault-tolerant execution of BLAST. Network virtualization is also supported in Reservoir [22].

### 9.2.1.8 Venus-C [79]

Venus-C is a European project exploring the applicability of cloud computing in areas such as building structure analysis, 3D architecture rendering, prediction of marine species population, risk prediction for wildfires, metagenomics, system biology, and drug development. The project explores both Azure (Windows platforms) and Eucalyptus-based infrastructure at two European HPC centers: the Royal Institute of Technology (KTH, Sweden) and the Barcelona Supercomputing Center (BSC, Spain). The project will develop interoperability tools between different clouds for various applications.

## 9.2.2 Data-Intensive Scalable Computing (DISC)

As elaborated in previous chapters, we live in an era of data explosion. In 2007, Randal Bryant was among those who called attention to the increasing demand for data-intensive scalable computing. In this section, we extract some observations he made and compare them to the potential of data-center-based cloud computing.

### 9.2.2.1 Big Data Sources

In today's data-driven world, huge databases are generated from science in astronomy, genomics, natural languages, and seismic modeling, among other applications. In addition, scanned books, newspapers, and historical documents are creating a data ocean. In commerce, corporate sales, stock market transactions, census data, and airline traffic data all demand the handling of large data sets on a daily basis. Other good examples come from the areas of entertainment and medicine. Two sources, according to Bryant, are Wal-Mart and Sloan Digital Sky, as discussed in [Example 9.5](#).

#### Example 9.5

#### Wal-Mart Data Warehouse and Sloan Digital Sky Survey [5]

The first example of big data sources comes from the global operations of Wal-Mart. Wal-Mart has 6,000 stores with supply chains involving more than 10,000 manufacturers. Each day, Wal-Mart sells 267 million items. HP helped Wal-Mart to build data centers to handle 4 petabytes of data daily. Wal-Mart needs to perform DISC for data mining and management of all its supply chains, understanding market trends, and formulating pricing strategies.

The second example comes from the Sloan Digital Sky Survey. Its telescope, in the New Mexico Observatory, captures 200 GB of image data per

day. Its latest data set consists of 10 TB of data on 287 million objects in outer space. Sloan uses SkyServer to provide SQL access in DISC-like data processing continuously.

To build DISC systems to handle the oceans of data in both examples, one must solve the skinny processing pipe problem. Use of many-core GPUs has created a possible solution to exploit massive data-level parallelism in these DISC applications. Cloud computing may be slow compared with supercomputers, but massive parallelism is achieved through the use of large numbers of data centers collectively.

### **9.2.2.2 DISC Systems versus Supercomputers**

**Table 9.4** summarizes four technical differences between DISC systems and conventional supercomputers. In general, supercomputers are essentially used in batch mode and it is difficult to tune a supercomputer due to limited user control. Heavy data movement or I/O activities exist when using supercomputers. It is rather challenging to optimize program execution on a supercomputer. The major advantage of supercomputers lies in their sustained raw speed if everything is put in order.

---

**Table 9.4**  
**DISC Systems versus Supercomputers**

---

Features	Conventional Supercomputers	Data-Intensive Scalable Computing (DISC)
System architecture	An HPC system by which data is retrieved from remote sites and brought into the system for execution. Heavy data movement overhead.	An HTC data-center cluster which collects and maintains data. Computation collocated with storage for faster access. No data movement overhead.
Programming models	Machine-dependent programs written at a very low level. Use fewer software tools; need specialists to optimize.	Machine-independent application programs on data. Use runtime system controls to optimize execution through load balancing, etc.
System access and use model	Main machine for batch processing when resources are ready. Uses offline visualization at remote site.	Interactive access with priority control and user intervention over large number of users simultaneously.
Reliability issues	Brittle systems with which to recover from most recent checkpoint. Must bring down system for maintenance.	Flexible error detection and recovery. Use redundancy techniques to enter graceful degraded operations in case of failure.

Courtesy of Randal Bryant [5].

On the other hand, DISC systems are HTC data-center clusters which have all the advantages of server clusters we discussed in previous chapters. In other words, DISC systems are much better in terms of satisfying a large number of simultaneous users with a high task throughput rate. Data centers can eliminate all single points of failure. Therefore, they are more reliable and highly available. Above all, DISC systems are more cost-effective or time/space-efficient for small users. They can handle distributed computing applications using the cloud service model. **Figure 9.10** show the distinctions

between the two supercomputing paradigms. The advantages of using the DISC architecture to satisfy HTC applications in distributing systems or cloud data centers are obvious from this comparison.

## 9.2.3 Performance Metrics for HPC/HTC Systems

HPC performance has been well defined over the years. The most popular metrics used are Gflops and Tflops in the past, Pflops on top systems now, and Eflops for the future. Parallel benchmarking programs are available to evaluate HPC systems in batch processing of a large-scale problem. Well-known benchmark programs include the Linpack Benchmark, NAS, Splash, and Parkbench [41]. On the other hand, HTC performance on business servers, clusters, data centers, and cloud systems is much more complex a problem to evaluate because the systems are used by many clients simultaneously.

Thus, HTC performance is attributed to many orthogonal factors. Some of them are measurable and some are not. The basic assumption is that many (millions or more) independent users may use the shared resources in the cloud or data centers simultaneously. The goal is to satisfy as many concurrent users as possible, even if each user program is a simple web service, social contact, or simple cloud storage. In this section, we attempt to present some performance evaluations and QoS assessment models for both HPC and HTC systems.

### 9.2.3.1 Basic Performance Attributes

The performance models applied to evaluate MPPs, data-center clusters, and virtualized clouds could be very different. To fit the layered structure of the cloud computing paradigm, the performance metrics of IaaS serve as a basis to model PaaS performance. Similarly, the attributes applicable to model PaaS performance serve as a foundation to evaluate SaaS performance. A good model should be able to cover all the layers of computing services. The model is generic in the sense that it applies to various cloud platforms under different workload distributions. To assess QoS in cloud services, the impacts on the top layer (SaaS) should remain transparent to the lower layers (PaaS and IaaS). A cloud platform should be built to serve many users simultaneously.

Therefore, multitasking is a necessity to assess distributed system performance. Five basic performance metrics are introduced in this section: namely system throughput, multitasking scalability, availability measure, data security, and cost-effectiveness. This analytical model offers a first-order approximation to model the performance of any HPC or HTC system, including elastic clouds. One can always add more performance attributes to expand the model from the user's perspective to cover more dimensions or those concerns by service providers. In other words, refined performance models could be extended from basic attributes to include program behavior, environmental demand, QoS, and cost-effectiveness.

### 9.2.3.2 System Throughput and Efficiency

In general, the *system throughput* of a distributed system or a cloud platform measures the number of jobs that can be done per unit of time. The throughput measure is attributed to several key factors that affect the *total execution time* ( $T_{total}$ ) of all jobs processed in a given time window. First, we have to examine all component times in  $T_{total}$ . Each job submitted may experience an initiation time to acquire resources. This overhead time includes the boot time of all machine instances and the scheduling time of incoming user jobs. The type of the application program limits horizontal scalability. In general, the overhead can be attributed to five components: infrastructure initiation delay, resource provisioning delay, interjob communication delay, OS overhead, and loading application software overhead.

The OS and software overheads remain constant in running different jobs. The other three time factors may vary with problem size and system management policies. For simplicity in analysis, we lump all five time factors into a single overhead time  $kT_o$ , where  $T_o$  is their sum averaged over many jobs running in a fixed time interval. The constant  $k$  is a platform-dependent coefficient that varies with specific system configurations. This overhead applies to all three system models presented in this book. Let  $T_e(n, m)$  be the effective execution time to complete  $n$  independent jobs in a cloud platform, where  $m$  accounts for the number of machine instances in a given system configuration. These machine instances could be cluster nodes (or processor cores) if physical servers are used, or VM counts if a virtualized cloud is used.

We compute the total job execution time (or *makespan*) with  $T_{total} = kT_o + T_e(n, m)$ . Then the *system throughput* ( $\pi$ ) is defined by:

(9.1)

In the ideal case, the overhead  $kT_o$  is assumed to be very small or simply zero, compared with the magnitude of the execution time  $T_e$ . Hence, the ideal throughput is simply measured by  $n/T_e(n, m)$ . The *system efficiency* ( $\alpha$ ) is defined by normalized throughput as follows:

(9.2)

System efficiency represents the effective utilization of all provisioned resources in a system. Note that the parameters  $n$  and  $m$  in the expression  $T_e(n, m)$  vary with the workload (user number), system size ( $m$ ), the parallel and distributed computing model applied.

### 9.2.3.3 Multitasking Scalability

Multitasking implies the use of a system to handle many jobs simultaneously or concurrently. System services should be able to scale both horizontally across the machine or cluster size and vertically from applications to middleware, runtime and OS

support, and hardware. The ability to scale resources dynamically is crucial to provide elastic resources. By *horizontal scalability*, we mean to increase cloud resources of the same type. For example, the number of VM instances offers such a measure in a cloud.

*Vertical scalability* means the increase in performance due to adding more resources along the service layers. For simplicity in illustrating scalability, we concentrate on horizontal scalability. However, we consider both scale-up and scale-down situations. Sometimes, scaling down could be more efficient than scaling up. The potential benefit of a system or cloud platform is largely attributed to being able to scale resources up and down dynamically, on demand.

We define the *multitasking scalability* ( $\beta$ ) as follows. Recall that  $m$  is the number of machine instances and  $n$  is the total number of user jobs (or tasks) being processed. In general, we have  $n \gg m$ , because  $n$  could be in millions and  $m$  only in the hundreds. The upper bound on scalability is roughly indicated by the ratio  $n/m$ , which is the average number of user jobs executed per machine instance, when all overhead times  $T_o$  are ignored. The higher the overhead time, the lower the scalability. Of course, one can increase the magnitude of  $m$  to approach  $n$  to get a 100 percent scalability. In reality, achieving a scalability  $\beta$  of around 20 percent is already very good in a real-life server cluster.

(9.3)

#### 9.2.3.4 System Availability

*System availability* ( $\gamma$ ) refers to the percentage of time the system is up and running normally. This percentage reflects the effects of downtime after unexpected failures and scheduled maintenance for software upgrades. Recall the availability definition given in Example 1.5 in Chapter 1; system availability is defined by the ratio of MTTF (mean time to failure) to the sum of MTTF and MTTR (mean time to repair). That definition applies to all systems (clusters, MPPs, grids, and clouds) studied in this book.

High availability (HA) specifies the performance level of a system with controlled downtime through quick recovery from failure. The term “continuous availability” (CA) refers an operational level with almost no failure. Generally speaking, system availability should be the responsibility of the providers. Usually, availability is maintained sufficiently high with at least five 9s, meaning that with 99.999 percent of uptime (at most, five minutes of downtime per year) the system can provide services. Of course, an HA cluster could demand 99.99999 percent availability, meaning only three seconds of downtime per year. In general, that would be very expensive to achieve in a real-life machine environment.

#### 9.2.3.5 Security Index

We assessed the vulnerability of commercial cloud platforms in Sections 6.3 and 6.5. Availability is affected by platform architecture, service models applied, system vulnerability, and resilience to network attacks. Google has hundreds of data centers and

more than half a million servers. The platform consists of the server cluster, GFS, and data centers. We discussed security issues in more detail in [Section 6.5](#). Here we summarize important factors affecting cloud security in a dynamic cloud environment.

Cloud security is attributed to user confidentiality, data integrity, access control, firewalls, IDSes, defense capability against viruses or worm attacks, reputation systems, copyright protection, data lock-in, APIs, data-center security policies, trust negotiation, and security auditing services. We suggest fine-grained access control at the file level in data centers. Trust among resource sites can be negotiated with nonconflicting security policies. To secure elastic resources, the reputation system should be designed to safeguard scattered resource sites and data centers. The resource site security index ( $\delta$ ) and user-access records should be maintained through periodic auditing.

### 9.2.3.6 Cost Effectiveness

This refers to the estimate of an effective scale of economy achievable by a given system. The following cost model for clouds is a rough estimation. This model considers costs by both the cloud service provider and the data-center owner. The provider must pay the data-center owner for the resources used even if they belong to the same company. Regardless of the cloud service model applied, the cost of cloud usage is primarily determined by the hours used.

We denote  $c$  as the hourly charge to the user by the provider and  $h$  as the number of service hours provided to end users. The total income of the provider from user payment of the service is calculated by  $Cost_p = hc$ . The total cost the provider paid to the data center is estimated by  $Cost_d = h(d/u)$ , where  $d$  is the hourly charge by the data center and  $u$  is the utilization rate of the data-center resources. This rate often falls in the range of 0.6 to 0.8 as estimated by the Berkeley Group. The *cost-effectiveness* ( $\mu$ ) of the cloud service is a measure for assessing the profit margin by the service provider.

(9.4)

As a numeric example, given  $c = \$3/\text{hour}$ ,  $d = \$1/\text{hour}$ , and  $u = 60$  percent in using Amazon EC2, this means the provider has a profit margin of  $\mu = 80$  percent on top of its cost if the data-center resources are 60 percent utilized. This cost model gives a very rough estimate of the profit margin by the provider. If the provider charges the user a reduced hourly rate of  $\$2/\text{hour}$ , the profit will be reduced to only 20 percent of the cost. In general,  $c$  is set to be greater than  $d$  by a large margin.

### 9.2.4 Quality of Service in Cloud Computing

We present the cumulative performance of a cloud platform by a compound metric, called *quality of cloud services* (QoCS) and denoted by  $\theta$ . This metric measures the overall quality of the cloud service, regardless of the particular service models applied. We use a five-dimensional Kiviat graph to plot the aggregated cloud performance or to quantify

the QoCS graphically. Each Kiviat graph has five orthogonal dimensions, representing the five performance attributes we introduced earlier. The Kiviat graph provides a holistic view of the capabilities of the cloud service model. The shaded area of the Kiviat graph represents the cumulative strength or quality of the cloud service being provided.

Specifically, the larger the shaded area  $A_{shaded}$  on the Kiviat graph for a given user application running on the cloud platform, the better the QoCS,  $\theta$ , of the cloud platform being tested for a given cloud application. The scale and range of these five dimensions are specified by the template Kiviat graph in [Figure 9.11\(a\)](#). Let  $A_{pentagon}$  be the ideal operating area of the pentagon in the Kiviat graph. The scale on all dimensions is normalized to be 1. In other words, the maximum radius of the pentagon is set to be 1 along all dimensions. All five fractional measures are scaled from 0 to 1. The scale of 0 is located at the center of the perfect pentagon, meaning the lowest values of the five performance parameters ( $\alpha, \beta, \gamma, \delta, \mu$ ). The parameter value of 1 means the highest performance in each dimension of the measure. We formally defined the QoCS by the following ratio:

(9.5)

The lower value of the  $\theta$  measure means poor performance. The higher value of  $\theta$  toward 1 implies higher performance. [Figures 9.11\(b-d\)](#) show the corresponding Kiviat graphs for three cloud services hypothetically running on three cloud platforms. This is based on the aforementioned analytical modeling of cloud performance. It would be interesting to validate the analytical results with real-life benchmark experiments on the cloud platforms from Amazon, Google, and [Salesforce.com](#).

### Example 9.6

#### Performance Analysis of Three Cloud Services

Three cloud service models are evaluated analytically by projected hypothetical runs with three known cloud applications. One can run Amazon EC2 to test IaaS performance. The SaaS test runs on GAE with [Salesforce.com](#)'s CRM software. The Hadoop application is tested on IBM Blue Cloud to project PaaS performance. These projected runs are under two extreme cloud conditions: One corresponds to the best scenario with HA and high security assurance, and the other corresponds to the worst case of low availability and insecurity. The remaining three dimensions assume attribute values by common-sense estimates.

The small pentagon areas in [Figure 9.11\(b\)](#) and [9.11\(d\)](#) resulted from the worst-case experiments. In [Figure 9.11\(c\)](#), the EC2 application is tested on the AWS platform under the best platform conditions with high efficiency, continuous availability, highly secured data centers, and cost-effective

returns. The Kiviat graph in Figure 9.11(c) reports the best results with a much larger pentagon area. These plots demonstrate the relative strengths and weaknesses of the three cloud service models in terms of the five attributes ( $\alpha, \beta, \gamma, \delta, \mu$ ).

The results shown in Table 9.5 are estimated values of QoCS and the relative rankings of the three cloud service models under optimistic versus pessimistic assumptions. The pessimistic QoCS results indicate that all three service models have demonstrated about 30 percent of their maximum performance. The QoCS of the IaaS is very low compared with those for the PaaS and SaaS models shown in Figure 9.11(b) and 9.11 (c), respectively. This happens because of the scalability constraints and lower cost-effectiveness values assumed. The QoCS of the PaaS and SaaS models are fairly close to each other, with the PaaS model being secure and efficient and SaaS being cost-effective.

**Table 9.5**

**Relative Performance of Three Computing Cloud Service Models Applied to an EC2 Application under Two Extreme Security and Availability Conditions**

Service Model	QoCS of Cloud Services $\theta$ , under Worst-Case Scenario	Ranking Order under Worst-Case Scenario	QoCS of Cloud Services, $\theta$ , under Best-Case Scenario	Ranking Order under Best-Case Scenario
IaaS	19.3661%	3	59.7055%	1
PaaS	25.5026%	1	54.4187%	2
SaaS	22.2344%	2	52.1941%	3

In an optimistic analysis, the QoCS of the three cloud service models are in the range of 50 percent to 60 percent. The IaaS model performs better than the PaaS and SaaS models. This is attributed primarily to improved efficiency, scalability, and security measures. The PaaS and IaaS models have similar levels of performance. The PaaS model is slightly higher in performance than the IaaS model, thanks to significant improvements by providers of PaaS in the areas of efficiency and scalability. However, the ranking of IaaS, PaaS, and SaaS models varies with specific applications. Benchmark experiments are needed to rank cloud performance.

## 9.2.5 Benchmarking MPI, Azure, EC2, MapReduce, and Hadoop

Several detailed benchmarking studies of cloud system have been conducted [23–29] on

HPC applications. Some of these conclusions may be a little misleading as they use technologies such as message-passing interface (MPI), which is usually unsuitable for clouds, rather than, say, MapReduce, whose features we discussed in [Chapter 6](#). Other studies [15,30–38] have taken a more focused look at scientific problems which, as we discussed in [Section 6.2.6](#), work well with clouds. Some of the latter results on the performance of Twister were discussed in [Section 6.2.2](#) and are not repeated here.

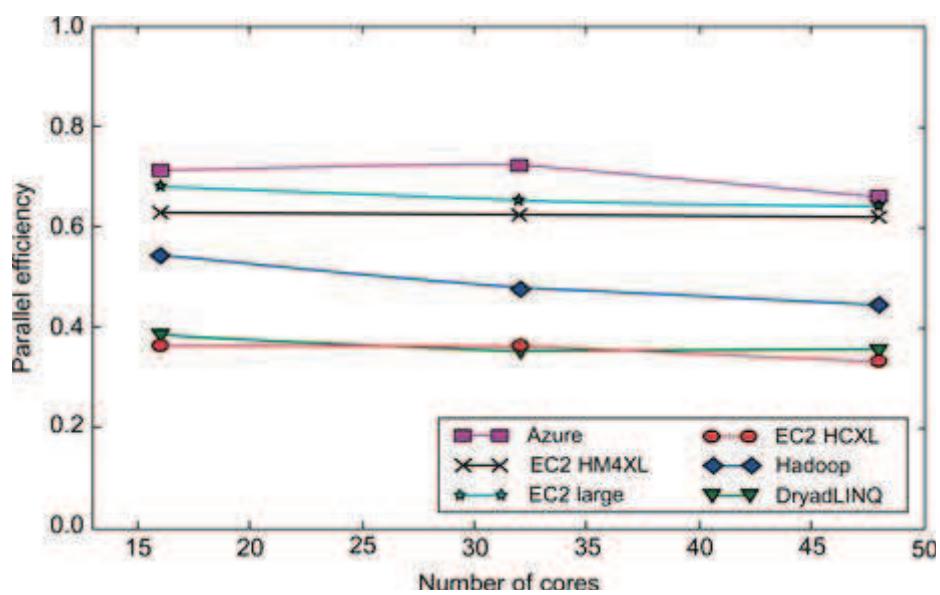
### 9.2.5.1 Competitive Cloud Performance

Our understanding is that clouds offer competitive performance and pricing for applications and programming paradigms that are well suited to cloud environments. For these cases, one benefits from the obvious utility computing feature of clouds, and for PaaS, one can also benefit from platform features such as Azure Table/Amazon SimpleDB which are not available on traditional HPC clusters. Performance-relevant problems with clouds include the bandwidth and cost of transmitting data to them; performance degradations in I/O which impact latency and bandwidth of MPI; and consequences of virtualization that negatively impact the affinity between compute and storage and between different compute instances [39]. Traditional large-scale simulations—often requiring low-latency, high-bandwidth communications and clever data decomposition schemes—perform very poorly on clouds.

The costs of various implementations were examined [41] on Amazon EC2 with the conclusions that compute costs dominated over storage but that costs are sensitive to particular Amazon instance types ([Table 6.13](#)). This sensitivity appears in many studies and suggests that the first step in any major cloud activity is to optimize the instance types used. Another astronomy application [42] looked at Amazon EC2 for the Nearby Supernova Factory (SNfactory) analysis environment that relies upon a complex pipeline of serial processes that execute various image processing algorithms in parallel on ~10 terabytes of data. This paper found that careful attention was needed to use storage effectively on EC2 while the original Linux cluster application needed modification to cope with failures present on clouds. Note that cloud-optimized environments such as MapReduce have failure tolerance built in.

### 9.2.5.2 MPI versus MapReduce Performance

Now we describe a set of performance measurements from the SALSA group at Indiana University comparing MPI, MapReduce, and master-worker approaches on clouds and traditional platforms using Linux and Windows [22,39]; further details may be found in papers from this group. Consider the performance of running the *generative topographic mapping* (GTM) interpolation algorithm on various EC2 sizes. The purpose is for dimension reduction to analyze the NIH *PubChem* data set of 26.4 million data points in 166 dimensions. Azure tests on variable instances are shown in [Figure 9.12](#).

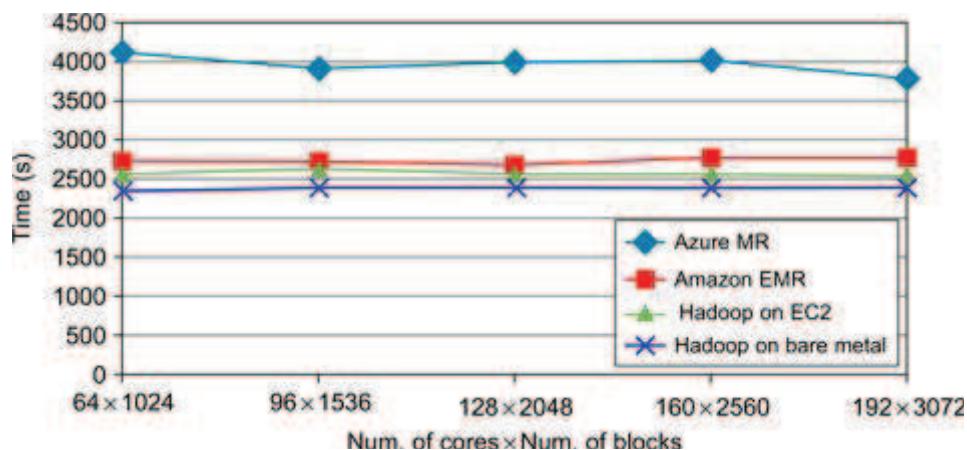


**FIGURE 9.12** Cloud performance variation with respect to the number of virtual cores used [19].

The EC2 tests were performed on EC2 Large, High-CPU-Extra-Large (HCXL), and High-Memory-Quadruple-Extra-Large (HM4XL) instances separately. HM4XL and HCXL instances use eight cores per instance while “large” instances use two cores per instance. The MapReduce implementations lie in the middle. The low efficiency is due to memory bandwidth effects on individual nodes, which is most pronounced for DryadLINQ runs on 16 core systems, whereas Hadoop systems had eight cores per node. This example illustrates that the cloud instance choice does affect the scalable performance.

### 9.2.5.3 MapReduce on Clouds and Bare Metal

Finally, we look at four different implementations of MapReduce [38] on the SWG computation introduced earlier (Figure 9.13). We compare MapReduce developed on Azure by the SALSA group with Hadoop on bare metal and two Amazon implementations: the “official” elastic MapReduce and Hadoop implemented on Amazon instances. The largest effect is a trivial one—the C# implementation is slower than the Java distance computation. This is removed by dividing out the single computation performance.



**FIGURE 9.13** Relative SWG performance of MapReduce on four cloud platforms: Azure MR, Amazon EMR,

Amazon EC2 is typically six times slower than a mid-range Linux cluster, and 20 times slower than a modern HPC system. However, the hardware, software, and interconnect on the EC2 cloud platform limit its performance in terms of raw speed. Clouds are very successful in business applications. In the scientific arena, data-intensive problems have achieved significant success on clouds, for two reasons. First, they often have the closely coupled structure that is well supported in clouds. Second, these are typically “new applications” which are attractive to build from scratch with programming models suited for clouds. Most of the current studies have looked at test kernels of a problem, but we can highlight two papers that examine complete applications. Montage delivers science-grade mosaics of the sky to a community composed of both professional and amateur astronomers and can naturally be implemented as a service in the cloud.

## 9.3 Enabling Technologies for the Internet of Things

In ubiquitous computing, the Internet of Things (IoT) provides a network of sensor- or radio-connected devices that can be uniquely identified and located in the cyber-physical space. This IoT is mostly wirelessly connected as a self-configuring network of radio-frequency tags, low-cost sensors, or e-labels. The concept is attributed to original auto-ID tracking in 1999. The term “IoT” combines RFID technology with today’s IPv6-based Internet technology. All things (objects) have IP addresses, which can be uniquely identified. The IP-identifiable objects are readable, recognizable, locatable, addressable, and/or controllable via the Internet, aided by RFID, WiFi, ZigBee, mobile networks, and GPS. The IoT enabling technologies are covered in this section for object tagging, wireless sensing, and locating positions. Interesting IoT applications are presented in Section 9.4.

### 9.3.1 The Internet of Things for Ubiquitous Computing

As introduced in Chapter 1, the IoT is a natural extension of the Internet. The foundation of the IoT is *radio-frequency identification* (RFID). This enables the discovery of tagged objects and mobile devices by browsing an IP address or searching for a database entry. Whereas RFID is a kind of amoeba of the pervasive computing world, the IoT presents a vision of the future. In the nineteenth century, computers learned to do as instructed. In the twentieth century, computers learned to think. In the twenty-first century, computers want to learn how to perceive by sensing and understanding.

Everyday objects include not only electronic devices, but also objects such as humans, animals, food, clothing, homes, vehicles, commodities, trees, hills, and landmarks. The IoT system could greatly reduce the chances of a company running out of stock or wasting products, as all concerned parties would know exactly what products are needed. Mislaid items and physical theft would be affected by the fact that the location of an item would be known to all parties. This demands that all objects on Earth be radio-tagged,

ranging from a milk carton to a truck container to a jumbo jet. Computers identify and manage all tagged objects in the same way humans do. This IoT system will improve our quality of life and make society cleaner and more secure, convenient, and enjoyable.

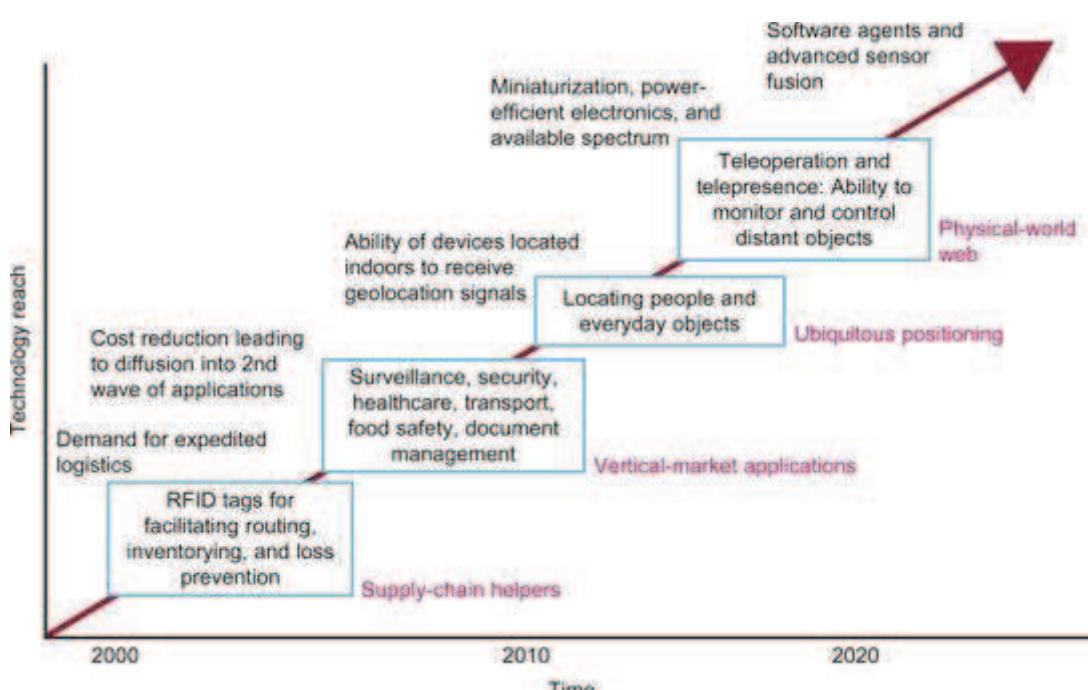
### **9.3.1.1 Ubiquitous Computing**

Ubiquitous computing is a post-desktop model of human-computer interaction in which information processing is integrated into everyday objects and activities. For daily activities, people may engage in using many pervasive devices simultaneously. They may not even be aware of the existence of the interactive devices. Although the idea is simple, its application is difficult. If all objects in the world were equipped with minuscule identifying devices, daily life on our planet could undergo a major transformation.

The IoT cannot be realized without systems design and engineering, and user interfaces. Contemporary human-computer interaction models, whether command-line, menu-driven, or GUI-based, are inappropriate and inadequate to meet ubiquitous computing demands. The natural IoT paradigm appropriate to a ubiquitous computing world has yet to emerge. Contemporary devices that lend support to ubiquitous computing include smartphones, tablet computers, sensor networks, RFID tags, smart cards, GPS devices, and others.

### **9.3.1.2 Development of the Internet of Things**

In 2005, the concept of the IoT entered the limelight. The IoT should be designed to connect the world's objects in a sensory manner. The approach is to tag things through RFID, feel things through sensors and wireless networks, and think things by building embedded systems that interact with human activities. A road map for IoT development is shown in [Figure 9.14](#) to anticipate the desired technology advances to make the IoT a reality in 25 years. In 2002, the U.S. *National Science Foundation* (NSF) called for the convergence of cutting-edge technologies including ubiquitous computing and nanotechnology.



**FIGURE 9.14** Technology road map of the Internet of things. Source: SRI Consulting Business Intelligence

In April 2008, the U.S. National Intelligence Council published a report titled “Disruptive Civil Technologies,” which also identified the IoT as a critical technology on U.S. interests out to 2025. Quantitatively speaking, the IoT should be designed to encode 50 trillion to 100 trillion objects. Moreover, the IoT should be designed to follow the movement of those objects. With today’s population of more than 6 billion worldwide, that means every person is surrounded by 1,000 to 5,000 objects on a daily basis.

### 9.3.1.3 Enabling and Synergistic Technologies

Many technologies can be applied to build the IoT infrastructure and specific IoT systems for special application domains. Supportive technologies are divided into two categories. *Enabling technologies* build up the foundations of the IoT. Among the enabling technologies, tracking (RFID), sensor networks, and GPS are critical. We will further illustrate these in [Figure 9.19](#). *Synergistic technologies* play supporting roles, as shown [Table 9.6](#). For example, biometrics could be widely applied to personalize the interactions among humans, machines, and objects. Artificial intelligence, computer vision, robotics, and telepresence can make our lives more automated in the future.

**Table 9.6**

#### Enabling and Synergistic Technologies for the IoT

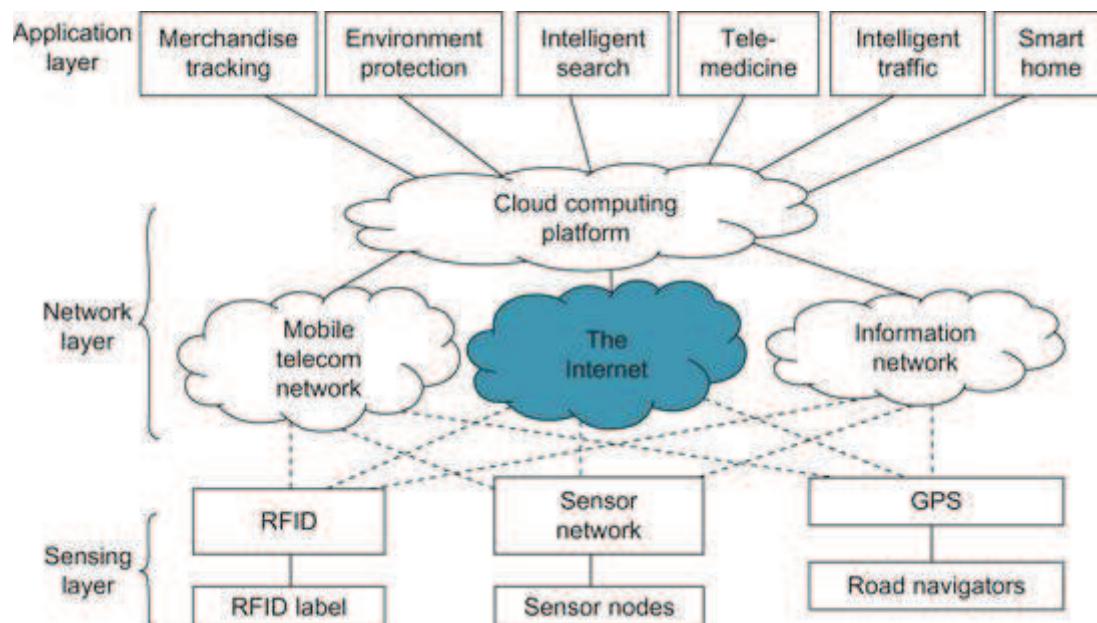
Enabling Technologies	Synergistic Technologies
Machine-to-machine interfaces	Geotagging/geocaching
Protocols of electronic communication	Biometrics
Microcontrollers	Machine vision
Wireless communication	Robotics
RFID	Augmented reality

Energy harvesting technologies	Telepresence and adjustable autonomy
Sensors and sensor networks	Life recorders and personal black boxes
Actuators	Tangible user interfaces
Positioning or location technology (GPS)	Clean technologies
Software engineering	Mirror worlds

The IoT development could become more mature and more sophisticated in the next 15 years. [Figure 9.14](#) shows the major technology advances and key applications that may benefit from the IoT. For example, supply chains are now better supported than before. Vertical market applications may represent the next wave of advances. Ubiquitous positioning is expected to become a reality as we move toward 2020. Beyond that, a physical IoT may be in place in a global scale. These advances will significantly upgrade human abilities, societal outcomes, national productivity, and quality of life.

### 9.3.1.4 Architecture of the Internet of Things

The IoT system is likely to have an event-driven architecture. In [Figure 9.15](#), IoT development is shown with a three-layer architecture. The top layer is formed by driven applications. The application space of the IoT is huge, and will be discussed in [Section 9.4.1](#). The bottom layers represent various types of sensing devices: namely RFID tags, ZigBee or other types of sensors, and road-mapping GPS navigators. The sensing devices are locally or wide-area-connected in the form of RFID networks, sensor networks, and GPSes. Signals or information collected at these sensing devices are linked to the applications through the cloud computing platforms at the middle layer.



**FIGURE 9.15** The architecture of an IoT consisting of sensing devices that are connected to various applications via mobile networks, the Internet, and processing clouds.

The signal processing clouds are built over the mobile networks, the Internet backbone, and various information networks at the middle layer. In the IoT, the meaning

of a sensing event does not follow a deterministic or syntactic model. In fact, the SOA model is adoptable here. A large number of sensors and filters are used to collect the raw data. Various compute and storage clouds and grids are used to process the data and transform it into information and knowledge formats. The sensed information is used to put together a decision-making system for intelligence applications. The middle layer is also considered as a Semantic web or grid. Some actors (services, components, avatars) are self-referenced.

## 9.3.2 Radio-Frequency Identification (RFID)

RFID is applied with electronic labels or RFID tags on any objects being monitored or tracked. The tagging may be applied to any objects, such as merchandise, tools, smartphones, computers, animals, or people. The purpose is to identify and track the objects using radio waves or sensing signals. Some tags can be read from tens or hundreds of meters away via a wireless reader. Most RFID tags contain at least two major parts. One is an integrated circuit for storing and processing information, modulating and demodulating a radio-frequency (RF) signal, and other special functions. The other part is an antenna for receiving and transmitting the radio signals.

### 9.3.2.1 RFID Tags and Device Components

There are generally three types of RFID tags: *active RFID tags* containing a battery and transmitting signals autonomously, *passive RFID tags* which have no battery and require an external source to provoke signal transmission, and *battery-assisted passive RFID tags* which require an external source to wake up the battery but have significantly higher forward link capability. Based on the radio frequency used, the passive RFID tags operate with from *low frequency* (LF) to *high frequency* (HF), *ultra high frequency* (UHF), and the *microwave range*. In terms of functionality, there are three major components of RFID hardware:

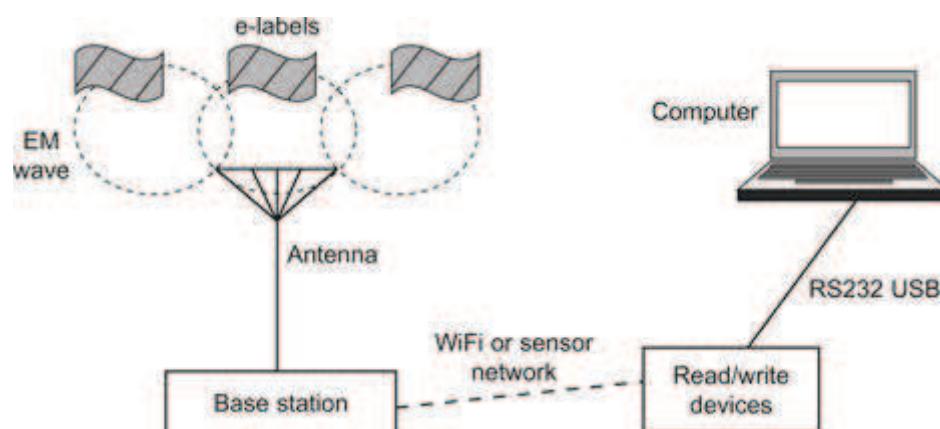
- **RFID tag** A tiny silicon chip attached to a small antenna.
- **Reader antenna** Used to radiate the energy and then capture the return signal sent back from the tag. It can be integrated with a handheld reader device or connected to the reader by cable.
- **Reader** The device station that talks with the tags. A reader may support one or more antennae. Resembling an electronic barcode, a reader device can detect signals even without a line of sight.

Some RFID readers can identify multiple objects concurrently. And some RFID tag-reader architectures support security features such as requiring a human operator to input a challenge code before decoding an ID. RFID devices have varying sizes, power requirements, operating frequencies, amounts of rewritable and nonvolatile storage, and software intelligence. They operate from a few centimeters to hundreds of meters. However, larger devices having an internal power source tend to operate at longer ranges. Conversely, very smaller devices have no internal power source.

### 9.3.2.2 How RFID Works

Innovations in RFID include *active*, *semi-active*, and *passive* RFID tags. These tags can store up to 2 KB of data and are composed of a microchip, antenna, and battery for active and semi-passive tags. The tag's components are enclosed within plastic, silicon, or sometimes glass. Data stored in the microchip waits to be read. The tag's antenna receives electromagnetic energy from an RFID reader's antenna. Using power from its internal battery or power harvested from the reader's electromagnetic field, the tag returns radio signals back to the reader. The reader picks up the tag's radio signals and interprets the frequencies as meaningful data.

There are two ways to couple schemes in RFID tags: inductively coupled or capacitively coupled. [Figure 9.16](#) shows the RFID operations between the RFID tags (e-labels), reading/writing devices, and backend computers. RFID tags are still not in widespread use by ordinary people, because they are expensive and bulky. They are more often used by large businesses, moving or shipping companies, and service companies. However, RFID tags can make our daily life or work much easier, more convenient, and accessible to visible or hidden objects surrounding us. These tags are not that expensive to produce in large quantities, and they can be made small enough to fit on almost any product or object.



**FIGURE 9.16** RFID tagging and signals read/write through sensor or WiFi networks.

External RF antenna devices can illuminate the tag devices similar to radar illuminating a target. However, RFID is more effective at operating at shorter ranges. A variety of agencies are defining the standards and regulating the use of RFID. They include the *International Organization for Standardization* (ISO), *International Electrotechnical Commission* (IEC), ASTM International, DASH7 Alliance, and EPCglobal. RFID has many industrial applications including merchandise tracking and supply chain management. Its use can improve the efficiency of inventory control and accuracy of assembly line management. Active and semi-passive RFID tags use internal batteries to power their circuits. An active tag can also use its own battery to broadcast radio waves to a reader, whereas a semi-passive tag relies on the reader to supply its power for broadcasting.

Active and semi-passive tags are reserved for reading over 30 to 100 meters, if repeater batteries are used to boost a tag's range. For example, a customer finds a pair of jeans in her size on a store shelf with a radio wand provided by the store. She can order the clothing and pay via her cell phone with the help of an RFID setting. Passive RFID tags rely entirely on the reader as their power source. These tags are read only up to 20 feet away. They have lower production costs, meaning they can be applied to less expensive merchandise. These tags are manufactured to be disposable. A railway car may apply active RFID tags. A bottle of shampoo may use only a passive tag.

Another factor that influences the cost of RFID tags is data storage. There are three storage types: *read-write*, *read-only*, and *write-once-read-many* (WORM). A read-write tag's data can be added to or overwritten. Read-only tags cannot be added to or overwritten. WORM tags can have additional data added once, but they cannot be overwritten. Most passive RFID tags cost 7–20 cents each. Active and semi-passive tags are more expensive, and RFID manufacturers typically do not quote prices for these tags without first determining their range, storage type, and quantity. The RFID industry's goal is to get the cost of passive RFID tags down to 5 cents each once more merchandisers adopt them.

### 9.3.3 Sensor Networks and ZigBee Technology

Today's sensor networks are mostly wireless, and are known as *wireless sensor networks* (WSNs). A typical WSN consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion, or pollutants. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance. WSN technology is now used in many industrial and civilian application areas, including process monitoring and control, machine health monitoring, environment and habitat monitoring, health care and home automation, and intelligent traffic control.

#### 9.3.3.1 Wireless Sensor Networks

A WSN is a group of specialized transducers with a communications infrastructure intended to monitor and record conditions at diverse locations. Commonly monitored parameters are temperature, humidity, pressure, wind direction and speed, illumination intensity, vibration intensity, sound intensity, power-line voltage, chemical concentrations, pollutant levels, and vital body functions. A sensor network consists of multiple detection stations called sensor nodes, each of which is small, lightweight, and portable. Every sensor node is equipped with a transducer, microcomputer, transceiver, and power source. The transducer generates electrical signals based on sensed data.

The microcomputer processes and stores the sensor output. The transceiver, which can be hard-wired or wireless, receives commands from a central computer and transmits data to that computer. The power for each sensor node is derived from the electric utility or from a battery. A sensor node may vary in size from that of a shoebox down to a dust grain. The cost of sensor nodes also varies a lot, ranging from \$100's to a few pennies. Size and cost constraints on sensor nodes are often decided by energy, memory,

computational speed, and bandwidth of the sensors used.

### 9.3.3.2 Wireless Network Support for Ubiquitous Computing

Table 9.7 compares four classes of wireless networks for ubiquitous computing. The ZigBee networks are mainly used in low-cost and low-speed monitoring and control applications, such as those used in a *wireless home area network* (WHAN). The GSM/GPRS or CDMA/1XRTT networks are cellular mobile networks that can cover a wide area for voice and data telecommunications. The WiFi networks specified in the IEEE 802.11b standard. WiFi is used for wirelessly accessing the Internet, reading e-mails, or conducting web page searches. Bluetooth is mainly used for short-distance wireless connection of computer peripherals (e.g., keyboard, mouse, printers, etc.). In terms of data rate, WiFi is the fastest (54 Mbps for 802.11g network). The next is Bluetooth with 720 Kbps and 115 Kbps for the 2.5G GPRS mobile network. ZigBee has the slowest rate of 20–250 Kbps. However, ZigBee has high reliability and low power/cost advantages.

**Table 9.7**

**Wireless Networks for Supporting Ubiquitous Computing**

Market Name Standard	ZigBee 802.15.4	GSM/GPRS CDMA/1XRTT	WiFi 802.11g	Bluetooth 802.15.1
Application focus	Monitoring and control	Wide area voice and data	Web, e-mail, video	Cable replacement
System Resources	4 KB–32 KB	18 MB+	1 MB+	250 KB+
Battery Life (days)	100–1,000+	1–7	0.5–5	1–7
Network Size	Unlimited ( $2^{64}$ )	1	32	7
Bandwidth (Kbps)	20–250	64–128+	54,000+	720
Range (meters)	1–100+	1,000+	1–100	1–10+
Success Metrics	Reliability, power, cost	Reach, quality	Speed, flexibility	Cost, convenience

Network self-organization is expected in a sensor network. Given the large number of nodes and their potential placement in hostile locations, it is essential that the network be able to self-organize; manual configuration is not feasible. Moreover, nodes may fail (either from lack of energy or from physical destruction), and new nodes may join the network. Therefore, the network must be able to periodically reconfigure itself so that it can continue to function. Individual nodes may become disconnected from the rest of the network, but a high degree of connectivity must be maintained.

Collaborative signal processing is desired in a sensor network. To improve detection/estimation performance, it is often quite useful to fuse data from multiple sensors. This data fusion requires the transmission of data and control messages. A user may want to query an individual node or a group of nodes for information collected in the region. Depending on the amount of data fusion performed, it may not be feasible to transmit a large amount of the data across the network. Instead, various local sink nodes will collect the data from a given area and create summary messages. A query may be

directed to the sink node nearest to the desired location.

### 9.3.3.3 Three Generations of Sensor Networks

Table 9.8 classifies WSNs into three generations over the past 30 years. The sensors used in the first generation were mainly vehicle-placed or air-dropped single sensors. They were quite bulky, like a shoebox, and weighed in the kilograms. Networks assumed only star or point-to-point topologies and were powered by large batteries that could last hours or days. In the second generation, the sensors became smaller, like a pack of playing cards, and weighed in the grams, and worked with AA batteries that lasted for days or weeks. They appear in client/server or P2P configurations. The current generation are as large as dust particles, weigh only a negligible amount, and are used in P2P networks for embedded and remote applications.

**Table 9.8**

**Three Generations of Wireless Sensor Networks**

WSN Features	First Generation (1990s)	Second Generation (2000s)	Third Generation (2010s)
Manufacturers	Custom constructors (e.g., for TRSS)	Crossbow Technology, Inc., Sensoria Corp., Ember Corp.	Dust, Inc., and others
Physical Size	Large shoebox and up	Pack of cards to shoebox	Dust particle
Weight	Kilograms	Grams	Negligible
Node Architecture	Separate sensing, processing, and communication	Integrated sensing, processing, and communication	Integrated sensing, processing, and communications
Topology	Point-to-point, star	Client/server, P2P	P2P
Power Supply	Large batteries; hours, days, and longer	AA batteries; days to weeks	Solar; months to years
Lifetime			
Deployment	Vehicle-placed or air-drop single sensors	Hand-placed	Embedded, sprinkled, left behind

Wireless ad hoc sensor networks apply a large number of (mostly stationary) sensors: Aside from the deployment of sensors on the ocean surface or the use of mobile, unmanned, robotic sensors in military operations, most nodes in a smart sensor network are stationary. Networks of 10,000 or even 100,000 nodes are envisioned in the future. The scalability becomes a demand. Low energy use is expected in the modern sensors. Since in many applications the sensor nodes will be placed in a remote area, service of a node may not be possible. In this case, the lifetime of a node is determined by the battery life, thereby requiring reduced energy expenditures or use of solar energy to power the devices.

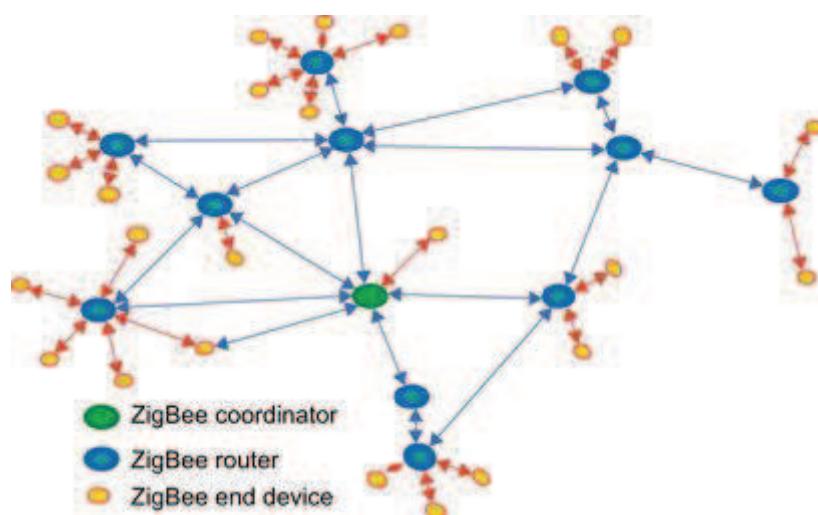
### 9.3.3.4 ZigBee Network

The term “ZigBee” is originated from the behavior of honeybees when they return to the

beehive. This is a high-level communication protocol using small, low-power, radio-based sensors as specified by the IEEE 802.15.4 standard. For example, ZigBee devices are applied in WHANs in wireless light switches and consumer electronics controlled by short-range radio. The ZigBee technology is simpler to use and less expensive than Bluetooth or WiFi. The RF applied in ZigBee results in low data rates, long battery life, and secure networking. The low cost allows the technology to be widely deployed in wireless control and monitoring applications. The low power usage allows a longer life with smaller batteries. The mesh networking provides higher availability and an extended operating range. There are three different types of ZigBee devices, as listed here and shown in [Figure 9.28](#):

- **ZigBee coordinator (ZC)** This is the most capable ZigBee device serving as the coordinator or the root of a ZigBee network. There is exactly one coordinator in each network since it is the device that started the network. It is able to store information about the network, including acting as the trust center and repository of security keys.
- **ZigBee Router (ZR)** This can act as an intermediate router, passing on data from end device to end device.
- **ZigBee End Device (ZED)** This contains just enough functionality to talk to the parent node (either the coordinator or the router). The end device cannot relay data from other devices. This relationship allows the node to be asleep a significant amount of the time, thereby ensuring a long battery life. A ZED requires the least amount of memory, and therefore can be less expensive to manufacture than a ZR or ZC.

ZigBee devices are required to conform to the IEEE 802.15.4 Wireless Personal Area Network (WPAN) standard. The standard specifies the lower protocol layers—the physical layer and the MAC at the data link layer. This standard operates in the unlicensed 2.4 GHz (worldwide), 915 MHz (Americas), and 868 MHz (Europe) radio bands. In the 2.4 GHz band there are 16 ZigBee channels, with each channel requiring 5 MHz of bandwidth. A typical ZigBee network is illustrated in [Figure 9.17](#). The radios use direct-sequence spread spectrum coding, which is managed by the digital stream into the modulator.

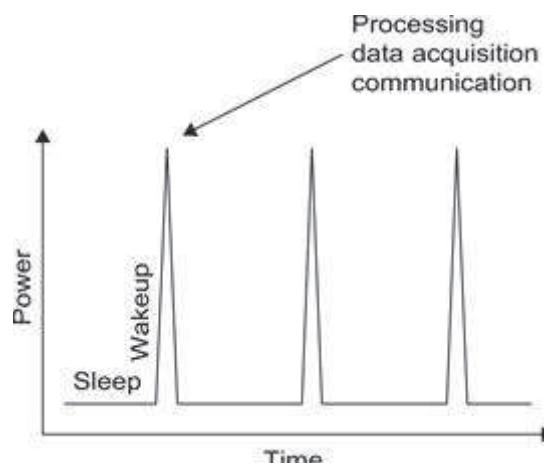


**FIGURE 9.17** Structure of a typical ZigBee network. Source: [http://mesh-matrix.com/en/technology/tech\\_zigbee.aspx](http://mesh-matrix.com/en/technology/tech_zigbee.aspx)

The ZigBee has a data rate of 250 Kb/second per channel in the 2.4 GHz band, 40 Kb/second per channel in the 915 MHz band, and 20 Kb/second in the 868 MHz band. Transmission range is between 10 and 75 meters. The output power of the radio is generally around 1 mW. The basic channel access mode is *Carrier Sense, Multiple Access/Collision Avoidance* (CSMA/CA). That is, the sensor nodes talk in the same polite way that people converse. They check to see that no one is talking before they start. Therefore, the hidden-terminal problem can be solved using a pair of RTS (*request to send*) and CTS (*clear to send*) handshaking signals to avoid collision.

### 9.3.3.5 Power Management in Wireless Sensor Networks

The sensors in most WSNs are in *sleep* mode a majority of the time. As illustrated in Figure 9.18, sleeping sensors consume only a few  $\mu\text{A}$ . They wake up quickly to process the sensed data and return to sleep again. *Active* mode is very brief, to minimize power consumption. This periodic application pattern results in a long lifetime for the sensors (months or years) without changing batteries. Good power management is the key to WSN success.



**FIGURE 9.18** Power management in a typical sensor application pattern.

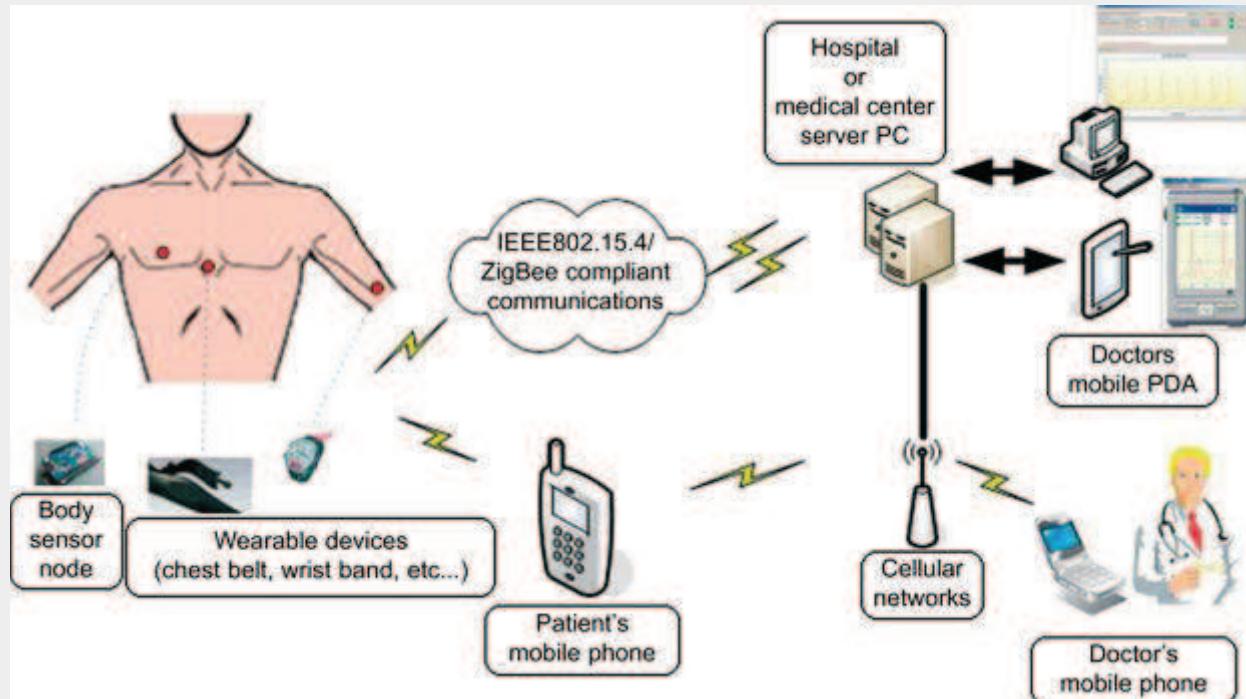
#### Example 9.7

#### Wireless Sensor Networks for Health Monitoring

Wireless sensors offer many possibilities for measuring different parameters of the human body and most of them are imperceptible and comfortable to use. Wireless sensors and a large communication capacity make it possible to measure people without inhibiting their everyday life; such wireless measurements are being implemented in the Ubiquitous City. UWB and ZigBee solutions are studied widely in many countries all over the world, and

thus they have become more popular in hospital and rehabilitation environments (see Figure 9.19).

Even though a wide selection of commercial sensors are available, many factors—for example, motion artifacts and noise—have to be considered before implementing a sensor into an application. A large number of measurements and testing with all used sensors needs to be performed to make sure all results are reliable and ready to be analyzed by a health care professional. Also, combining and analyzing a large amount of data from different sensors needs a new kind of approach. In addition, implementing a noninvasive glucose measurement unit into a wireless health monitoring system is being studied intensively.



**FIGURE 9.19** An example of how measured data can be transferred to doctors or medical professionals using a wireless sensor network. Source:

[www.infotech.oulu.fi/Annual/2007/pics/opme\\_19\\_sm.jpg](http://www.infotech.oulu.fi/Annual/2007/pics/opme_19_sm.jpg)

### 9.3.4 Global Positioning System (GPS)

*Location-based service (LBS)* helps people and machines find things and determine their physical whereabouts. Sensors play a role in dead reckoning, but that approach does not satisfy practical needs for geolocation. This has triggered the creation of global positioning and navigation systems. The GPS was developed in 1973 by the U.S. Air Force. Similar developments also occurred in the European Union, Russia, and China. Generally, the GPS is augmented by other signals and cellular towers. Fixed or orbiting satellite transmitters broadcast timing signals and receiving device response signals to

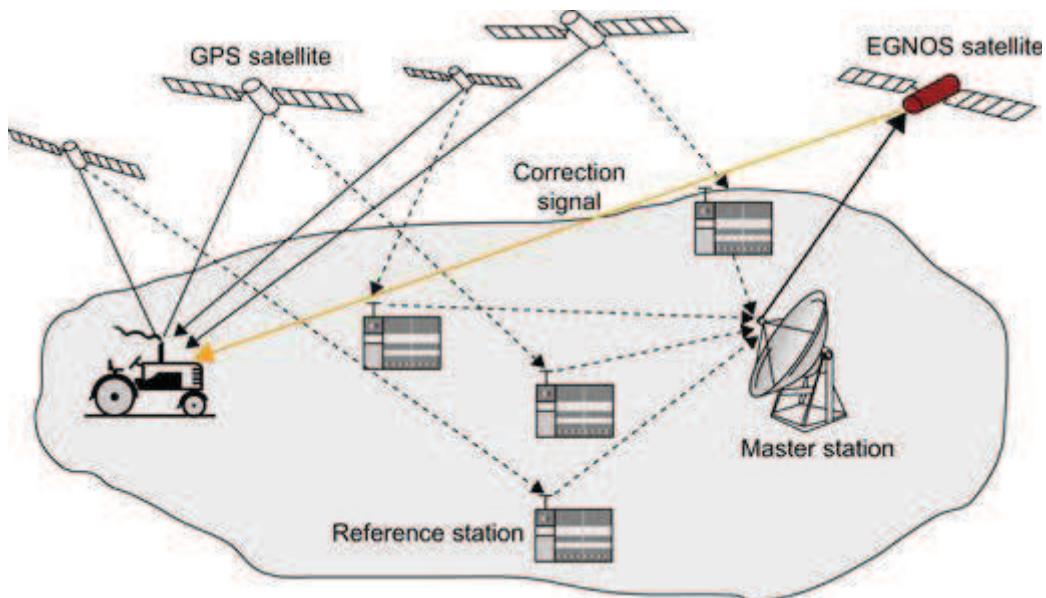
locate the position of moving objects.

Radar, lidar, and sonar can all detect relative locations of things using electromagnetic, optical, and acoustic signals. With help from satellites, some objects can be also located by attaching active GPS devices to disclose their locations by radio, light, and/or sound. Since 1994, a degraded GPS has been made available for civilian applications in providing reliable positioning, navigation, and timing services. For anyone with a GPS receiver, the system will provide accurate location and time information for an unlimited number of users in all weather conditions, day and night, anywhere in the world.

### **9.3.4.1 How GPS works**

GPS consists of three parts: the *space segment*, *control segment*, and *user segment*. The U.S. Air Force develops, maintains, and operates the space and control segments. There are 24 satellites deployed around the Earth in fixed orbits. The satellites orbit at an altitude of approximately 20,200 KM. GPS satellites broadcast signals from space, by which each GPS receiver calculates its 3D location (*latitude, longitude, and altitude*) plus the current time. The space segment is composed of 24 satellites in medium Earth orbit and also includes the boosters required to launch them into orbit. GPS satellites circle Earth twice a day in very precise orbits and transmit signals to Earth. GPS devices on the ground receive these signals and use a triangulation method to calculate the user's exact location. At different times, different satellites will become visible to the receiver.

As shown in [Figure 9.20](#), the control segment is composed of a master control station and a host of dedicated and shared ground antennas and monitor stations. The user segment is composed of hundreds of thousands of U.S. and allied military users of the secure GPS precise positioning services. Tens of millions of civil, commercial, and scientific users are only allowed to use a degraded functionality of the so-called standard positioning services that cannot be used in hostile attack purposes. Essentially, the GPS receiver compares the time a signal was transmitted by a satellite with the time it was received. The time difference tells the GPS receiver how far away the satellite is. With distance measured from several satellites, the receiver can determine the user's position and display it on the unit's electronic map.



**FIGURE 9.20** The ground GPS receiver, which calculates its 3D location from four or more satellites with help from a few ground reference stations and a master station.

### 9.3.4.2 Passive versus Active GPSes

The GPS tracking device makes it possible to track people, vehicles, and other assets, anywhere on Earth. There are two types of GPS tracking systems: passive and active. In passive tracking, the GPS is just a receiver, not a transmitter. Passive GPS tracking devices lack a transmission capability to send the GPS data from the vehicle. Therefore, passive GPS is also known as data loggers used primarily as recording devices. Active GPS tracking units incorporate a method to transmit user information from a vehicle. Although satellite uplink of data is available, cellular data communication is the most common and cost-effective. Automatic incremental updating provides a continuous source of tracking throughout a recording period. This provides current as well as historical logging positions.

Passive GPS tracking devices store GPS location data in their internal memory which can then be downloaded to a computer for viewing at a later time, while active GPS tracking systems send the data at regular intervals to be viewed in real time. When real-time data is not required, passive GPS tracking devices tend to be favored more by individual consumers for their compact convenience and affordability. Concerned parents can install a GPS tracking unit just about anywhere in their teens' vehicles to monitor their driving habits and know where they've been going, and even law enforcement officials now rely on passive GPS tracking to trail criminal suspects and enhance civilian safety via electronic surveillance of parolees. Passive GPS tracking units also serve as a theft prevention and retrieval aid in consumer as well as commercial vehicles.

### 9.3.4.3 Operating Principles of GPS

Knowing the distance from the receiver to a fixed-position satellite implies that the receiver is on the surface of a sphere centered at the satellite. With four satellites, the

receiver location is detected at the intersection of four sphere surfaces. The intersection of two satellite spheres is a circle in general. This circle could be reduced to a single point, if the two spheres merely touch on their surfaces. Having found those two intersecting sphere surfaces, now we consider how the intersecting circle intersects with a third satellite signal sphere. A circle and a sphere surface intersect at zero, one, or two points. With the receiver on the surface of Earth, the receiver needs to choose the point which is closest to the receiver from at most two intersecting points.

Obviously, this triangulation method may result in some errors in narrowing down to exactly one point with a minimum of inaccuracy. To locate the point accurately, the receiver has to use a fourth satellite to hone in almost precisely. The fourth satellite sphere will come very close to the final two intersecting points of three satellite spheres. The final receiver location is thus decided by noting the closest point calculated from the two final points to the sphere surface of the fourth satellite. In the case of no error, the precise position is located. Otherwise, some offset—say, 10 meters from the exact location—could be resulted from the error introduced. To further reduce the errors, more satellites could be involved, but this could be rather costly.

Each satellite continually transmits messages that include the time the message was transmitted, precise orbital information (the ephemeris), and the general system health and rough orbits of all GPS satellites (the almanac). A GPS receiver calculates its position by precisely timing the signals sent by GPS satellites high above Earth. A GPS receiver must be locked onto the signal of at least three satellites to calculate a 2D position (latitude and longitude) and track movement. With four or more satellites in view, the receiver can determine the user's 3D position (latitude, longitude, and altitude).

Once the user's position has been determined, the GPS unit can calculate other information, such as speed, bearing, track, trip distance, distance to destination, sunrise and sunset times, and more. The receiver utilizes the messages it receives to determine the transit time of each message and computes the distances to each satellite. These distances along with the satellites' locations are used to compute the position of the receiver. This position is displayed perhaps with a moving map display or latitude, longitude, and elevation information. Many GPS units show derived information such as direction and speed calculated from positional changes.

#### 9.3.4.4 Triangulation Location Calculation

This location calculation method is illustrated in Figure 9.20. The receiver uses messages received from four satellites to determine the satellite positions and time sent. The  $x$ ,  $y$ , and  $z$  components of position and the time sent are designated as  $[x_i, y_i, z_i, t_i]$ , where the index  $i = 1, 2, 3$ , or  $4$  to denote the satellite. Knowing when the message was received  $t_{ri}$ , the receiver computes the message's transit time as  $t_{ri} - t_i$ . Assuming the message traveled at the speed of light  $c$ , the distance traveled is calculated by  $d_i = (t_{ri} - t_i) \times c$ .

Having discussed how sphere surfaces intersect, we now formulate the equations for the case when errors are present. Let  $b$  denote the clock error or bias, the amount by which the receiver's clock is off. The receiver has four unknowns, the three components

of GPS receiver position and the clock bias  $[x, y, z, b]$ . The equation of the sphere surfaces is calculated by the following expression for  $i = 1, 2, 3$ , and 4:

(9.6)

Multidimensional root finding methods such as the Newton-Raphson method can be used. The approach is to linearize around an approximate solution—say,  $[x^{(k)}, y^{(k)}, z^{(k)}, b^{(k)}]$ , at iteration  $k$ , then solve four linear equations derived from the quadratic equations above to obtain the corresponding values at time instance of  $k + 1$ . The Newton-Raphson method converges faster than other position methods. When more than four satellites are available, the calculation can choose from four possible results. The error of position calculation is very sensitive to the clock error. Therefore, in the satellite-based navigation system, clock synchronization is critically important to minimize location errors.

Three satellites are enough to locate the receiver position, since space has three dimensions and a position near Earth's surface is assumed. However, even a very small clock error multiplied by the light speed of satellite signals may result in a large positional error. Therefore, most receivers use four or more satellites to solve for the receiver's location and time. The computed time is often hidden by most GPS applications, which use only the location information. A few specialized GPS applications do use the time for time transfer, traffic signal timing, and synchronization of cell phone base stations.

Although four satellites are required for normal operation, if one dimensional variable is already known, a receiver can determine its position using only three satellites. For example, a ship or plane may have known elevation. Some GPS receivers may use additional clues or assumptions (such as reusing the last known altitude, dead reckoning, inertial navigation, or including information from the vehicle computer) to give a less accurate (degraded) position when fewer than four satellites are available.

### 9.3.4.5 Worldwide Deployment Status

In addition to the GPS deployed by the United States which is now open for global civilian applications by many countries, Russia has deployed GLONASS (for Global Navigation Satellite System) for Russian military use exclusively. In the European Union, there is the Galileo positioning system (EGNSS). By 2011, China had launched 8 satellites for the 31-satellite Beidou navigation system. The full Beidou system will become operational by 2020.

Civilian GPS applications include navigation, surveying, mapmaking, cellular phones, tectonics, disaster relief and emergency services, GPS tours, geofencing, recreation, aircraft tracking, and geotagging, among other. Military applications of GPS include navigation, target tracking, missile and projectile guidance, search and rescue, reconnaissance, and nuclear detonation detection, among others. The following example shows a civilian application.

### **Example 9.8**

#### **Real-Time Vehicle Tracking Using Active GPS Devices**

Active GPS tracking systems find their applications primarily in industries and businesses. These systems are fast becoming the standard for companies that wish to monitor fleet vehicles as well as other heavy equipment. Real-time GPS tracking is practical for acquiring immediate and detailed information about large numbers of vehicles or objects that are being tracked. An example is a car rental business that provides cars for many customers; real-time vehicle tracking systems could prove beneficial in supplying current and historical data on routes taken by all the drivers. The process is divided into the following four steps:

1. GPS receivers in each car asset receive signals from a network of satellites.
2. The collected satellite information is sent to the communication center via mobile networks.
3. The control center enters calculated location information over global maps.
4. The control center sends commands to each unit to trigger alarms, stop engines, change direction, or send personal messages, and so on.

## **9.4 Innovative Applications of the Internet of Things**

In this section, we will review the application domains of the IoT. In particular, case study examples of the IoT are reported for applications in tracking, retailing, supply chain management, smart power grid, and smart buildings. Finally, European IoT projects are assessed along with American *cyber-physical system* (CPS) projects for general-purpose applications.

### **9.4.1 Applications of the Internet of Things**

Table 9.9 summarizes IoT applications in three major civilian application domains. Obviously, the IoT has a lot of military applications, which is beyond the scope of this book. In general, use of the IoT aims to promote industrial productivity and raise economical growth. The IoT plays important roles in environment protection, including pollution control, weather forecasting, and disaster avoidance and recovery. In terms of societal impacts, the IoT can make our lives more convenient and comfortable. Government services, law enforcement, and home and health improvements are the major beneficiaries.

**Table 9.9**

## Some Application Domains of the Internet of Things

Domain	Brief Descriptions	Indicative Examples
Growth in industry and economy	Activities involving financial or commercial transactions between companies and organizations	Manufacturing, logistics, service sector, banking, financial governmental authorities, intermediaries, etc.
Environment and natural resources	Activities regarding the protection, monitoring, and development of all natural resources	Agriculture and breeding, recycling, environmental management services, energy management, etc.
Society and daily life	Activities/initiatives regarding the development and inclusion of societies, cities, and people	Governmental services toward citizens and society structures, e-inclusion (aging, disabled people), etc.

### 9.4.1.1 Specific Wireless Sensor Applications

Highlighted below are several applications of sensor networks:

- Military sensor networks to detect and gain as much information as possible about enemy movements, explosions, and other phenomena of interest
- Sensor networks to detect and characterize chemical, biological, radiological, nuclear, and explosive attacks and materials
- Sensor networks to detect and monitor environmental changes in plains, forests, oceans, and so on
- Wireless traffic sensor networks to monitor vehicle traffic on highways or in congested parts of a city
- Wireless surveillance sensor networks for providing security in shopping malls, parking garages, and other facilities
- Wireless parking lot sensor networks to determine whether the lot is occupied or available

### 9.4.2 Retailing and Supply-Chain Management

IoT applications will promote business, improve our society, and enhance economical growth. These include specific IoT applications in railings and logistics services and supply chain management. Many industrial, government, and community services can benefit from these applications. These include activities or initiatives to promote the development of better and more efficient societies, cities, and governments.

#### 9.4.2.1 Retailing and Logistics Services

Emergence of RFID applications depends strongly on adoption by retailers, logistics organizations, and package-delivery companies. In particular, retailers may tag individual objects in order to solve a number of problems at once: accurate inventorying, loss control, and ability to support unattended walk-through point of sale terminals (which promise to speed checkout while reducing both shoplifting and labor costs). Cold-chain auditing and assurance could require tagging food and medicine with temperature-

sensitive materials and/or electronics; ensuring or monitoring whether perishable materials are intact and/or need attention may entail communications among things, refrigeration systems, automated data logging systems, and human technicians.

For example, at the grocery store, you buy a carton of milk. The milk container would have an RFID tag that stores the milk's expiration date and price. When you lift the milk from the shelf, the shelf would display the milk's specific expiration date, or the information could be wirelessly sent to your personal digital assistant or cell phone. The idea is illustrated in [Figure 9.25](#). As you exit the store, you pass through doors with an embedded tag reader. This reader would tabulate the cost of all the items in your shopping cart and send the grocery bill to your bank. Product manufacturers would know what you've bought and the store's computers would know exactly how much of each product needs to be reordered.

Once you get home, you would put your milk in the refrigerator, which is equipped with a tag reader. This smart refrigerator would be capable of tracking all of the groceries stored in it. It would be able to track the food you use and how often you restock your refrigerator, and could let you know when that milk and other food spoils. Products would also be tracked when they are thrown into a trash can or recycle bin. Based on the products you buy, your grocery store would get to know your unique preferences. Instead of receiving generic newsletters with weekly grocery specials, you might receive one created just for you.

### **9.4.2.2 Supply Chain Management**

Supply chain management can be aided by an IoT system. The idea is to manage a whole network of related businesses or partners involved in product manufacturing, delivery, and services as required by end customers. At any given time, market forces could demand changes from suppliers, logistics providers, locations and customers, and any number of specialized participants in a supply chain. This variability has significant effects on the supply chain infrastructure, ranging from the foundation layers of establishing the electronic communication between the trading partners to the more complex configuration of the processes and the arrangement of workflows that are essential to a fast production process.

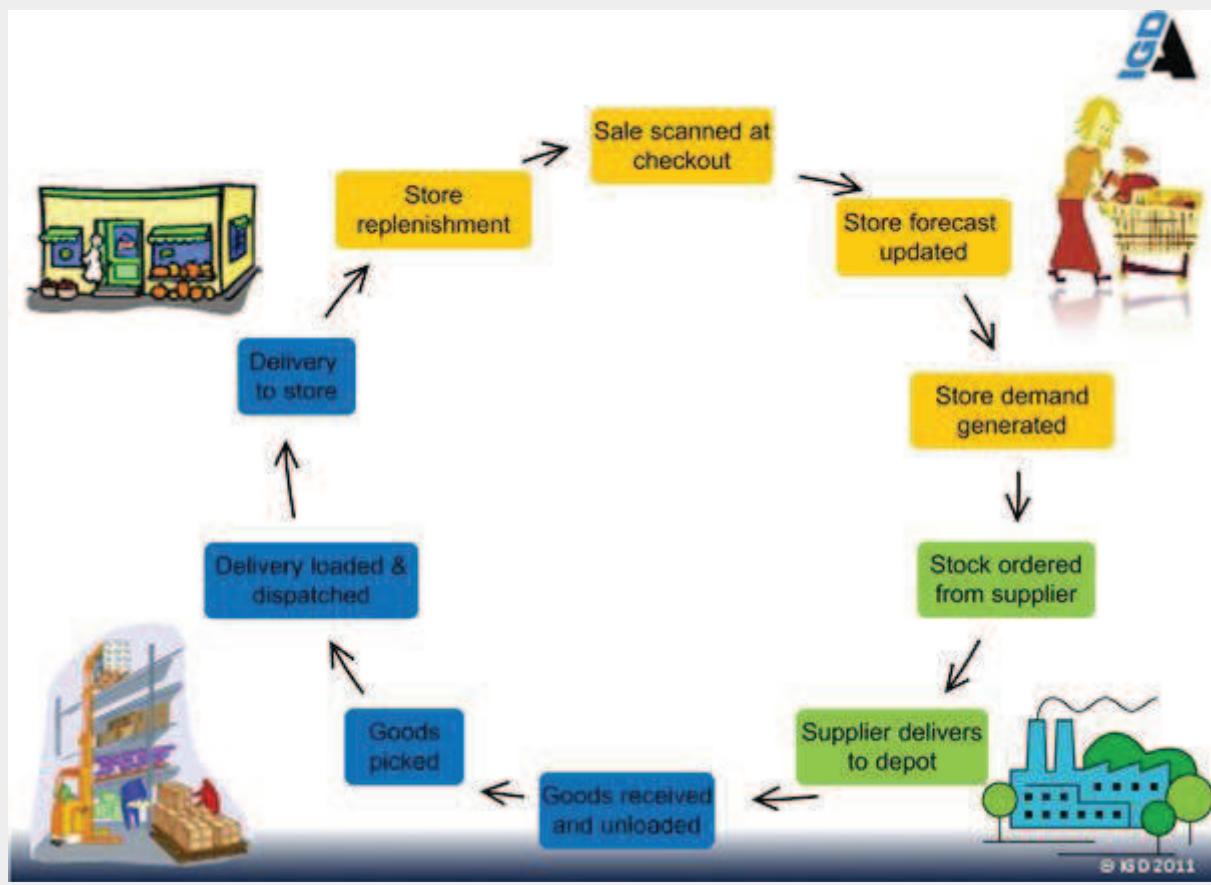
A supply line combines the processes, methodologies, tools, and delivery options to guide collaborative partners to work in a sequence to conduct business with high efficiency and delivery speed. The cooperative companies must work in lock-step quickly as the complexity and speed of the supply chain increase due to the effects of global competition, rapid price fluctuations, surging of oil prices, short product life cycles, expanded specialization, and talent scarcity. A supply chain is an efficient network of facilities that procures materials, transforms these materials to finished products, and finally distributes the finished products to customers. The following example could explain how the supply chain can be aided by the IoT, which is particularly tailored to promote business efficiency and fast growth.

## Example 9.9

### Supply Chain Management Aided by the Internet of Things

Supply chain management is a process used by companies to ensure that their supply chain is efficient and cost-effective. Figure 9.21 illustrates the supply chain for the production and sale of consumer products. The supply chain involves material or component suppliers, distribution centers, communication links, cloud data centers, a large number of retail stores, corporate headquarters (e.g., Wal-Mart), and bank payments. These business partners are linked by satellite, Internet, wired and wireless networks, truck, train, or shipping companies, and electronic banking, cloud providers, and so on. Sensors, RFID tags, and GPS devices could be placed everywhere along the supply chain. The idea is to promote online business, e-commerce, or mobile transactions. Supply chain management consists of five major stages:

- **Stage 1: Planning and Coordination** A plan or strategy must be developed to address how a good or service can satisfy the needs of customers.
- **Stage 2: Material and Equipment Supplies** This phase involves building a strong relationship with the raw material suppliers and planning methods for shipping, delivery, and payment.
- **Stage 3: Manufacturing and Testing** The product is tested, manufactured, and scheduled for delivery.
- **Stage 4: Delivery of Products** Customer orders are taken and delivery of goods is planned.
- **Stage 5: After-Sale Service and Returns** Customers may return defective products and the company addresses customers' demands. Supply chain software is used by many companies for efficient supply chain management.



**FIGURE 9.21** Supply chain management in the multipartner business arena. Courtesy IGD Academy

## 9.4.3 Smart Power Grid and Smart Buildings

Environmental protection is a global issue. Green energy is desired to save energy and to reduce the effects of global warming. A critical IoT application is promotion of a smart power grid.

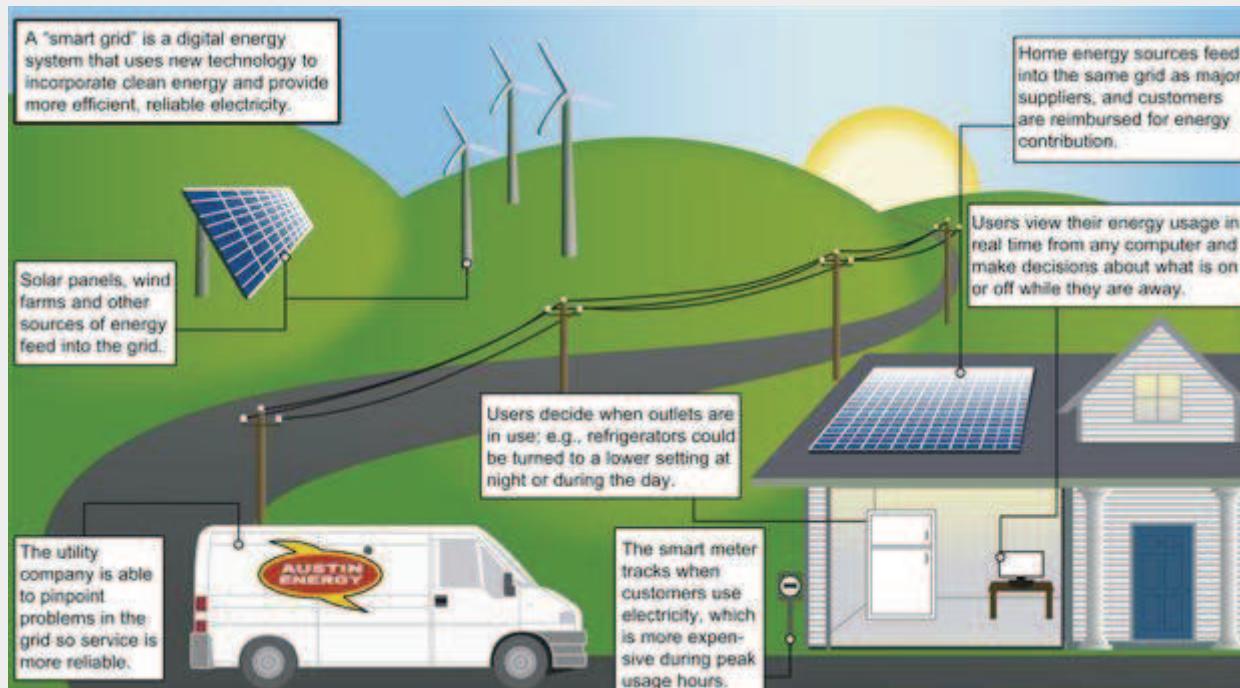
### 9.4.3.1 Smart Power Grid

Various power companies across the United States have or are in the process of upgrading their power management and distribution systems. Various sensors at individual homes (smart thermostats) can collect information that is sent via a network to main stations (perhaps even local “hubs”) that can apply complex power management and send control signals back to the grid to save energy. The smart grid is made possible by applying sensing, measurement, and control devices to electricity production, transmission, distribution, and consumption.

#### Example 9.10

## A Smart Power Grid Supporting the Internet of Things

A smart grid includes an intelligent monitoring system that keeps track of all electricity flowing in the system. Smart meters, a digital upgrade of current utility meters, track energy usage in real time so that both the customer and the utility company know how much is being used at any given time. Energy is paid for using “time of day” pricing, meaning electricity will cost more at peak times of use. Figure 9.22 illustrates IoT support of a smart power grid.



**FIGURE 9.22** Smart power grid. Source: <http://impactnews.com/images/stories/CTA/2009/04/15-grid.jpg>.

Courtesy of: Community Impact Newspaper

For example, when power is least expensive the user can allow the smart grid to turn on selected home appliances such as washing machines or factory processes that can run at arbitrary hours. At peak times it could turn off selected appliances to reduce demand. More involved users would be able to use the smart meter to view energy usage remotely and make real-time decisions about energy consumption. A refrigerator or air conditioning system could be turned down remotely while residents are away.

### 9.4.3.2 Smart Buildings

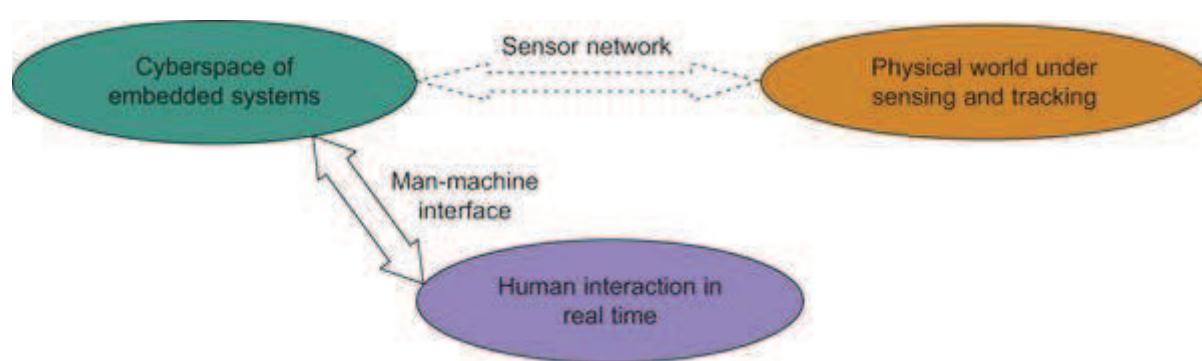
The IoT could enhance our lives with comfort, convenience, and security, and reduce energy costs and environmental impacts. The IoT has been suggested in construction of smart buildings in residential, commercial, industrial, and government settings. This may include alarm systems, access controls, indoor climate controls, elevators, and so forth. We desire to diffuse IT into lighting, appliances, and furniture in buildings. A

smart building can be a shopping mall or a home, a hospital or a high-rise office tower. Smart buildings need monitoring and regulation of heating, air conditioning, lighting, and environmental changes. They can oversee building security, fire suppression, and elevator operations. Smart building technologies focus on bringing more detailed monitoring and sensing “awareness” to buildings.

#### 9.4.4 Cyber-Physical System (CPS)

Computers have become pervasive and ubiquitous. Computing devices appear in TV remote controls, smartphones iPods, elevators, escalators, windshield wipers, thermostats in offices/homes, and traffic lights at intersections. These devices are so common in our daily lives that we do not even think of them as computers. Technical people refer to these devices *embedded systems* that perform one or more dedicated functions. With the advent of smartphones, GPS navigators, and tablet computers, embedded systems are turning into a new class of intelligent systems, known as *cyber-physical systems*.

A *cyber-physical system* (CPS) is an embedded system which integrates the computing process with the physical world as an interactive and intelligent system. CPSes appear in many computer and TV game systems. The best example is the popular Nintendo Wii interactive gaming system. Cursor-based CPSes can be found in the automotive, aerospace, health care, robotics, manufacturing, battlefield training, and consumer appliance industries, among others. Thus, a full-fledged CPS involves embedded computers, network monitors, and intelligent control of physical processes with humans in the feedback loop. In the physical world, man-machine interactions must be handled in real time by a CPS. [Figure 9.23](#) shows the abstraction architecture of a typical CPS.



**FIGURE 9.23** Three major components working interactively and intelligently in a CPS.

A CPS features tight coordination between the system's computational and physical elements. Today, most virtual reality systems tend to focus more on computational elements than on the intense link between the computational and physical elements. To amend this shortcoming, a CPS should be designed as a network of interacting elements with physical input and output instead of as a stand-alone device. The notion is extended from the concepts of robotics and sensor networks. The expectation from the CPS

community is to improve the link between computational and physical elements. The design objectives include the adaptability, autonomy, efficiency, functionality, reliability, safety, and usability of future CPSes.

These three components interact with one another heavily. Sensor networks and man-machine interfaces need to be established to glue the three sectors together. In a CPS, the cyberspace and physical worlds are intertwined. The embedded computing, sensed real-world data, and real-time responses are all equally important. New understandings and techniques are needed to deal with this new fusion. New advances are desired to broaden the potential of CPSes in several dimensions. One is human intervention at the right moment, such as collision avoidance in driving a vehicle. Doctors need precision in robotic surgery and workers need nano-level manufacturing.

It is desired to design CPSes to replace humans in dangerous or inaccessible environments such as battlefields, rescue from earthquake rubble, and deep-sea exploration. Users demand close coordination in air traffic control and war fighting and enhanced efficiency and augmentation of human capabilities in health care monitoring and life-saving operations. Since 2006, the U.S. National Science Foundation (NSF) has identified CPS as a key area of research. The NSF and other federal agencies have sponsored several workshops on CPS in recent years.

The applications of CPS typically fall under sensor-based systems and autonomous systems. For example, many wireless sensor networks monitor some aspect of the environment and relay the processed information to a central node. Other types of CPSes include autonomous automotive systems, medical monitoring, process control systems, distributed robotics, and automatic pilot avionics. The following example gives new directions of CPS applications.

### **Example 9.11**

#### **Distributed Robotic Garden Project at MIT**

Researchers involved in a distributed robotic garden project at MIT have experimented on a set of robots to tend a garden of tomato plants. This system combines collaborative work in distributed sensing (each plant is equipped with a sensor node monitoring its status), navigation, manipulation, and wireless networking. The long-term goal is to implement an autonomous greenhouse based on autonomous robots and sensors. Emphasis of the project is on scalability and robustness. The solution was fully distributed with simple individual components.

The MIT research group constructed a mobile end-effector to manipulate the robotic garden. They installed intelligent pots to provide distributed sensing and data storage in the system. They consider plants, pots, and robots to be systems with different levels of mobility, sensing, actuation, and autonomy. The system transforms energy, water, and nutrients into

production. The system studies important problems in multirobot coordination, force-feedback grasping, and spatial computing. The MIT research group envisions the following long-term applications of their CPS work:

- **Sustainable, organic agriculture** Using a large number of robots and embedding intelligence enables large-scale heterogeneous plant populations that are less prone to deplete the soil of specific nutrients and might allow for reduction of pesticides by intelligent local combinations of plants. Also, on-demand watering can reduce water consumption.
- **Autonomous farms** Providing a variety of products on a small scale in a fully autonomous fashion enables long-term deployments of human populations on remote areas and outer space missions.
- **Green architecture** Fruits and vegetables can be cultivated on the inside and outside of buildings where they positively contribute to room climate and energy efficiency, and support the inhabitants' diet.

Many scientific areas could be blended together to realize truly intelligent CPSes in addition to embedded and real-time systems. These areas include avionics, automobiles, power systems, control theory, and electro-mechanical engineering, among others. It has been suggested that CPS be a priority area for R&D in federal networking and information technology in the United States.

## 9.5 Online Social and Professional Networking

Online social networking enables many social and professional functions over the Internet. Through dedicated web sites, P2P connections can be easily established. Social networking has already transformed the way in which people interact in cyberspace. Social networking web sites such as Facebook, Twitter, LinkedIn, and MySpace are popular today. They provide online services that allow users to create online profile pages and make friends with those who are registered on the same web site. According to a recent global Internet traffic report, there were almost 1 billion registered users among the top 15 social networks ([www.ebizmba.com/articles/social-networking-websites](http://www.ebizmba.com/articles/social-networking-websites)).

In early 2011, Facebook was ranked at the top, exceeding 600 million monthly visitors. The connectivity among the social network's users and the community behind each special interest group can be analyzed by large-scale graph theory. A graph-theoretical approach is presented to analyzing social networks. The purpose is to evaluate social network profiles, user demographics, security measures, networking features, and search and technical support provided. As case studies, two social networks are examined: namely *Facebook* for sharing photos and making friends and *Twitter* for microblogging

and exchange of ideas.

## 9.5.1 Online Social Networking Characteristics

*Online social networks (OSNs)* are formed with individuals or organizations over the Internet. These individual or organizational entities are related, connected, or associated special interests or specific interdependencies. These include personal friendships, kinships, professional orientations, common interests, financial exchange, like or dislike relationships, community or racial groups, religious or political beliefs, knowledge or prestige, and celebrity fans. In general, social networks are formed by complex sets of interpersonal to international relationships among members of our societies or communities on Earth.

A social network is a structure representing the social relationships of individuals. In a social network, nodes represent the individuals, and the ties between the nodes represent the relationships such as friendship, kinship, and colleagueship [3]. OSN services are built to reflect the social relationships among people. Social networking services are a communication tool among people. Before these services, online communities that acted as early online social network services did exist—an example is the *bulletin board service (BBS)*. However, traditional online communities are group-oriented, which means users are organized in different groups based on different interests and regions, while modern social networking web sites are always individual-oriented.

### 9.5.1.1 Online Social Networking Services

An OSN offers special social or professional services through Internet access and web services. These social networking services are linked by Internet communication among people. Early online social network services include job hunting, dating, and BBSes. Traditional online communities are organized in different groups based on different interests and regions, while modern social networking web sites are always individual-oriented or follow P2P interactions. Here are some ideas for providing OSN services:

- Personal page or profiles for each user linked by social connections
- Social graph traversal along specific social links or networks
- Communication tools among participants or registered users
- Ability to share music, photos, and videos with friends or professional groups
- Operation of a community in special niche areas like health care, sports, and hobbies
- Customized software tools or databases are used in OSN services
- Strong customer loyalty and fast membership growth are seen
- Provider revenue from embedded advertisement and access to premium content

### 9.5.1.2 Social Networking Properties

In most social networks, each user has a personal page (or profile), which is linked via the user's social relationships. This is why modern social networking web sites are individual-oriented. Once a user accepts other users as "friends," he will be able to access

all permitted contents on those friends' profile pages. Also, users can access their friends' lists of friends, and view their profile pages. This is an example of social graph traversal. Commonly, online social networking web sites provide two basic communication methods: internal e-mail-like direct messages and public message boards. Also, users can share information, such as photos, videos, web links, and articles with friends. All the popular online social networking web sites have these four essential items, but the way they implement these features is different based on the different service models and market orientations.

However, logically the structure of social networks is P2P-based. Modern popular social networks apply mostly a client/server architecture that is open to the general public. This has led to many problems such as privacy violations, service availability, system stability, network scalability, and response performance. The service models and functionalities of Facebook and Twitter are studied. Users need to understand why they are so successful. To discover the problems they have caused in cyberspace, users need to generate effective solutions without degrading network performance.

### **9.5.1.3 Representative Online Social Networks**

Table 9.10 summarizes the top 5 popular OSNs based on the number of registered users. Social networking web sites such as Facebook, Twitter, and MySpace are extremely popular today. Logically, social networking services are P2P networks. However, most of these web sites are installed with a client/server architecture where the large number of servers form the data center. The five attributes of QoS are weighted equally to yield the overall rating with a maximum score of 5 with 5 being a perfect site and 3 being an average site.

---

**Table 9.10**

**Assessment of Top Five Social Networking Web Sites in April 2011**

---

Name (Year)	Brief Description and Web Site	Registered Users	Personal Profiles	Privacy Security	Networking Features	Search Support	Overall Rating
Facebook 2004	Content sharing with large number of apps; <a href="http://www.facebook.com/">www.facebook.com/</a>	600M	4.5	4	5	4.5	4.51
Skype 2003	Video telephony and instant messaging; <a href="http://www.skype.com">www.skype.com</a>	521M	3	4	4	3	3.50
YouTube 2005	Video clips forwarding and sharing web site; <a href="http://www.youtube.com">www.youtube.com</a>	500M	3	3	3	4.5	3.38
Twitter 2006	Micro-blogging, news, and alerts; <a href="http://www.twitter.com">www.twitter.com</a>	175M	3	4	5	5	4.25
MySpace 2003	Customized social services; <a href="http://www.myspace.com">www.myspace.com</a>	130M	5	3	5	4	3.75

Any social network with a rating above 4.0 is considered very good. These attributes could also be weighted differently to yield different ranking results, depending on the application domain applied. This means all the data, including blog entries, photos, videos, and social network relationships, is stored and managed on central server clusters owned by the service providers. Although the traditional architecture has advantages in scalability, because users can access their data from any client including fixed machines or mobile devices, there are some implementation problems too. For instance, private personal information could be lost, and large server clusters need to be monitored during usage rush hours.

#### 9.5.1.4 Online Social Networks

A typical OSN platform architecture. It is desirable to customize an OSN to sustain competition in this field. The social network provider should choose a brand name with its own API interfaces and profile variables. The chosen forum categories must be relevant to the sufficiently large user community. The OSN platform must include specific functionalities that make it easy for users to join and enjoy the services. Furthermore, the provider must prove the online marketing concept to attract members to join and leave freely. Highly sophisticated software, virtualized data centers, or processing and storage cloud platforms are needed.

As an on-demand service provider, the OSN platform should be appealing to all Web 2.0 communities. The social network community must operate reliably with high availability and performance. Logically, the OSN provides a P2P platform. However, modern popular social networking services are all built with the client/server architecture

for easy management and maintenance. This means that all blog entries, photos, videos, and social network relationships are stored and managed by private clouds owned by the service providers. A large social network site must maintain huge data centers. To serve clients better, many of the data centers are virtualized to provide cloud services.

### 9.5.1.5 Social Network Benefits

Editorial web sites often include a forum already immersed in Web 2.0 services. Forums are valuable but lack the attraction to deliver the same level of value as social network communities. A full-blown social network community can be built around a forum to result in the following four operational benefits:

- **High return visit rate** Users return to the social network community frequently. This opens up the opportunity for great page impressions and a huge advertising inventory.
- **User loyalty** Users connect to their friends and will not abandon them easily. They do not move over to a competing social network. Instead, they show high customer loyalty.
- **Virtual growth** Members invite their friends to the social network community. This is effective marketing at a low cost, and the OSN grows by itself.
- **Business model** With a social network community you can earn revenues through subscriptions to premium content in addition to advertising revenues.

Twitter does not completely copy all the offline relationships onto the web sites. Hence, the attraction to use Twitter is far less than Facebook at present (2010). On the other hand, Facebook is not as open as Twitter. These features make people trust Facebook more than Twitter. Last but not least, Facebook is more complex in functionality than Twitter. Although Twitter has plenty of third-party applications, Twitter is still not convenient for beginning users. Facebook has embedded many common features. If someone does not like to explore third-party applications for some common functions, she would go to Facebook instead of Twitter, based on the current trend.

### 9.5.2 Graph-Theoretic Analysis of Social Networks

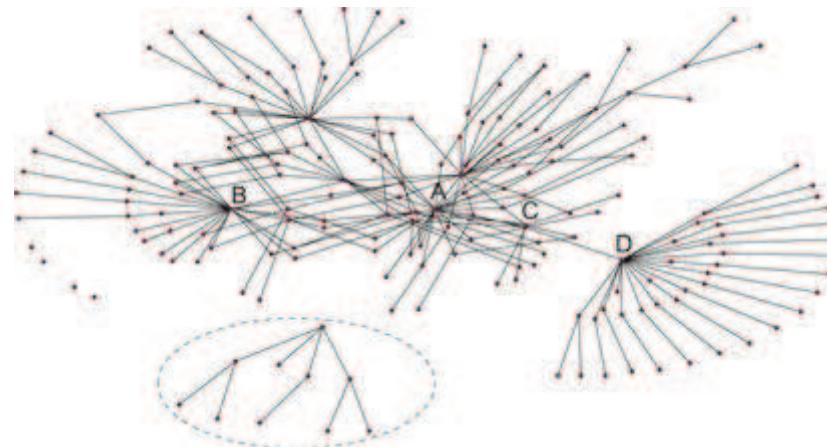
Social network analysis has emerged as a key technique in modern sociology. It has also gained a lot of attention in anthropology, biology, communication, economics, geography, information science, organizational studies, social psychology, and sociolinguistics. Characterizing the existing relationship among a person's social group is the main task in social network analysis. Also, users are facing a so-called "small-world" society, where all people are related in a short chain of social acquaintances, one way or another. All social networks are not so chaotic or random as once assumed, but rather have underlying structures. Some of the social networking rules or behaviors are analyzed in this section.

Social relationships are often mapped into directed or undirected graphs, sometimes called *acquaintance graphs* or simply *social connection graphs*. The nodes in a social graph

correspond to the users or actors and the graph edges or links refer to the ties or relationships among the nodes. The graphs can be complex and hierarchically structured to reflect relationships at all levels. There can be many kinds of ties between the nodes. Social networks operate from the family level up to national and global levels. There are pros and cons of social networks. Most free societies welcome social networks. For political or religious reasons, some countries block the use of social networks to prevent possible abuses.

### 9.5.2.1 Social Network Graph Properties

Social networks play a critical role in problem solving, running an organization, and the degree to which individuals succeed in achieving their goals. A social network is simply a map of all of the relevant ties between all the actor nodes. The network can also be used to measure social capital—the value that an individual gets from the social network. These concepts are often displayed in a *social network graph*. Figure 9.24 shows an example social network graph. The black dots are the nodes (users) and the lines link the nodes under specified relationships.



**FIGURE 9.24** A graph representation of an example social network.

### 9.5.2.2 Node Degree, Reach, Path Length, and Betweenness

The *node degree* is the number of ties of a node to other actors in the graph. The *reach* is defined as the degree any member of a network can reach other members of the network. *Path length* measures the distance between pairs of nodes in the network. *Average path length* is the average of these distances between all pairs of nodes. *Betweenness* reveals the extent to which a node lies between other nodes in the network. The measure reflects the number of people with whom a person is connecting indirectly through his direct links.

### 9.5.2.3 Closeness and Cohesion

*Closeness* is the degree to which an individual is near all other individuals in a network (directly or indirectly). It reflects the ability to access information through network

members. Thus, closeness is the inverse of the sum of the shortest distances between each individual and every other person in the network. *Cohesion* is the degree to which actors are connected directly to one another by cohesive bonds. Groups are identified as “cliques” if every individual is directly tied to every other individual.

#### **9.5.2.4 Social Circles or Clusters**

This refers to some structured groups. If there is less stringency of direct contact or as structurally cohesive blocks, a social circle can be created either loosely or tightly, depending on the stringency rules applied. Those nodes inside the circle of [Figure 9.24](#) form an isolated cluster. *Clustering coefficient* is the likelihood that two associates of a node are associates themselves.

#### **9.5.2.5 Centralized versus Decentralized Networks**

*Centrality* gives a rough indication of the social power of a node based on how well it “connects” the network. *Betweenness*, *closeness*, and *degree* are all measures of centrality. *Centralized networks* have links dispersed around one or a few nodes, while a *decentralized network* is one in which there is little variation between the number of links each node possesses.

#### **9.5.2.6 Bridge and Local Bridge**

An edge is a *bridge* if deleting it would cause its endpoints to lie in different clusters or components of a graph. For example, the edge between nodes C and D in [Figure 9.24](#) is a bridge. The endpoints of a *local bridge* share no common neighbors. A local bridge is contained in a cycle.

#### **9.5.2.7 Prestige and Radiality**

In a social graph, *prestige* describes a node’s centrality. *Degree prestige*, *proximity prestige*, and *status prestige* are all measures of prestige. *Radiality* is the degree to which a network reaches out and provides novel information and influence.

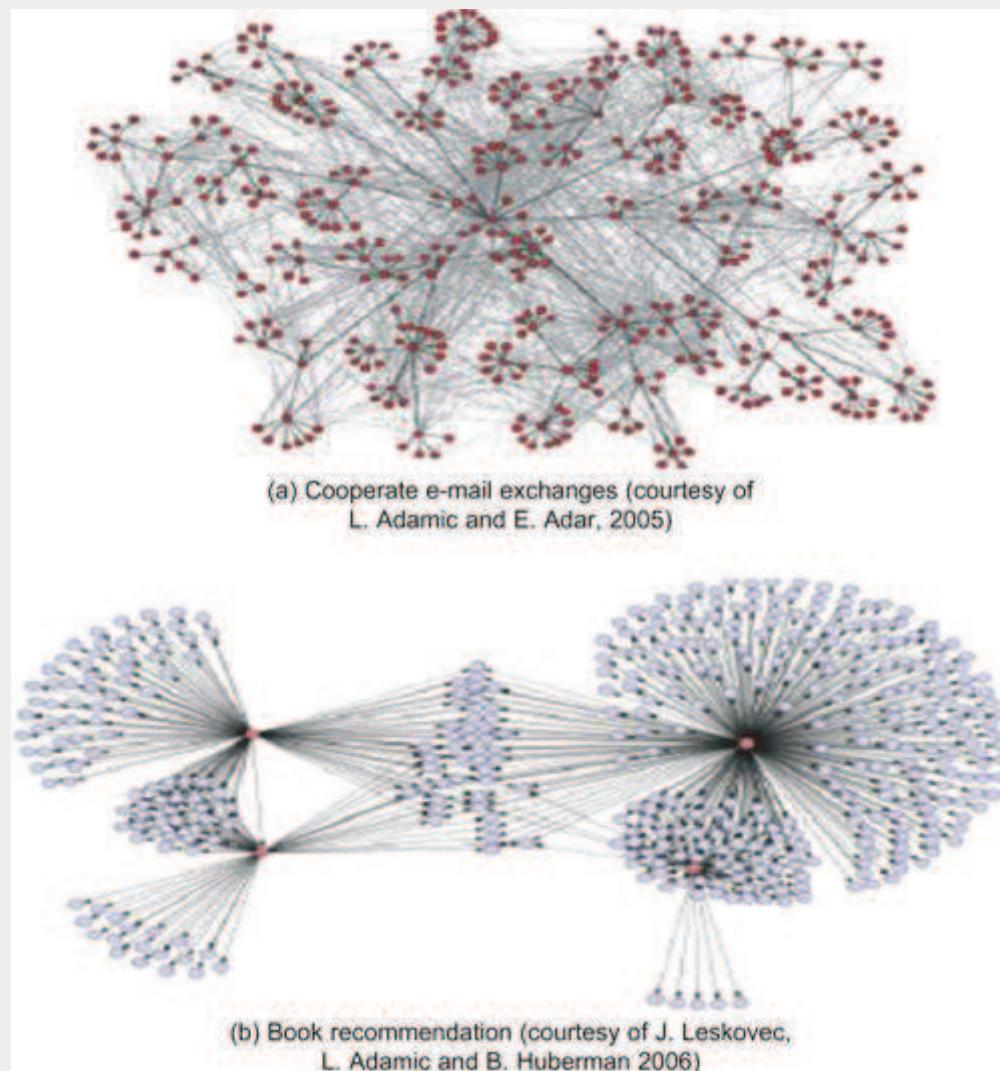
#### **9.5.2.8 Structural Cohesion, Equivalence, and Holes**

*Structural cohesion* is the minimum number of members who, if removed from a group, would disconnect the group. *Structural equivalence* refers to the extent to which nodes have a common set of linkages to other nodes. These nodes do not have any ties to one another. A *structural hole* can be filled by connecting one or more links to reach other nodes. This is related to social capital: By linking two disconnected people, you can control their communication.

#### **Example 9.12**

## Some Graph Representations of Sample Social Networks

Figure 9.25 shows four sample social networks. In part (a), the graph corresponds to a 436-node network of e-mail exchanges at a corporate research laboratory. However, social network data spans many orders of magnitude. For example, for a large university, the e-mail exchange network could end up with a 43,553-node network over a two-year period. Counting all instant messages over one month at MSN, one may end up with 240 million connections. The analysis of large-scale social networks may enable us to observe global behavior. In Part (b), we show the network of book recommendations. Jon Kleinberg [44] has identified many epidemic graphs on news, opinions, political mobilization, etc. Models for cascading behavior is a popular topic in social networks.



**FIGURE 9.25** Graph representations of two social networks.

## 9.5.3 Communities and Applications of Social Networks

Niche communities become increasingly important and are highly valuable, particularly for special interest magazines, radio stations, sports web sites, sports clubs, health web portals, private clinics which are specialized in certain health topics, industry associations, and companies that are looking for new ways to interact with their customers or employees. This section discusses a few social network communities.

### 9.5.3.1 Communities in Social Networks

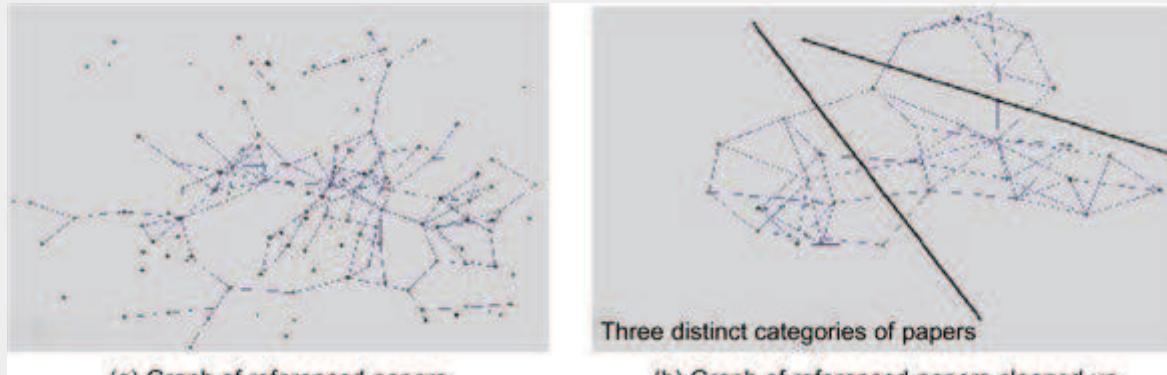
Very large-scale social networks result in very large-scale graphs with millions or hundreds of millions of user nodes. Such large graphs can often be partitioned into many smaller subgraphs, corresponding to various actor communities. Six specialized social network communities are pointing to different sectors of our society. This list only offers a few examples. One can form many other communities from online social networking.

- **Industry communities** Special industrial workers or professionals are often connected with one another. They share knowledge and work experience.
- **Artist communities** These network communities are specifically composed to enable artists, musicians, or celebrities to personalize and intensify their contact with their existing fans as well as enabling contact among community members.
- **Sport communities** These are network communities for special interests and activities of athletes and sport fans. People can find friends, celebrate their passion, and exchange ideas.
- **Health communities** These are dedicated to the needs of actors concerned about health issues.
- **Congresses and event communities** These are customized to support all preparations necessary for congresses and events, as well as all processes thereafter.
- **Alumni communities** After completing their studies, alumni can find fellow students, stay in touch, and foster friendships.

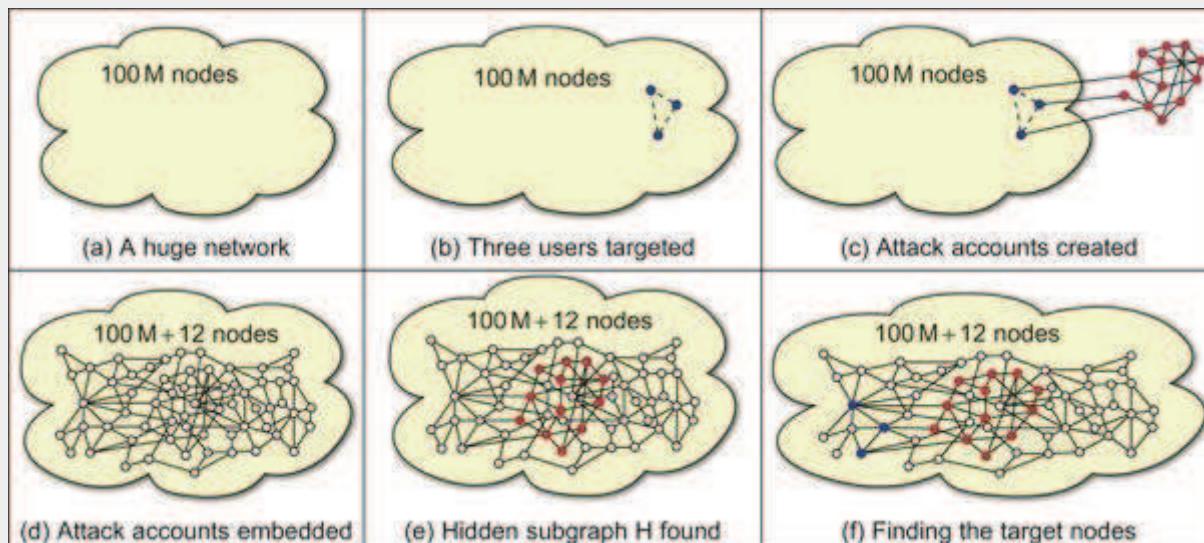
#### Example 9.13

#### Citation Graph to Partition Research Communities

In a 2010 talk at USC, John Hopcroft [40] identified the use of large-scale graph theory to analyze the impact of online social networking. He identified the merging of computing and communications in a ubiquitous fashion and the explosive growth of digital data from networked devices and sensors. Accordingly, Internet search engines are changing. Internet queries should be able to address high-level questions such as “where I should I go to college?” in a faster, more personalized, and more intelligent way. He used an example (in Figure 9.26) to illustrate the tracking of the flow of ideas in scientific literature and evolution of communities in social networks.



**FIGURE 9.26** Partitioning in paper citation graph to identify research communities. Courtesy of Yookyoung Jo, Cornell University



**FIGURE 9.27** Attack tracking in an anonymized social network. Courtesy of Jon Kleinberg [44]

Figure 9.26(a) shows a citation graph of referenced papers by authors working in related areas. The dots represent individual papers. The edges link the papers if they reference each other one way or mutually. Through some cleaning process, certain papers are merged due to self-citation or other concerns. The reduced citation graph in Figure 9.26(b) is obtained with referenced papers cleaned up. One can partition the cleaned graph into three categories of papers that are related to a similar approach or based on the same solution model.

In tracking communities in social networks, many authors have revealed interesting statistical properties of community structure in large social and information networks. Jure Leskovec, et al. [49] have indicated that the best communities are size 100–150. This data is drawn from the study of 70 large, sparse, real-world social networks. Such small communities are formed with special relationships such as family and friends,

classmates, and work colleagues. These communities may overlap in many ways. The peers could be connected inside or outside the community boundary. He has revealed the fact that there are more connections outside than inside for most people, which is intuitively valid based on the common experiences of many of us. Other interesting observations include the existence of a core in social networks. Often, the core can be either unique or shared.

### 9.5.3.2 Social Network Application Domains

There are an infinite number of applications for social networks. In Table 9.11, we classify those applications into five specific domains. Examples and their web sites are listed for our readers to explore. The use of social network services in an enterprise context presents the potential of having a major impact on the world of business and work. Social networks connect people at a low cost; this can be beneficial for entrepreneurs and small businesses looking to expand their contact bases. These networks often act as a customer relationship management tool for companies selling products and services. Companies can also use social networks to advertise. With global operations, social networks can make it easier to promote business around the world.

**Table 9.11**  
**Applications of Existing Social Networks**

Domains	Examples and Web Sites
Business	LinkedIn connects professionals ( <a href="http://www.linkedin.com">www.linkedin.com</a> ) and Hub Culture connects entrepreneurs ( <a href="http://www.hubculture.com">www.hubculture.com</a> ).
Education	The National School Boards Association and education topics can be researched online ( <a href="http://www.ning.com">www.ning.com</a> ).
Government	The U.S. Centers for Disease Control and Prevention demonstrated the importance of vaccinations on <a href="http://www.whyville.net">www.whyville.net</a> , a popular children's site. The National Oceanic and Atmospheric Administration has a virtual island on <a href="http://secondlife.com">http://secondlife.com</a> , where people can explore underground caves or explore the effects of global warming.
Medical and health care	<a href="http://www.patientslikeme.com">www.patientslikeme.com</a> offers its members the chance to connect with others dealing with similar issues and research patient data related to their condition. <a href="http://www.sobercircle.com">www.sobercircle.com</a> , which has joined forces with <a href="http://www.onerecovery.com">www.onerecovery.com</a> , gives people in recovery the ability to communicate with one another and strengthen their recovery.
Online dating	Online social networking sites such as Facebook, Second Life, and MySpace are quickly becoming the new way to find dates online.

One example of social networking being used for business purposes is [LinkedIn.com](http://LinkedIn.com), which aims to interconnect professionals. Another is the use of physical spaces available to members of a social network. Some hub culture extends invitations only to entrepreneurs and other business successors to join a social network. Having a physical presence allows members to network in the real as well as virtual worlds, adding extra business value. Applications for social networking sites have extended to businesses, and brands are creating their own sector known as brand networking, which is based on the idea that a brand can build its consumer relationship by connecting consumers to the brand image on a platform.

### 9.5.3.3 LinkedIn for Professional Networking

Social networks for professional connections include [LinkedIn.com](#), [Xing.com](#), and [Ecademy.com](#). LinkedIn is a business-oriented social networking site. LinkedIn's main focus is to manage and build online and/or offline professional networks. LinkedIn has more than 80 million members in more than 200 countries. A new member joins LinkedIn approximately every second, and about half of the members are outside the United States. Executives from all Fortune 500 companies are LinkedIn members. This site allows registered users to maintain a list of contact details of people they know and trust in business. Users can invite anyone to become a member.

#### **9.5.3.4 Educational Applications**

The National School Boards Association reports that almost 60 percent of students who use social networking talk about education topics online and, surprisingly, more than 50 percent talk specifically about schoolwork. Yet the vast majority of school districts have stringent rules against nearly all forms of social networking during the school day—even though students and parents report few problem behavior online. Social networks focused on supporting relationships between teachers and between teachers and their students are now used for learning, educator professional development, and content sharing. Ning for teachers, Learn Central, [60] TeachStreet, and other sites are being built to foster relationships that include educational blogs, e-portfolios, formal and ad hoc communities, as well as communication such as chats, discussion threads, and synchronous forums.

#### **9.5.3.5 Government Applications**

Social networking tools serve as a quick and easy way for the government to get the opinion of the public and to keep the public updated on their activity. The U.S. Centers for Disease Control and Prevention demonstrated the importance of vaccinations on the popular children's site Whyville and the National Oceanic and Atmospheric Administration has a virtual island on Second Life where people can explore underground caves or explore the effects of global warming. Similarly, NASA has taken advantage of a few social networking tools, including Twitter and Flickr. They use these tools to promote a vigorous and sustainable path to achieving its boldest aspirations in space.

#### **9.5.3.6 Health Care and Medical Applications**

Social networks are beginning to be adopted by health care professionals as a means to manage institutional knowledge, disseminate P2P knowledge, and highlight individual physicians and institutions. The advantage of using a dedicated medical social networking site is that all the members are screened against the state licensing board list of practitioners. The role of social networks is especially of interest to pharmaceutical companies who spend approximately 32 percent of their marketing dollars to influence the opinion leaders of social networks.

A new trend is emerging with social networks created to help members with various

physical and mental ailments. For people suffering from life-altering diseases, PatientsLikeMe offers its members the chance to connect with others dealing with similar issues and research patient data related to their condition. For alcoholics and addicts, SoberCircle gives people in recovery the ability to communicate with one another and strengthen their recovery through the encouragement of others who can relate to their situation. DailyStrength is also a web site that offers support groups for a wide array of topics and conditions, including the support topics offered by PatientsLikeMe and SoberCircle. SparkPeople offers community and social networking tools for peer support during weight loss.

### **9.5.3.7 Dating Services**

Many social networks provide an online environment for people to communicate and exchange personal information for dating purposes. Intentions can vary from looking for a one-time date to looking for a short-term or long-term relationship. Most of these dating services require users to give out some personal information such as age, gender, location, interests, and perhaps a picture. Releasing personal information is often discouraged for safety reasons. This allows other users to search or be searched, but at the same time people can maintain a degree of anonymity. Online dating sites are similar to social networks in the sense that users create profiles to meet and communicate with others.

Online dating sites usually require a fee, where social networks are free. This difference is one of the reasons the online dating industry is seeing a massive decrease in revenue due to many users opting to use free social networking services. Many popular online dating services such as [Match.com](#), Yahoo! Personals, and [eHarmony.com](#) are seeing a decrease in users, where social networks such as MySpace and Facebook are experiencing an increase in users. The number of Internet users in the U.S. that visit online dating sites has fallen from a peak of 21 percent in 2003 to 10 percent in 2006. In other words, social networks are gradually taking over most of the dating services.

### **9.5.3.8 Privacy Breach in Social Networks**

Many people are reluctant to join online social networks for worry about privacy breaches. Still, the growth of social network members is rather impressive, regardless of the fact of serious abuses reported every day. For example, many people charge that MyPlace has made public what should be private matters. As observed by Jon Klienberg, the software we use has a trace of our activities. Anonymization may not be totally effective in protecting sensitive data in massive social networks. The following example explains the scenario. Active attacks can easily compromise privacy by creating new accounts in practice. We need better algorithmic ideas to cope with this problem.

#### **Example 9.14**

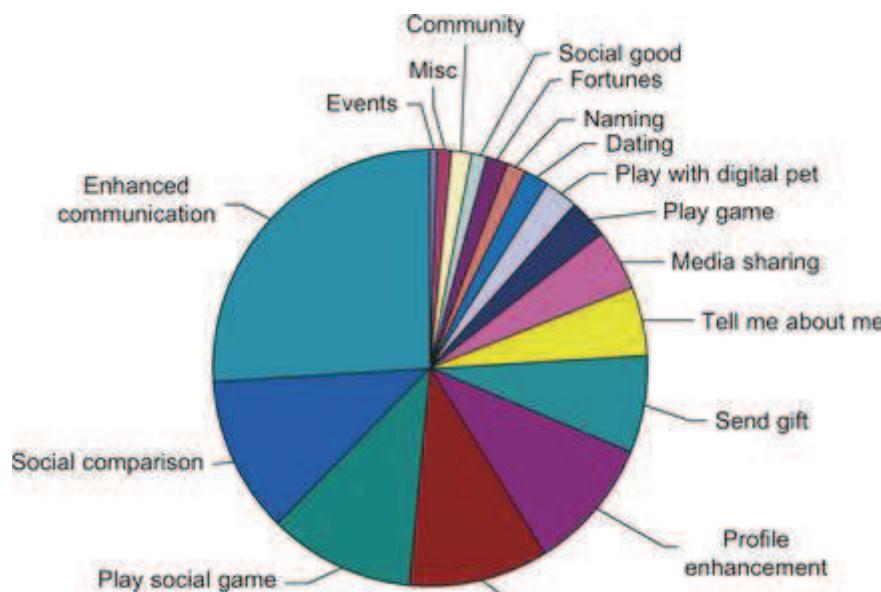
## Attacking an Anonymized Social Network (Jon Kleinberg [44])

Consider the attack scenario: By accident, an organization released an anonymized communication graph on 100 million users, as shown in [Figure 9.27\(a\)](#). An attacker chooses a small set of user accounts to dig into their private relations or activities, as shown in [Figure 9.27\(b\)](#). Before the data set is released, the attacker may create a small set of new accounts with links among them to form a subgraph  $H$  attached to the targeted accounts, as shown in [Figure 9.27\(c\)](#). When the data set is released, the organization wants to find  $H$ . This is a computationally hard problem, as shown in [Figure 9.27\(d\)](#).

In fact, Kleinberg indicated that the small random graph  $H$  will likely be unique and efficiently findable as shown in [Figure 9.27\(e\)](#), because each edge has a probability of 0.5 percent to be present on a random graph. Once  $H$  is found, one can find the targeted nodes by following the edges from  $H$  in [Figure 9.27\(f\)](#). One can breach privacy with  $O((\log n)^{1/2})$  new nodes. The company should avoid releasing an anonymized social network. Privacy-preserving mechanisms are still very much in demand to protect users from such attacks.

### 9.5.4 Facebook: The World's Largest Social Network

Facebook was launched by Mark Zuckerberg in 2004. Initially, Facebook was only available to Harvard University students. Today, anybody older than 13 years of age and with a valid e-mail address can register on Facebook. In March 2011, it was reported that Facebook had 600 million active users and a monthly growth rate of 3.57 percent [\[88\]](#), making Facebook the world's largest social network. [Figure 9.28](#) shows various services provided.



**FIGURE 9.28** Distribution of 28.4 million Facebook applications, averaged for the week ending November 8, 2007.

Source: The Facebook Application Ecosystem: Why Some Thrive- and Most Don't An O'Reilly Radar Report by Shelly D. Farnham, Ph.D  
Copyright © 2008 Waggle Labs All rights reserved. Used with Permission

The largest sector of applications is to enhance communications among the users. Social comparison is welcome by job seekers. Playing social games is popular among the youngsters and retirees. Social selection, sending gifts, and profile enhancement facilitate more frequent social contacts. The tell-me-about-me, media sharing, playing, naming, dating, and social events are more traditional areas. This distribution varies with time. Once the privacy and security issues are resolved for social networks, the number of registered users will grow even more in the future.

### 9.5.4.1 Functionality of Facebook

Facebook includes six essential items, as shown in [Table 9.12](#). Facebook needs to improve in terms of privacy. Compared with Twitter, Facebook has stricter access control, and thus is more secure to use. However, one must be aware that all personal data is stored on the provider's servers, including education and career information as well as personal messages and social links. Security and trust are the major concerns that prevent users from joining or staying active on Facebook. Users are willing to share more information because they like to communicate with their real-life friends.

**Table 9.12**

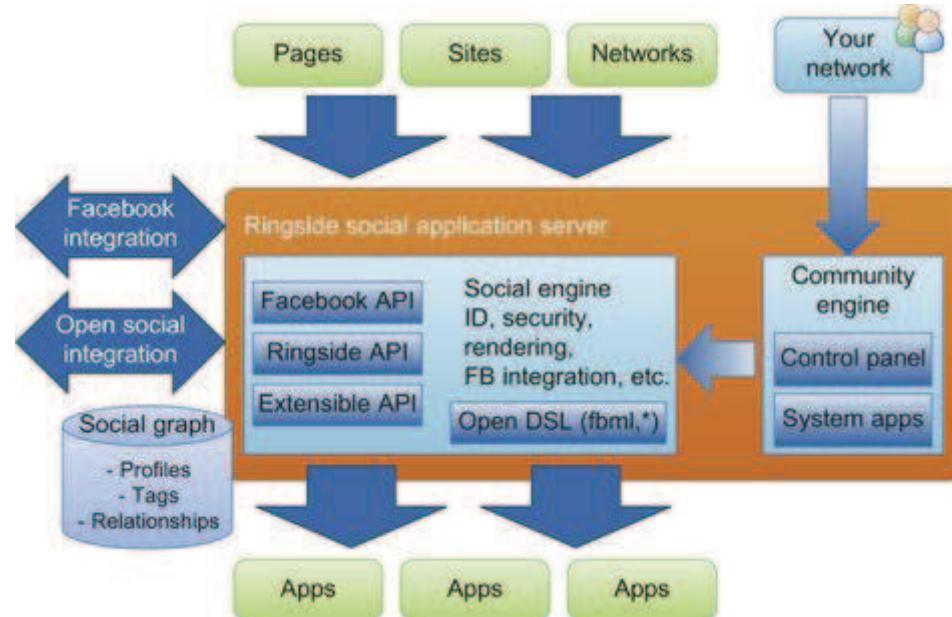
#### Functionality of Facebook Features

Function	Implementation
Profile page	Combined profile: profile picture, bio information, friends list, user's activity log, public message board, other components selectively displayed
Social graph traversal	Access through user's friends list on profile pages, with access control
Communication tools	Internal e-mail-like message: send and receive private messages among friends; instant messaging: accessed on the web page, or through third-party client; public message board: "Wall," with access control; update status: a short message, like micro-blogging, with access control
Shared information	Photo album: built-in, with access control; links: post links to outside URL, will appear on the activity log; videos: embedded outside videos on profile page
Access control	Every item on the profile page can be set to one of four access control levels: only me, only friends, friends of friends, or everyone
Special APIs	Games, calendars, mobile clients

Without registration constraints, maintaining trust among Facebook users is a serious problem. Facebook uses a network with authentication to authenticate users. A *network* is a subset of the users on Facebook with certain authentications. For example, to join a university network, one may have to register with a university e-mail address, or follow some other authentication method approved by the network administrator. The friends of this user could identify the user by his profile in the network.

## 9.5.4.2 Architecture of the Facebook Platform

With 600 million active users, you can imagine how many personal profiles and multimedia information items are shared by various social groups on Facebook. You can also imagine how much user traffic is flowing into the Facebook web site ([www.facebook.com](http://www.facebook.com)). The Facebook platform is essentially a huge data center with a very large storage capacity, intelligent file systems, and searching capabilities. [Figure 9.29](#) illustrates the structure of the Facebook platform.



**FIGURE 9.29** The Facebook architecture built with social engine servers and Application Programming Interfaces (APIs). Courtesy of Facebook, Inc. 2010

The platform is formed with a huge cluster of servers. These machines are called *ringside social application servers*. The *social engine* is the core of the server, which performs the functions of identification, security, rendering, and Facebook integration. Three specific APIs are installed to facilitate user access. A *community engine* provides networking services to users. Requests are shown as pages, sites, and networks entering the Facebook server from the top. The social engine executes all sorts of user applications. Open DSL is used to support application execution.

## 9.5.4.3 Some New Facebook Applications

Successful Facebook applications help users to achieve such social goals as improved communication, learning about themselves relative to others, finding people with similar interests, improving self-presentation, engaging in social play, and engaging in social exchanges via gifts and media. Facebook has greater appeal in the private and personal domain, as opposed to professional connections.

In late 2010, Mark Zuckerberg announced a new form of messaging over the Internet that could trigger a revolutionary change in Internet communication among trusted friends and social circles. He feels that the traditional e-mail for epistolary interpersonal

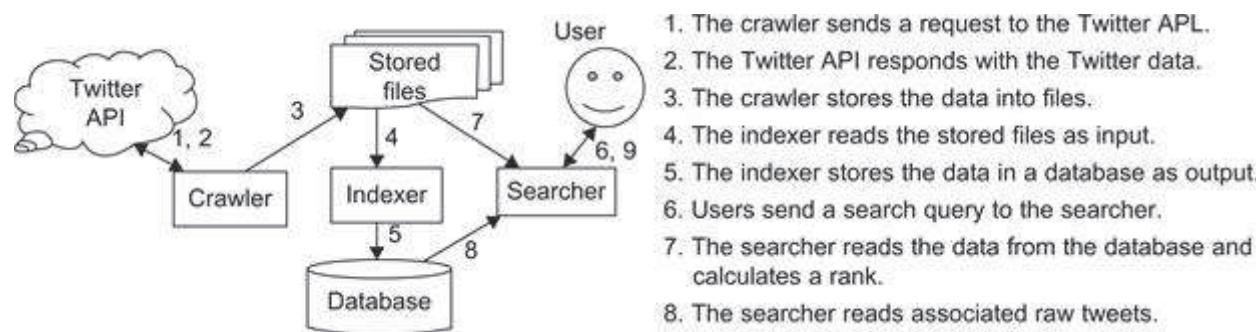
communication is outmoded. He proposes a *social inbox* concept where users can readily access messages from friends and then sort them based on personal preference. This is something between instant messaging and Twitter. The more we text or Twitter or make friends through social networking, the less likely we want to express ourselves in complex ways. In other words, simplicity and convenience will dominate the future demands of online social networking.

## 9.5.5 Twitter for Microblogging, News, and Alert Services

Twitter is a microblogging service launched in 2006 by Jack Dorsey, Biz Stone, and Evan Williams. Each message sent through Twitter can be no more than 140 characters in length. This idea originated from *Short Message Service* (SMS) technology over mobile devices such as smartphones. A user can use SMS to send a message to Twitter, and the message will be forwarded to many other Twitter users that are related to the sender. Twitter was built to share thoughts or ideas instantly. But gradually major TV networks and newspapers also started to use Twitter to provide updated news to the public. Twitter has become a media platform in which ordinary users can be in the front line of breaking news.

### 9.5.5.1 Twitter Architecture and Access Control

There are three major components in the Twitter system: *crawler*, *indexer*, and *searcher*. Each component differs in the functions it performs. They are used together through balanced coordination to form a complete system. Figure 9.30 shows the Twitter access architecture in eight steps. [Twitter.com](#) started as a status sharing site that people accessed via their mobile phones. Twitter soon became a Web 2.0 microblogging site for information sharing and news reporting. As of late 2010, Twitter had 175 million users. Furthermore, HubSpot reported that Twitter was enjoying an astonishing 18,000 percent growth rate.



**FIGURE 9.30** Twitter architecture and access protocol sequence. Courtesy of Hao Song, USC 2009

### 9.5.5.2 Twitter Development Trends

Twitter's success can be attributed to two main factors: elegance in design, and simplicity

in adding third-party improvements. Twitter calls microblog posts from users *tweets*. The elegance is due to the character limit in tweet messages. Each tweet has a 140-character limit which is inherited from text messaging. A comparison between blogging and microblogging gives us a good understanding behind Twitter's popularity. While blogging requires good writing skills and lots of content to fill pages, Twitter restricts posts to 140 characters, which encourages many people to post.

Simplicity is provided with an HTTP-based open source API and sharing posts with third-party applications. Twitter's API consists of two different parts: a Search API and a REST API. The REST API enables Twitter developers to access core Twitter data. This data includes tweets, timelines, and user data. The Search API enables developers to query the tweets. It also provides information about trending topics. The usage of both APIs is subject to rate limiting. Twitter may add these applications to its white list and remove request limitations. The ease and flexible usage of Twitter APIs have encouraged many developers to write Twitter applications.

### 9.5.5.3 Functionalities of Twitter

Twitter provides six essential functions in [Table 9.13](#), two of which are access control and API applications. Simplicity and openness are the keys to Twitter's popularity. The core service of Twitter is very simple: just sending messages to a group of people. All other uses are contributed by third-party developers. Twitter provides a very open API, so third-party developers can create powerful clients or special applications for Twitter. The connections on Twitter are not offline social relationships. Users can follow someone they do not know in real life. Not everyone on Twitter is using his real name, and even when real names are used, there is no effective way to identify users. Ever since launched in 2009, Twitter has provided a verified-account service for celebrities and official agencies. However, the revealing of personal information on Twitter still poses a privacy problem.

**Table 9.13**  
**Functions of Twitter and Their Implementations**

Function	Implementation
Profile page	Simple profile: profile picture, bio information, followers list, following list, user's message timeline
Social graph traversal	Access through the following and follower's lists on users' profile pages, without access control
Communication tools	Internal e-mail: send private direct messages to followers; instant messaging: not really instant, can be implemented by direct messages; public message board: message timeline, with access control
Share information	Photo album: third party, like Twitpic, but no combined access control; links: post links to outside URL, will appear as a message on timeline
Access control	Message timeline on profile page can be set access control in two levels: private or public
API applications	Mobile clients, PC clients, photo sharing, automatic news feed

Twitter cannot be regarded as a fully trusted network or a reliable acquaintance basis.

Consider two types of accounts: *public* and *protected* accounts. Anyone can see updates at public accounts. For protected accounts, only the users that have been approved can see their updates. In order to prevent identity confusion, Twitter selects accounts from celebrities to provide verification. The verified notice is posted on verified accounts. Twitter provides two new features, *Twitter Lists* and *Location*, to support this verification process. Twitter Lists allow users to create a list of Twitter accounts. The list can help users organize the people they follow. The lists can also support location-based services. Location-sensitive applications like those aided by GPS devices can be greatly simplified with Twitter services.

#### **9.5.5.4 Twitter Applications**

Many interesting Twitter applications exist. For instance, one can use the public to help determine business or fashion trends and identify popular music and movies. Twitter was even used during the 2008 presidential campaign in the United States to attract young voters to spread then-candidate Barack Obama's "Change" message. Twitter also gives users the option to send their current locations along with their tweets. Twitter provides the Twitter API for developers to request Twitter data. The API supports REST calls.

Two types of APIs are supported: *normal APIs* and *streaming APIs*. Streaming APIs are for real-time access to the public statuses over a persistent connection. @anywhere offers a service that developers use to integrate external parties with Twitter. Lastly, Twitter also collects information about programming libraries that help create Twitter-based applications. Two specific Twitter applications are described in the following list:

- **Fast News Release** Twitter is becoming the fastest way to spread breaking news. Massive user collaboration has given Twitter a clear edge over most news centers. Some news centers have set up Twitter accounts to encourage users to spread breaking news. For example, CNN maintains 45 official Twitter accounts with more than 5 million followers. During the election periods in various countries, Twitter attracted more attention than news centers. Some governments even warned [Twitter.com](#) not to enter maintenance cutoffs in order to receive continuous information flow. Information to Twitter may consist of photos, links, and videos that arrive much faster than via conventional news media.
- **Alert Systems** Twitter provides a system that can connect residents of a city with virtually no cost. It also increases the capabilities of an alert system by inputting more user-generated data. Some cities have already opted for Twitter to alert their residents. The Virginia Tech killings in 2007 highlighted the security issues on university campuses. To integrate the e2Campus emergency notification network with popular social networks, Pacific University of Forest Grove, Oregon, implemented a Twitter-based alert system for its students, and the trend is likely to grow.

### **9.6 Bibliographic Notes and Homework Problems**

Wide coverage of the principles and paradigms of cloud computing are provided in [8].

Cloud design and security issues are covered in [70]. Buyya, et al. [9] have introduced a market-oriented cloud computing framework. Basic cloud research can be found in [1,34,59–62,69–71,94]. Cloud computing applications are studied in [28,33,48,56,59,61,63,67,68,93]. Science and research clouds are studied in [13,15,19–23,30,35–39,52,64,76,79]. Trust models for clouds are proposed in [12,24,42]. Cloud support for ubiquitous computing and mobile applications can be found in [6,8,16,74]. Public clouds are covered in [6,8,31,34,48,65] and private clouds in [11,72]. Cloud mashups are studied in [18,88]. Google’s experiences on MapReduce and BigTable are reported in [14].

Bryant introduced the DISC paradigm in [5]. The SGI Cyclone was reported in [3]. Data coloring and cloud watermarking are studied in [42]. Grid and cloud performance, simulation, and benchmark studies can be found in [9,25,33,50,51,67,68,75,89,90,93]. For science clouds and applications, readers are referred to FutureGrid [30], Grid’5000 [35], Magellan [56], Open Circus and Open Cloud Testbed [64], Science Clouds [76], Sky Computing [43], and Venus-C [79]. Amazon benchmarking results are reported in [31,32,75,80]. Cloud security is treated in [12,42,54,55,57,70,83]. Operating system support for cloud computing can be found in [60,62,72]. Parallel benchmarks and performance evaluation of clusters, MPPs, and distributed systems can be found in Hwang and Xu [41].

The Internet of Things was treated in [45,73,78]. RFID is treated in [2,81,91,92]. Material on sensors and ZigBee technologies can be found in [73] and GPS in [78]. Supply chain management is treated in [91]. The CPS material is based on [17,47,85]. Online social networks are introduced in [4,7,53,86]. Social network analysis is given in [10,29,40,42,46,49,53,58,82,84]. Facebook is reported in [23,26,27]. Twitter is reported in [16,42,66,77]. P2P technology for trust management and social networking can be found in [12,24,43,51,54]. Kleinberg analyzed large-scale social networks in [44]. Reed assessed clouds, many cores, and clusters in [69].

## Acknowledgements

Kai Hwang of USC is the primary author of this chapter. He was assisted by Zhongyuan Qin, Kaikun Dong, Vikram Dixit, and Hao Song of the University of Southern California. Judy Qiu of Indiana University has contributed Sections 9.2.1 and 9.2.5.

## References

1. M. Armbrust, A. Fox, R. Griffith, et al., Above the Clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28, University of California at Berkley, 10 February 2009.
2. C. Bardaki, P. Kourouthanassis, RFID-integrated retail supply chain services: Lessons learnt from the SMART project, in: Proceedings of the Mediterranean Conference on Information Systems (MCIS 2009), Athens, Greece.
3. M. Barrenechea, SGI CEO. HPC Cloud-Cyclone. [www.sgi.com/cyclone](http://www.sgi.com/cyclone), 2010.

4. Bishop J. *Understanding and Facilitating the Development of Social Networks in Online Dating Communities: A Case Study and Model* 2008; [www.jonathanbishop.com/Library/Documents/EN/docSNCEDS\\_Ch15.pdf](http://www.jonathanbishop.com/Library/Documents/EN/docSNCEDS_Ch15.pdf).
5. R. Bryant, Data Intensive Supercomputing: The Case for DISC, Technical Report, CMU CS-07-128, <http://www.cs.cmu.edu/~bryant>, 2007.
6. J. Brodkin, Ten Cloud Computing Companies to Watch. Network World. [www.cio.com/article/print/492885](http://www.cio.com/article/print/492885), 2010.
7. S. Buchegger, A. Datta, A case for P2P infrastructure for social networks: Opportunities, challenges, in: Sixth International Conference on Wireless on-Demand Network Systems and Services (WONS 2009).
8. Buyya R, Broberg J, Goscinski A, eds. *Cloud Computing: Principles and Paradigms*. New York: Wiley Press; 2011.
9. Buyya R, Yeo C, Venugopal S. Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. In: *10th IEEE International Conference on High Performance Computing and Communications*. September 2008.
10. Carrington P, ed. *Models and Methods in Social Network Analysis*. Cambridge University Press 2005.
11. CERN VM download. <http://rbuilder.cern.ch/project/cernvm/build?id=81>, 2010.
12. Chen K, Hwang K, Chen G. Heuristic discovery of role-based trust chains in P2P networks. *IEEE Trans Parallel Distrib Syst*. 2009;83–96.
13. Condor Cloud Computing. [www.cs.wisc.edu/condor/description.html](http://www.cs.wisc.edu/condor/description.html), 2010.
14. J. Dean, Handling Large Datasets at Google: Current Systems and Future Directions, Invited Talk at HSF Panel. <http://labs.google.com/people/jeff>, 2008.
15. Deelman E, Singh G, Livny M, Berriman B, Good J. The cost of doing science on the cloud: The Montage example. In: *Proc of ACM/IEEE Conf on Supercomputing*. Austin, TX: IEEE Press; 2008;1–12.
16. Demirbas M, Bayir MA, Akcora CG, Yilmaz YS, Ferhatosmanoglu H. Crowd-sourced sensing and collaboration using Twitter. In: *IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (WoWMoM)*. Montreal. 14–17 June 2010;1–9.
17. Distributed Robotics Garden, MIT. <http://people.csail.mit.edu/nikolaus/drg/>, 2010.
18. V. Dixit, Cloud Mashup: Agility and Scalability, EE 657 Final Project Report, Univ. of S. Calif., May 2010.
19. Ekanayake J, Qiu X, Gunarathne T, Beason S, Fox G. High performance parallel computing with clouds and cloud technologies. In: *Cloud Computing and Software Services: Theory and Techniques*. CRC Press (Taylor and Francis) 2010;30.
20. Ekanayake J, Balkir AS, Gunarathne T, et al. DryadLINQ for scientific analyses. In: *Fifth IEEE International Conference on eScience*. Oxford, England. 2009.
21. Ekanayake J, Li H, Zhang B, et al. Twister: A runtime for iterative MapReduce. In: *Proceedings of the First Int'l Workshop on MapReduce and Its Applications*, ACM HPDC. Chicago. 20–25 June 2010.
22. J. Ekanayake, T. Gunarathne, J. Qiu, Cloud technologies for bioinformatics

applications, in: IEEE Transactions on Parallel and Distributed Systems, accepted to appear,  
[http://grids.ucs.indiana.edu/ptliupages/publications/BioCloud\\_TPDS\\_Journal\\_Jan\\_2011](http://grids.ucs.indiana.edu/ptliupages/publications/BioCloud_TPDS_Journal_Jan_2011).

23. S. Farnham, The Facebook Application Ecosystem, An O'reilly Radar Report, 2008.
24. Feng Q, Hwang K, Dai Y. Rainbow Product Ranking for Upgrading e-Commerce. *IEEE Internet Comput.* 2009;13(5):72–80.
25. D.G. Feitelson, Workload Modeling for Computer Systems Performance Evaluation, Draft Version 0.7, Hebrew University of Jerusalem, 2006.
26. Fong P, Anwar M, Zhao Z. A privacy preservation model for Facebook-style social network systems. In: *European Symposium on Research in Computer Security (ESORICS 2009)*. 21–23 September 2009.
27. Fovet F. Impact of the use of Facebook amongst students of high school age with social, emotional and behavioural difficulties (SEBD). In: *IEEE 39th Frontiers in Education Conference*. 2009.
28. Fox G, Bae S, Ekanayake J, Qiu X, Yuan H. Parallel data mining from multicore to cloudy grids. In: *High Speed and Large Scale Scientific Computing*. Amsterdam: IOS Press; 2009.
29. Freeman L. In: *The Development of Social Network Analysis*. Vancouver: Empirical Press; 2006.
30. FutureGrid Cyberinfrastructure to allow testing of innovative systems and applications, Home page. [www.futuregrid.org](http://www.futuregrid.org), (accessed 13.11.10).
31. S.L. Garfinkel, An evaluation of Amazon's grid computing services: EC2, S3 and SQS, in: Center for Research on Computation and Society, Harvard University, Technical Report, 2007.
32. Garfinkel S. Commodity grid computing with Amazon's S3 and EC2. *Login*. 2007;32(1):7–13.
33. Gong L, Sun SH, Watson EF. Performance modeling and prediction of non-dedicated network computing. *IEEE Trans Computers*. 2002;51(9):1041–1055.
34. A. Greenberg, J. Hamilton, D.A. Maltz, P. Patel, The cost of a cloud: Research problems in data center networks, in: SIGCOMM Computer Communication Review, Vol. 39, No. 1, pp. 68–73, 2008,  
<http://doi.acm.org/10.1145/1496091.1496103>.
35. Grid'5000 and ALADDIN-G5K: An infrastructure distributed in 9 sites around France, for research in large-scale parallel and distributed systems.  
<https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home>, (accessed 20.11.10).
36. Grossman R, Gu Y, Sabala M, et al. The open cloud testbed: Supporting open source cloud computing systems based on large scale high performance. In: Doulamis A, ed. *Dynamic Network Services*. Berlin Heidelberg: Springer; 2010:89–97.
37. Grossman R, Gu Y, Mambretti J, et al. An overview of the open science data cloud. In: *Proc of the 19th ACM Int'l Symp on High Performance Distributed Computing*.

Chicago. 2010;377–384.

38. Gunarathne T, Wu TL, Qiu J, Fox G. Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications. In: *Proceedings of the Emerging Computational Methods for the Life Sciences Workshop of ACM HPDC 2010 Conference*. Chicago. 20–25 June 2010.
39. Hoffa C, et al. On the use of cloud computing for scientific workflows. In: *IEEE Fourth International Conference on eScience*. December 2008.
40. Hopcroft J. Computer science theory to support research in the information age. In: *Distinguished Lecture*. University of Southern California 6 April 2010.
41. Hwang K, Xu Z. In: *Scalable Parallel Computing: Technology, Architecture and Programmability*. New York: McGraw-Hill Book Co.; 1998.
42. Hwang K, Li D. Trusted cloud computing with secure resources and data coloring. *IEEE Internet Comput.* 2010;September:14–22.
43. Keahey K, Tsugawa M, Matsunaga A, Fortes J. Sky computing. *IEEE Internet Comput.* 2009;13:43–51  
doi:<http://doi.ieee.org/10.1109/MIC.2009.94>;  
[www.nimbusproject.org/files/Sky\\_Computing.pdf](http://www.nimbusproject.org/files/Sky_Computing.pdf).
44. Kleinberg J. *Algorithmic Perspectives on Large-Scale Social Network Data* Cornell University 2008.
45. Kortuem G, Kawsar F, Fitton D, Sundramoorthy V. Smart objects as building blocks for the Internet of things. *IEEE Internet Comput.* 2010;14(1):44–51.
46. Kumar R, Novak J, Tomkins A. Structure and evolution of online social networks. In: *The 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. August 2006.
47. Lee EA. Cyber physical systems: Design challenges. In: *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing*. 5–7 May 2008;363–369.
48. Langville A, Meyer C. *Google's PageRank and Beyond: The Science of Search Engine Rankings* Princeton, NJ: Princeton University Press; 2006.
49. Leskovec J, Langt K, Dagupta A, Mahoney M. Statistical properties of community structure in large social and information networks. In: *International World Wide Web Conference, (WWW)*. 2008.
50. Li H. Performance evaluation in grid computing: A modeling and prediction perspective. In: *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*. May 2007;869–874.
51. Z.Y. Li, G. Xie, K. Hwang, Z.C. Li, Proximity-Aware overlay network for fast and churn resilient data dissemination, in: *IEEE Transactions on Parallel and Distributed Systems*, Accepted to appear 2011.
52. Linthicum D. *Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide* Addison Wesley Professional 2009.
53. Lloyd BA. Professional networking on the Internet. In: *Pulp and Paper Industry Technical Conference*. Birmingham, AL. 2009;62–66.
54. Lou X, Hwang K. Collusive Piracy Prevention in P2P Content Delivery Networks.

*IEEE Trans Computers.* 2009;58(July):970–983.

55. Lou X, Hwang K, Hu Y. Accountable file indexing against poisoning DDoS attacks in P2P networks. In: *IEEE Globecom*. Honolulu. 3 November 2009.
56. Magellan: A cloud for science at Argonne. <http://magellan.alcf.anl.gov/>, (accessed 15.11.10).
57. Mei L, Chan W, Tse T. A tale of clouds: Paradigm comparisons and some thoughts on research issues. In: *IEEE Asia-Pacific Services Computing Conference*. December 2008.
58. Mislove A, Marcon M, Gummadi KP, Druschel P, Bhattacharjee B. Measurement and analysis of online social networks. In: *The 7th ACM SIGCOMM Conference on Internet Measurement*. October 2007.
59. Napper J, Bientinesi P. Can cloud computing reach the top500? In: *Proceedings of the Combined Workshops on Unconventional High Performance Computing Workshop Plus Memory Access Workshop*, ACM. Ischia, Italy. 2009;17–20.
60. Nimbus Cloud Computing. <http://workspace.globus.org/>, 2010.
61. Norman W, Paton M, de Aragao T, et al. Optimizing utility in cloud computing through autonomic workload execution. In: *IEEE Computer Society Technical Committee on Data Engineering*. 2009.
62. D. Nurmi, R. Wolski, C. Grzegorczyk, et al., Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems, UCSB, Santa Barbara, Technical Report, 2008.
63. Olston C, Reed B, Srivastava BU, et al. Pig Latin: A not-so-foreign language for data processing. In: *Proceedings of the 2008 ACM SIGMOD Int'l Conf on Management of Data*. Vancouver. 9–12 June 2008.
64. Open Cirrus, Welcome to Open Cirrus, the HP/Intel/Yahoo! Open Cloud Computing Research Testbed. <https://opencirrus.org/>, (accessed 20.11.10).
65. M. Palankar, A. Onibokun, A. Iamnitchi, M. Ripeanu, Amazon S3 for science grids: A viable solution? Computer Science and Engineering, University of South Florida, Technical Report, 2007.
66. A. Passant, T. Hastrup, U. Bojars, J. Breslin, Microblogging: A semantic web and distributed approach, in: 4th Workshop on Scripting for the Semantic Web in conjunction with ESWC 2008.
67. Qiu J, Gunarathne T, Ekanayake J, et al. Hybrid cloud and cluster computing paradigms for life science applications. In: *11th Annual Bioinformatics Open Source Conference (BOSC 2010)*. Boston. 9–10 July 2010.
68. J. Qiu, T. Ekanayake, T. Gunarathne, et al., Data Intensive Computing for Bioinformatics.  
[http://grids.ucs.indiana.edu/ptliupages/publications/DataIntensiveComputing\\_Book.pdf](http://grids.ucs.indiana.edu/ptliupages/publications/DataIntensiveComputing_Book.pdf) 29 December 2009.
69. Reed D. Clouds, clusters and ManyCore: The revolution ahead. In: *IEEE International Conference on Cluster Computing*. 29 September–1 October 2008.
70. Rittinghouse J, Ransome J. *Cloud Computing: Implementation. Management and Security* CRC Publisher 2010.

71. Rochwerger B, Breitgand D, Levy E, et al. The Reservoir Model and Architecture for Open Federated Cloud Computing. *IBM Syst J.* 2008.
72. V. Sanhu, The CERN Virtual Machine and Cloud Computing, B.S. Thesis at the Dept. of Physics, University of Victoria, Canada, 29 January 2010.
73. G. Santucci, The Internet of Things: Between the Revolution of the Internet and the Metamorphosis of Objects.  
[http://ec.europa.eu/information\\_society/policy/rfid/documents/iotrevolution.pdf](http://ec.europa.eu/information_society/policy/rfid/documents/iotrevolution.pdf), 2010.
74. Satyanarayanan M, Bahl V, Caceres R, Davies N. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Comput.* 2009;8(4):14–23.
75. Schopf J, Berman F. Performance prediction in production environments. In: *12th International Parallel Processing Symposium*. Orlando, FL. April 1998;647–653.
76. Science Clouds: Informal group of small clouds made available by various institutions on a voluntary basis. <http://scienceclouds.org/>, (accessed November 2010).
77. H. Song, Exploring Facebook and Twitter Technologies for P2P Social Networking, in: EE 657 Final Project Report, University of Southern California, May 2010.
78. H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, Vision and Challenges for Realising the Internet of Things, European Union, March 2010.
79. Venus-C, Virtual Multidisciplinary Environmemnts Using Cloud Infrastructure. [www.venus-c.eu/Pages/Home.aspx](http://www.venus-c.eu/Pages/Home.aspx), (accessed November 2010).
80. Walker E. Benchmarking Amazon EC2 for high-performance scientific computing. *Login.* 2008;33(5):18–23.
81. Welbourne E, Battle L, Cole G, et al. Building the Internet of things using RFID: the RFID ecosystem experience. *IEEE Internet Comput.* 2009;13(3):48–55.
82. Wasserman S, Faust K. *Social Networks Analysis: Methods and Applications* Cambridge: Cambridge University Press; 1994.
83. Watts D. *Small Worlds: The Dynamics of Networks between Order and Randomness* Princeton: Princeton University Press; 2003.
84. Weng M. *A Multimedia Social Networking Community for Mobile Devices*. Tisch School of The Arts New York University 2007.
85. Wikipedia, Cyber-physical systems. [http://en.wikipedia.org/wiki/Cyber-physical\\_system](http://en.wikipedia.org/wiki/Cyber-physical_system), 2010.
86. Wikipedia, Social Network. [http://en.wikipedia.org/wiki/Social\\_network](http://en.wikipedia.org/wiki/Social_network), 2010.
87. Wikipedia, Facebook. <http://en.wikipedia.org/wiki/Facebook>, March 3, 2011.
88. Wikipedia, Mashup (web app hybrid).  
[http://en.wikipedia.org/wiki/Mashup\\_%28web\\_application\\_hybrid%29](http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29), 29 November 2010.
89. Wu Y, Hwang K, Yuan Y, Wu C. Adaptive workload prediction of grid performance in confidence windows. *IEEE Trans Parallel Distrib Syst.* 2010;21(July):925–938.
90. Xu Z, Hwang K. Early prediction of MPP performance: SP2, T3D, and paragon experiences. *J Parallel Comput.* 1996;22(7):917–942.

91. Yan L, Zhang Y, Yang LT, Ning H. *The Internet of Things: From RFID to the Next-Generation Pervasive Networked Systems* Auerbach Publications 2008.
92. Yan B, Huang G. Supply chain information transmission based on RFID and Internet of things. In: 2009;166–169. *International Colloquium on Computing, Communication, Control, and Management (CCCM 2009)*. Vol. 4.
93. Yigitbasi M, Iosup A, Epema D. C-Meter: a framework for performance analysis of computing clouds. In: *International Workshop on Cloud Computing*. May 2009.
94. B.J. Zhang, Y. Ruan, T.L. Wu, J. Qiu, A. Hughes, G. Fox, Applying twister to scientific applications, in: International Conference on Cloud Computing (CloudCom 2010),  
<http://grids.ucs.indiana.edu/ptliupages/publications/PID1510523.pdf>, 2010.

## Homework Problems

### Problem 9.1

Describe the shortcomings of mobile devices and distance clouds for mobile cloud computing. Explain how the cloudlets developed at CMU can alleviate these shortcomings to enable mobile cloud computing requests submitted from handheld pervasive devices.

### Problem 9.2

Answer the following two questions on GPSes:

- a. Compare the four satellite positioning systems developed in the United States, Russia, the European Union, and China, in terms of their satellite technology, deployment architecture, coverage areas, precision, capability, and application areas.
- b. Explain why a GPS receiver needs to use the signals from four satellites to locate a single point (a 3D location) on the Earth's surface.

### Problem 9.3

What are the differences between active RFID and passive RFID? Comment on their advantages and limitations. Do the same for active versus passive GPS receivers.

### Problem 9.4

Compare Facebook and Twitter in four reported applications and discuss their relative strengths and weaknesses in these applications.

### Problem 9.5

Workload traces on AuverGrid can be described with three workload patterns including *sudden level change*, *sudden fluctuation*, and *gradual level change*. What are the three load

patterns and how would you characterize their potential impact on the performance prediction results?

## Problem 9.6

The IoT differs from the traditional Internet in many ways. Identify their differences and describe their distinctions in connecting entities, infrastructure and networking, and application domains.

## Problem 9.7

Distinguish the following pairs of terminologies:

- a. Open source versus proprietary operating systems
- b. Internet of Things versus cyber-physical systems
- c. Social networks versus professional networks

## Problem 9.8

Answer the following two questions on wireless sensor networks:

- a. Characterize three generations of wireless sensor networks in terms of their size, weight, power supply, deployment, and so on.
- b. Low power operation is desired in using a wireless sensor network (WSN). Plot the desired power distribution in using a WSN. You should explain how to minimize power consumption by operating the WSN in three modes periodically.

## Problem 9.9

Explain how to apply the RFID technology in merchandise positioning during transportation and counting of the remaining goods in a supermarket.

## Problem 9.10

Describe the following technology terms, functional modules, and cloud services associated the IBM Blue Cloud and RC2 cloud. You may need to get more information from IBM's web sites or the *IBM Journal of Research and Development* for reported achievements.

- a. IBM WebSphere
- b. Tivoli Service Automation Manager (TSAM)
- c. IBM Research Compute Cloud (RC2)
- d. Tivoli provisioning manager
- e. Linux with Xen

## Problem 9.11

Conduct research on the possible use of cloud resources for processing data collected in selected IoT applications (see Figure 9.19). Propose a solution scheme and the possible cloud resources needed. You can narrow down the scope of sensor data collected to control the complexity involved.

## Problem 9.12

Study the performance metrics for HPC applications (Section 9.2.2) and HTC applications (Section 9.2.3). Research how to improve their accuracy and effectiveness through analysis of real-life HPC and HTC benchmark experiments on data-center clusters or on cloud platforms that you have accessed. Suggest new or improved performance metrics that can cover more machine characteristics and application parameters identified in the benchmark experiments.

## Problem 9.13

Briefly explain how Facebook and Twitter cope with the security, privacy and copyright violation issues in online social networking applications. Common protection techniques and unique solutions to each system should be treated, separately.

## Problem 9.14

Match the following 20 names or abbreviated terms in the left column with the best-match definitions or descriptions in the right column.

Term	Description to Be Mapped with the Terms on the Left Column
MySpace	a) The world's largest social network in terms of registered users today
Cloudlet	b) A social network mainly used for video strip forwarding applications
Bluetooth	c) A natural extension of the Internet involving humans in the loop of control
IoT	d) A social network for microblogging and news alert applications
WiMAX	e) A radio tagging technology used in Internet of Things construction
RFID	f) An infrastructure that supports remote cloud access by mobile devices
ZigBee	g) A private cloud built by IBM for internal use in R&D applications
GPS	h) An Internet extension for interconnecting all objects, people, animals, and products
Facebook	i) An Internet access technology used mainly for its speed and flexibility
YouTube	j) The most scalable wireless sensor network
Twitter	k) A fixed wireless technology for broadband Internet access
Nebula	l) A satellite technology for location-sensitive applications
CPS	m) A wireless technology used to replace cables in short-range applications
RC2	n) One of the most frequently used SaaS applications in the business world
LinkedIn	o) A social network mainly used in professional networking over the Internet
WiFi	p) A social network web site specialized in customized social services
CRM	q) A large private cloud used initially for aerospace and weather forecasting applications

## Problem 9.15

Circle only one answer in each of the following multiple-choice questions:

1. Which one of the following *social network properties* is correctly stated?
  - a. The structure holes can be eliminated by filling one or more links to other nodes.
  - b. The local bridge consists of only one link, like a bridge.
  - c. The centrality has nothing to do with the node degree.
2. Which design choice was NOT adopted in the Kimberley cloudlet prototype at CMU?
  - a. The use of an imported virtual machine (VM) from a mobile device.
  - b. A base VM is deployed in the cloudlet host machine.
  - c. An imported VM, once deployed in the cloudlet, cannot be removed.
3. Which of the following descriptions of the Internet of Things is correctly stated?
  - a. The IoT does not apply GPS technology in supply chain management.
  - b. A smart grid does not explore solar energy or wind power.
  - c. ZigBee networks are more scalable than WiFi and Bluetooth networks.
4. Which of the following assessments on today's Facebook applications is correctly stated?
  - a. The most popular use of Facebook today is to enhance communication.
  - b. Gift sending and dating services are not supported by Facebook.
  - c. Personal profiles of Twitter users are better supported than those for Facebook users.

## Problem 9.16

Conduct a Google search for technical information on the following two large-scale private clouds. Describe what you have found out from reliable sources of information. Try to reveal the cloud platform architecture and its infrastructure bases and scope of major applications.

- a. The private cloud under development by NASA in the United States
- b. The private cloud under development by CERN in Europe

## Problem 9.17

Briefly describe the following technology terms, functional modules, and cloud services associated with the SGI Cyclone cloud engine:

- a. Scale-up, scale-out, and hybrid management of cloud servers
- b. SGI Altix servers and ICE clusters
- c. Reported Cyclone HPC applications

## Problem 9.18

This problem is related to [Example 9.13](#). Prove that the identification of the attack accounts (subgraph  $H$ ) and the tracing of the targeted users are solvable mathematically.

# Index

---

Page numbers in *italics* indicate figures and tables

## 0-9, and Symbols

@Swineburne, 30

$2^{128}$  IP addresses, 13

3D printing, 11

3D torus interconnect, 117–118, 118

*80h* interrupt, 147

## A

ABI, *See Application Binary Interface (ABI)*

Access control, 465

Access routers (ARs), 195

Accounting module, 440

ACOM, 513

Active state, 160

ActiveMQ, 291, 292

Activeness, 99

Active-takeover clusters, 99

Ad hoc connectivity, 481

Adaptation, in programming model, 8

Adaptive hybrid model (AH-Model), for workload prediction, 460

Adaptive workload prediction, 458–459, 458

Advanced Message Queuing Protocol (AMQP), 291

Advanced Queuing, 308

Agility, 558  
AIX/OS, 72  
Akamai CDN, 529  
ALICE, 554–555  
AliEn, 554  
Alliance to Save Energy (ASE), 51  
Altix ICE, 550–551  
Amazon, 228, 246, 336, 548  
Amazon Dynamo, *See* Dynamo  
Amazon EBS, *See* Elastic Block Store (EBS)  
Amazon EC2, *See* Elastic Compute Cloud (EC2)  
Amazon Elastic MapReduce, 203  
Amazon Machine Images (AMIs), 380, 380  
Amazon S3 (Simple Storage Service), 242, 275–276, 382–383, 383  
Amazon Simple Queue, 292  
Amazon VPC, 201–203, 202  
Amazon Web Services (AWS), 193, 231–233, 548  
    infrastructure, 232  
    mashup on, 556–557  
    programming on, 379–387  
    service offerings, 228, 229, 233  
    virtualization support in, 224  
    virtualized resources in, 223  
Amber, 321  
Amdahl’s law, 47  
AMGA (ARDA Metadata for Grid Applications), 309  
Amoeba, 41, 41  
AMQP, *See* Advanced Message Queuing Protocol (AMQP)  
Anabas, 293  
Analytical ultracentrifuge (AUC), 295

Andrew File System (AFS), 90

Aneka, 203, 393–398

advantages of, 394

application of Maya rendering case study, 397–398

architecture and components, 394, 396–398

capabilities, 395

virtual appliances, 398

Anomaly detection, 253–255

Anonymity, 481, 488

Apache, 529

Apache Maven, 303

APIs (application programming interfaces), 419

grid, 444

UDDI, 306

Appliances, 384

virtual, 23, 398

Application Binary Interface (ABI), 138

Application isolation, 133

Application layer, 52, 52–53, 88, 419

in cloud architecture, 218, 218

in multitier architecture, 282, 283

Application level

checkpointing at, 101–102

virtualization, 132

Application sandboxing, 133

Application services, 396

Application streaming, 133

Application traffic support, 208–209

Applications

asynchronous, 369

classification of, 369

data-intensive, 546

flexibility in deployment of, 8

HPC, 9

HTC, 9

of IoT, 590–597

loosely synchronous, 369, 371

scalability, 45

social networking, 603–608

third-party, 244

ARCH interconnect, 114, 115

ASE, *See* Alliance to Save Energy (ASE)

Astrolabe, 313

Asynchronous applications, 369

Atom Publishing Protocol (APP), 306

Atonomic computing, 26

Attribute authorities, 465

Authentication, 89, 464–465

in GSI, 468

improper, 50

mutual, 468, 469, 469

Authorization

for access control, 465

in GSI, 466

methods, 464–465

models, 465, 465

Authorization agent model, 465, 465

Automation, data center, 169–179

Autonomous system (AS), 487

Auto-regression (AR) method, 459, 461

Auto-scaling, 380

AuverGrid, 458, 461

Availability, 49, 70, 95, 98, 176, 570

  clouds and, 339

  of computer system types, 96

  cost analysis and, 101

  failure rate and, 95–96

  levels of, 99

  node, 108

Availability clusters, 77

AWS, *See* Amazon Web Services (AWS)

Azure, *See* Microsoft Azure

Azure Table, 336, 341, 386–387

## B

Backup components, 98

Backward recovery, 100

Bare metal, 575–576

Bare-metal VM, 23

Batch jobs, 107

BCube network, 212, 212, 214

Behavior-sensitive instructions, 147

BEinGRID, 429

Beowulf clusters, 71

*See also* Compute clusters

Berkeley NOW, 66

Betweenness, 601

BigTable, 42, 313, 341, 376–379, 377

Binary translation, with full virtualization, 141–143, 142

BindingTemplate, 305

Bioinformatics, 321–323

Biometrics, 577

BirdBath, 289–288

Bitcoin, 497

Bit-level parallelism (BL), 9

BitTorrent, 497, 525–528, 526–527, 531, 531

BLAST program, 321

Blobs, 340, 385

Blobstore, 372

Block blobs, 385

Blockreports, 356

BLP, *See* Bit-level parallelism (BL)

Blue Cloud, 193, 228, 238, 549–550, 549

Blue Gene/L, 78–79

BOINC (Berkeley Open Infrastructure for Network Computing), 87, 417, 420, 429, 444

Bootstrap agent, 534

Border routers (BRs), 195

Bottlenecks, 226, 242

    sequential, 47

BPEL, *See* Business Process Execution Language (BPEL)

BPMN, *See* Business Process Modeling Notation (BPMN)

BPSS, *See* Business Process Specification Schema (BPSS)

Bridge, 602

Broadband over power line technology, 11

Brokerage systems, 379

Buffer overflows, 251

Bulletin board services (BBS), 598

Bush, Vannevar, 547

Business grids, 417

Business logic layer, 283

Business Process Execution Language (BPEL), 317, 320

Business Process Modeling Notation (BPMN), 317

Business Process Specification Schema (BPSS), 317

Business Transaction Protocol (BTP), 317

BusinessEntity, 305

BusinessService, 305

Buyya, Rajkumar, 6

## C

Caching, 372, 425

Calendar scheduling, 104

CAM-Chord, 513

CAN (Content Addressable Network), 498, 500–501, 500, 504, 505

cceHUB.org, 297

CDC 6400, 4

CDNs, *See Content delivery networks (CDNs)*

Cell/B.E. processors, 119

Cellular Disco, 165, 166

Centralized clusters, 68

Centralized computing, 6–7, 193

Centralized networks, 602

Centralized reputation systems, 258

Centralized systems, 7

CERN, *See European Council for Nuclear Research (CERN)*

Certificate authorities (CAs), 424, 464, 468–469

CGSP, *See ChinaGrid Support Platform (CGSP)*

Chatter, 552

Checkpoint file, 101

Checkpoint interval, 102, 102

Checkpointing, 101–104

choosing optimal interval, 102

coordinated versus independent, 104

forked, 103

incremental, 102

kernel, library, and application levels, 101–102

overheads, 102

parallel programs, 103–104, 103

user-directed, 103

Checkpoints, 97

Chemical Informatics Grid, 288–289

Chemistry Development Kit (CDK), 288–289

China Education and Research Network (CERNET), 434

ChinaGrid, 31, 428, 434

ChinaGrid SuperVision (CGSV), 438, 438

ChinaGrid Support Platform (CGSP), 434–435, 444, 452–455

components, 453

functional building blocks, 454

job execution flow, 453–455

ChinaGrid Supporting Platform (CGSP), 440

Chip multiprocessors (CMPs), 9, 170

Chord, 497, 498–499, 499–500, 503, 504, 505–507, 524

Chroot command, 137

Chubby, 378, 379

Churn resilience, 489–490, 512–514

Churn-Resilient Protocol (CRP), 512, 513

CICC services, 288–289

Client machines, 5

Client nodes, 109

Client/server architecture, 32, 482, 482

Clock rate, 21

Closeness, 602

## Cloud architecture, 215–227

Amazon Web Services (AWS), 231–233

challenges of, 225–227

design, 215–217

disaster recovery and, 223, 224–225

generic, 216–217, 217

Google App Engine (GAE), 229–231

hardware virtualization and, 221–223

layered development of, 218–221, 218

market-oriented, 219–220, 220

Microsoft Azure, 233–234

QoS and, 220–221

resource provisioning and platform development, 237–243

virtualization support and, 221–225, 222

## Cloud computing, 5, 7, 26, 35, 417

*See also* Clouds

advantages of, 28, 36

benefits of, 193

challenges, 135

concept of, 192

cost model, 197–198, 198

data center virtualization for, 25–26

data features and databases, 340–341

definition of, 34

deployment modes, 36

design objectives, 197–198

ecosystems, 199–200

emergence of, 135

emerging software environments, 387–398

extended services, 235–237, 235

grids and, 456

growth of, 6

IaaS model, 200–203

inter-cloud resource management, 234–249

marketplaces, 215

middleware applications, 291–294

mobile, 558–561

over Internet, 34–36

PaaS model, 203–206, 203

paradigm, 196

performance of, 561–576

platforms, 27

programming and runtime support, 341–343

quality of service in, 572–574

SaaS model, 205–206

service models, 35, 36, 192–206, 228

software environments for, 36–44

software stack for, 236–237

system models for, 27–36

technology, 11

trends in, 546–547

ubiquitous computing and, 546–561

vs. utility computing, 10–11

Cloud Exchange (CEx), 248

Cloud management tools, 200

Cloud managers, 199

Cloud mashups, 206, 555–558

advantages of, 558

concept of, 556–557, 556

content for, 306

definition of, 306

experimental results, 557–558, 557

lesson learned about, 558

RESTful Web services and, 276

Cloud of clouds, 40

Cloud of grids, 40

Cloud OS, for virtualized data centers, 172–176

Cloud programming models, 343

Cloud resources

global exchange of, 246–249

inter-cloud management of, 234–249, 248

provisioning of, 237–243

Cloud security, 249–260, 339, 561

challenges, 251

copyright protection, 252

data and software protection techniques, 255–257

data coloring, 256–257, 256

data lock-in problem, 257

defense strategies, 249–253, 250–251

distributed intrusion/anomaly detection, 253

firewalls, 252, 253

gateways, 252, 253

privacy protection, 252–253, 255–256

virtualization and, 252

watermarking, 256–257, 256

Cloud service providers, 546–552, 548

Cloud services, 235–237

provisioning of, 237–238

Cloud watermarking, 256, 256–257

Cloudera, 343

CloudFront, 226, 232

Cloudlets, 558–561

vs. clouds, 559, 560

concept of, 559, 559

fast VM synthesis in, 560

Clouds, 7, 27, 34–36, 35, 416

benchmarking, 574–576

capabilities, 336, 337

vs. cloudlets, 559, 560

data transport and, 337–339

data-intensive, 369

development trends, 196

enabling technologies for, 215–216, 216

features of, 336–343, 338–339

versus grids, 40

virtual grids, 419

hybrid, 194, 194, 199, 218

performance, 568–572

private, 193–194, 194, 196, 199–200, 552–555

public, 193, 194, 199

scalability, 226

science and research, 562–566

CloudWatch, 232, 380

ClueWeb data sets, 355, 355

Cluster compute instances, 382, 382

Cluster jobs

management systems, 107–109

scheduling methods, 104–106, 105

types, 107

Cluster operating system, 29, 76

Clustering coefficient, 602

Cluster-on-Demand (COD), 165, 166–167, 166–167

Clusters, 5, 27, 27–29, 546

active-takeover, 99

architecture, 28, 28, 75–87

availability, 77

compute, 71

dedicated vs. enterprise, 69

definition of, 66

design issues, 29, 30, 69–71

design objectives, 68–69

design principles, 87–104

development trends, 66–68

failover, 99

family classification, 71

fault-tolerance, 99–101

GPU, 83–87

grids of, 369

HA, 71

hardware, software, and middleware support, 29, 83

hot standby, 99

integrated, 159

job and resource management, 104–112

Linux, 41–42

load sharing facility (LSF) for, 109–110

load-balancing, 71

management of, 338

for massive parallelism, 66–75

milestone systems, 67–68

monitoring, 237

networks of, 7

organization and resource sharing, 76–77

physical, 155

physical vs. virtual, 156–159, 156

resource sharing, 78

server, 194

shared-memory, 77

single-system image, 29, 87–95

system interconnects, 80–82, 80

virtual, *See* Virtual clusters

workload behavior, 107

CMPs, *See* Chip multiprocessors (CMPs)

CNK/SLES9, 73

CNL/OS, 73

Coarse-grain multithreaded processor, 16

Code interpretation, 132

Cohesion, 602

Collaboration, 483

Collaborative platform, 33

Collective layer, 419

Colluders, 533, 535–536

Collusive piracy, 490, 533, 535–538

*Combiner* function, 349

Commodity technologies, 8

Communication, 13

among workers, 344

internode, 70

intracluster, 69

P2P, 480–492

Communication as a Service (CaaS), 235

Communities, on social networks, 604–605

Community engine, 611  
Compact clusters, 68  
Component grids, 314  
Computation, 13  
Computation partitioning, 344, 349  
Computation time, 437  
Computational Chemistry Grid (CCG), 295  
Computational fluid dynamics (CFD), 550  
Computational grids, 7, 27, 30–31, 71, 417  
Computation-intensive applications, 425  
Compute clusters, 71  
Compute nodes, 77, 88  
Compute resources, provisioning of, 237–238  
Computer, definition of, 6  
Computer clusters, *See* Clusters  
Computer room air conditioning (CRAC), 25, 208  
Computer Science and Telecommunication Board (CSTB), 26  
Computer technology, platform evolution, 4  
Computing paradigms  
    distinctions among, 6–7  
    new, 6  
Computing systems, classification of, 27, 27  
Concurrency, 8  
Concurrent computing, 7  
Concurrent programming, 7  
Condor, 312, 320, 444  
Condor-G, 288–289, 294, 442, 444, 444  
Confidence window, 458  
Confidentiality, 49–50  
Connectivity layer, 417–418

Consistency, 528

Consistent checkpointing, 104

Consistent hashing, 497

Consistent snapshots, 104

Consumer relationship management (CRM), 35

Consumer-generated media, 11

Containers, 135, 450–452

Content Addressable Network (CAN), 498, 500–501, 500, 504, 505

Content Addressable Storage (CAS), 171

Content delivery applications, 5, 529–532

Content delivery networks (CDNs), 529–532

- delivery methods, 532
- file chunking and hashing schemes in, 531, 531
- generations of, 530
- global, 529–530, 530

Content distribution, 483

Content poisoning, 533, 535, 535

Content-based systems, 307

Continuous availability (CA), 571

Control, 13, 68

- single point of, 88, 92

Control-sensitive instructions, 147

Convergence, of technologies, 26

Convergence overhead, 520–521, 521

CoopNet, 497

Coordinated checkpointing, 104

Coordinated TeraGrid Software and Services (CTSS), 431

Copyright protection, 50, 252–253, 531, 533–534, 534

Copyright violations, 33

Coral Content Distribution Network, 497

CORBA, 37–38, 321

CORBA Trading Service, 38

Core switches, 210

Cost effectiveness, 571–572

CouchDB, 341

COW (Copy on Write) format, 158

CPS, *See* Cyber-physical systems (CPS)

CPU scavenging, 419–422

CPU speed, 8, 14, 45

CPU virtualization, 147–148

CPUs

advances in, 14, 14–15

architecture, 16

dynamic allocation of, 170

multicore CPUs, 14–16

power efficiency, 19–20

schematic, 15

Cray interconnect, 81

Cray XT5 Jaguar, 78, 116–119

3D torus interconnect, 117–118, 118

features of, 116

hardware packaging, 118–119

interconnect router design, 117

Cron service, 373

CRP, *See* Churn-Resilient Protocol (CRP)

CRUD (create, read, update, delete) operations, 274, 304

Cryptography, 256

CSCW (computer-supported cooperative work), 492

CUDA (Compute Unified Device Architecture)

parallel programming, 86

programming interfaces, 86–87

usage trends, 87

CUDA C, 86

CUDA cores, 18, 19

CUDA model, 17

Cumulus, 389

Customer relationship management (CRM), 236

CuteFTP, 529

Cyber-physical systems (CPS), 6, 13, 590, 595–597

Cyclic graphs, 320

Cyclone, 550, 551

## D

$\delta$  flow, 352

D4Science, 429

DAG, *See* Directed acrylic graph (DAG)

Dally, Bill, 19

Data

management of, 338

structured and semistructured, 376

in UDDI registry, 305

Data access models, 426, 427

Data as a Service (DaaS), 235

Data centers, 53

automation, 26, 456

cloud OS for virtualized, 172–176

construction requirements, 206–207

cooling systems, 207–208, 208

design of, 206–208

green, 224

growth and cost breakdown, 25, 25

interconnection networks, 208–211, 214

low-cost design philosophy, 25–26

management issues, 213–215

modular, 211–212, 211

networking structure, 194–196, 195

reputation-guided protection of, 257–260

server consolidation in, 169–170

server-centric design, 210, 212–213

switch-centric design, 210–211

trust management of virtualized, 176–179

virtual storage management, 171–172

virtualization, for cloud computing, 25–26

virtualization for automation of, 169–179

warehouse-scale, 206–208, 207

Data coloring, 256–257, 256

Data deluge, 26

Data distribution, 349

Data grids, 7, 27, 31, 31

*See also* Computational grids

Data integrity, 49–51, 255–256

Data layer, 282, 283

Data locality, 487

Data lock-in problem, 225, 257

Data management

services, 284

using GT4 library, 451–452

Data Nucleus Access, 370

Data parallelism, 319

Data partitioning, 344, 348

Data privacy, 225–226

Data protection infrastructure, 50–51

Data replication, 425–426  
Data services, SQLAzure, 386–387  
Data sources, 567  
Data transfer time, 436  
Data transfers, parallel vs. striped, 427  
Data transport, 337–339  
Databases, 304–314  
    monitoring, 313  
    NOSQL nonrelational, 341, 370  
    publish-subscribe model and, 307–308  
    relational, 340–341  
DataGrid, 417, 428, 431–434  
Data-intensive applications, 546  
Data-intensive clouds, 369  
Data-intensive grid service models, 425–427  
Data-intensive scalable computing (DISC), 561, 566–567, 568  
Data-level parallelism (DLP), 9  
Dating services, on social networks, 607  
DCE, 41, 41  
DDoS, *See* Distributed denial of service (DDoS)  
Debugging, 226  
DEC PDP 11, 4  
Decentralization, 481  
Decentralized clusters, 68  
Decentralized networks, 602  
Dedicated clusters, 69  
Dedicated mode, 104  
Degradation, 209  
Degree of parallelism (DoP), 9  
Delegation, 468–469

trust, 469–470, 470

Delegation service, 448

DELETE, 274

Demand-driven resource provisioning, 239, 240

Denial of service (DoS), 49, 251, 489

Department of Defense (DoD), 287–288

Dependability, 8, 45

Deployment

dynamic, of virtual clusters, 165–169

fast, 158

of VMs, 158

Developmental Therapeutics Program (DTP), 288

Device emulation, 150, 151

Digital signatures, 468

Digital watermarking, 531

DirectCompute, 86

Directed acrylic graph (DAG), 359, 369

Directories

global, 90

home directory, 90

system, 90

Disaster recovery, 223, 224–225

DISC, *See* Data-intensive scalable computing (DISC)

Discover Middleware, 321

Discovery services, 304

Disk technology, 20, 21

Distributed computing, 6

*See also* Parallel and distributed programming

vs. centralized computing, 193

changes in, 546

computer utilities in, 10

discovery, registries, metadata, and databases, 304–314

energy efficiency in, 51–54

on P2P network, 483

system classification, 27

system models for, 27–36

Distributed data management system, 312–313

Distributed denial of service (DDoS), 225, 254–255, 254, 489, 522–523

Distributed file sharing, 5, 33, 33, 493–496

Distributed file system (DFS), 242, 390

Distributed hash tables (DHTs), 259, 480, 496–497, 496–497, 519–520

Distributed intrusion detection systems (DIDSes), 252

Distributed operating systems, 40–42

Amoeba, 41, 41

comparison of, 41

DCE, 41, 41

MOSIX2, 41–42, 41

Distributed P2P computing, 33, 33

Distributed power management (DPM), 51, 54

Distributed program, 7

Distributed programming, 7

models, 42–44, 43

Distributed RAID, 93, 94

Distributed ranking module, 520

Distributed reputation systems, 258

Distributed storage, 226

Distributed systems, 7

families of, 7–8

operational layers, 52

performance of, 45, 561–576

scalability analysis, 45–48

software environments for, 36–44

system availability, 48, 49

technologies for, 13–26

trends toward, 40–42

workflows, 314–322

Distributed transaction processing, 10

Distributed virtual environment (DVE), 241

DLP, *See* Data-level parallelism (DLP)

Domain name system (DNS), 89, 89, 384

DoS, *See* Denial of service (DoS)

Double-precision (DP) calculations, 84

Downtime, 96, 98

DPFS, 340

DRAM chips, 20

Drives, 340

Dryad, 312–313, 342, 342, 353, 359–365, 360

DryadLINQ, 353, 354, 361–365, 366

Dual-core processor, 15, 16

DVE, *See* Distributed virtual environment (DVE)

DVE manager, 241

Dynamic binary translation, 132

Dynamic deployment, of virtual clusters, 165–169

Dynamic heterogeneity, 153

Dynamic mode, 105

Dynamic power management (DPM), 53

Dynamic priority, 104

Dynamic replication, 425–426

Dynamic resource deployment, 239–241

Dynamic voltage-frequency scaling (DVFS), 53–54, 54

Dynamo, 313

Earth Simulator, 51

EBS, *See Elastic Block Store (EBS)*

EbXML (Electronic Business using eXtensible Markup Language), 38

EC2, *See Elastic Compute Cloud (EC2)*

EC2 Compute Units (ECUs), 382

Ecademy.com, 606

Echelon GPU cluster, 84, 85–86

Eclipse plug-in, 370

EDG, *See European Data Grid (EDG)*

Educational applications, of social networks, 605, 605

Efficiency, 297

Efficiency, 8, 569–570

EGEE (Enabling Grids for E-SciencE), 30, 428–429

EGEE Network Operations Center (ENOC), 429

EigenTrust, 517, 518, 521

Elastic Block Store (EBS), 231, 242, 340, 383–384

Elastic Compute Cloud (EC2), 156, 202, 229, 238, 252, 380–384

benchmarking, 574–576

execution environment, 381

instance types available on, 381

performance analysis, 574

performance results, 240

programming on, 380–382

use of, in AWS platform, 381–382

VM manager, 244

Elastic Computing Platform, 548

Elastic load balancing, 203, 380

Elastic MapReduce, 379, 575

Eli Lily Company, 205

E-mail systems, 242  
E-mail traffic, 480  
Embedded systems, 595  
EMC, 177  
EMule, 531, 531, 538  
Enabling technologies, for Internet of Things, 576–590, 578  
Enclosed clusters, 69  
Encryption, 467  
Endpoint reference (EPR), 287  
Energy consumption  
    reducing, 52  
    of unused servers, 51  
Energy efficiency, 51–54  
Enomaly, 548  
Enterprise bus, 289–290, 290  
Enterprise clusters, 69  
Enterprise multitier architecture, 282, 282–283  
Enterprise resources planning (ERP), 35  
Entity certificates, 466  
Entity groups, 372  
Environmental monitoring, 293  
Erdos-Renyi random graph, 493  
eSpeak, 417  
ESX Server, 144, 145  
Ethernet, 22, 80, 80–81  
Eucalyptus, 173–174, 173–174, 371, 387–389  
European Council for Nuclear Research (CERN), 196, 552–555  
European Data Grid (EDG), 426, 431–434  
    architecture, 432–433  
    data distribution in, 433–434, 433

European Grid Infrastructure, 428

European Union, 428, 431–434

Event scheduling, 104

Event-driven resource provisioning, 238–239, 240

Exascale systems, 17–20, 20, 86

Execution management services, 284

Execution time, 47

Expended page table (EPT), 149–150

Exposed clusters, 69

## F

Fabric layer, 418

Fabric services, 396

Facebook, 480, 597, 599, 608–611, 609–610

Factors, 423

Fallback, 83, 99

Fallback feature, 99

Failover, 83

Failover clusters, 99

Failure analysis, 511

Failure detection, 83, 424

Failure diagnosis, 99–100

Failure notification, 99

Failure rate, 95–96

Failure recovery, 99–100, 509–511

Failures

churn resilience against, 512–513

cost analysis, 100–101

node, 510

partial vs. total, 97

planned vs. unplanned, 96

transient vs. permanent, 96–97

Fast deployment, 158

Fat-tree interconnection topology, 210, 210

Fault tolerance, 37, 48, 51, 70–71, 209, 356, 481, 509–511

Fault-resilient systems, 98

Fault-tolerant clusters, 99–101

Federation access model, 426, 427

Fermi (GF110) chip, 86

Fermi GPU, 17, 18, 19, 87

File hierarchy, single, 90

File sharing, 483, 483, 490, 523–538

distributed, 493–496

File system migration, 162–163

Files

checkpoint, 101

global, 90

local, 90

remote, 90

visibility of, 91

File-sharing applications

BitTorrent, 525–528

copyright protection and, 533

replication and consistency in, 528

search algorithms, 524–528

Filter services (fs), 39

Fine-grain multithreaded processor, 16

Finger tables, 499–500, 506–507

Firewalls, 50, 252, 253, 484

Fixed workload, 47

Fixed-workload speedup, 47

Flat classification, 436

Flexibility, in application deployment, 8

FlexiScale, 202

Flooding algorithms, 524–525

Flooding-based delivery methods, 532

FMA (fused multiply and add) operations, 18

Folding@Home, 420

Force.com, 193, 203, 228, 229, 552

Foreign jobs, 107, 109

Forked checkpointing, 103

Forward recovery, 100

Foster, Ian, 416

Foundation services, 396

Four-issue superscalar processor, 16

Fread, 91

Freenet, 505–506, 506

Free-riding problem, 488

Full data flow, 352

Full virtualization, 141–143

Functional parallelism, 319

FutureGrid, 340, 343, 429, 562–563, 562, 564

Fuzzy theory, 456, 456

Fuzzy trust model, 463

FVM (Feather-Weight Virtual Machines), 137

Fwrite, 91

## G

GAE, *See Google App Engine (GAE)*

Gage, John, 6

Game theory, 456–457, 456

Gang scheduling, 106

Gang-scheduling skew, 106

Gannon, Dennis, 206

GASS module, 447, 447

Gateway nodes, 194

Gateways, 252, 253

GE, 14

GeForce 256, 17–20

GeForce 8800, 17

General-purpose computing on GPUs (GPGPU), 17, 85

Generative topographic mapping (GTM), 574

Geographic layout, 508

GET, 274

*Get* operation, 497

GetRight, 529

GFAC tool, 301–302

GFS, *See* Google File System (GFS)

Gigabit Ethernet, 22, 25, 81

Gigabits per second (Gbps), 14

Gilder's law, 8

Global access of secondary storage, 447

Global directory, 90

Global files, 90

Global Grid Forum (GGF), 422

Global Information Grid (GiG), 287

Global positioning system (GPS), 6, 11, 587–590

operating principles of, 588–589

passive vs. active, 587–588

real-time vehicle tracking using, 590

triangulation location calculation, 589–590

workings of, 587

worldwide deployment status, 590

Global reputation aggregation, 516–517, 520

Global reputation score, 515, 518

Global snapshots, 103–104

GlobalHub.org, 297, 300

Globus Container, 451

Globus Project, 416

Globus Security Infrastructure (GSI), 444, 447, 447, 451, 466–?470

authentication and delegation, 468–469

functional layers, 466–467, 466

message-level security, 467

transport-level security, 467

Globus Toolkit, 44, 44, 280, 294, 422, 433, 435, 439, 443, 444

architecture, 446–449

client-Globus interactions, 448–449, 449

data management using, 451–452

GT4 library, 446–448, 447

job workflow, 448, 448

GLUE, 444

Gmail, 205, 231

Gnutella, 482, 482, 493–496

complexity properties, 504

embedding Chord on top of, 503

file chunking and hashing schemes in, 531, 531

flooding scheme, 495, 501

search performance, 525

GOAT project, 293

GoCast, 513

GoFront, 397–398

GoGrid, 202, 205, 548

Google, 228, 548

Google Accounts, 373

Google App Engine (GAE), 193, 203, 204–205, 229–231, 336, 548

applications, 231

architecture, 229–230, 230

cloud infrastructure, 229

functional modules, 230–231

mashup on, 556–557

platform, 204

programming support, 370–379, 372

service offerings, 228

virtualization support in, 224

virtualized resources in, 223

Google BigTable, *See* BigTable

Google Data API, 372

Google Docs, 229, 231

Google Earth, 231, 242, 376

Google File System (GFS), 312–313, 373–375

architecture, 374

data mutation sequence in, 374, 374

features, 242, 242

MapReduce and, 351

Google MapReduce, *See* MapReduce

Google SDC Secure Data Connection, 372

Google search engine, 231

Google search engine cluster, 67

architecture, 80

crossbar switch, 81

Google Web Toolkit (GWT), 370

GossipTrust, 535

Government applications, of social networks, 606

GPS, *See Global positioning system (GPS); Grids*

GPU, *See Graphics processing units (GPUs)*

GPU clusters, 83–87

    architecture, 86

    capabilities, 83

    components, 85

    CUDA parallel programming, 86

    design, 84–85, 84

GQL, 370

GRAAL Project, 166

GRAM module, 448, 450

GRAM protocol, 445, 447

Graphics processing units (GPUs), 14

    GPU computing, 17–20, 18

    many-core, 15

    power efficiency, 19–20

    programming model, 17–19

    workings of, 17

Graphs

    random, 493

    scale-free, 493

    small-world, 493

Gray, Jim, 8, 21, 26

Great global grid (GGG), 417, 417

Green computing, 158

Green data centers, 224

Green pages, 304

Grid APIs, 444

Grid computing, 26

application trends, 455–470  
economy model for, 440, 441  
goals, 416  
infrastructures, 29–31, 31  
Open Grid Services Architecture (OGSA), 44  
software and middleware for, 443–455

Grid data access models, 426, 427

Grid families, 31, 417, 417

Grid Information Services (GISeS) agencies, 420

Grid Infrastructure Group (GIG), 430

Grid of clouds, 40

Grid of grids, 314–315

Grid resource brokers, 442

Grid resource management, 435–443

- job scheduling, 435–437
- resource management system (RMS), 435
- resource monitoring, 437, 437–438
- service accounting and economy model, 439–440
- using GRAM, 450–451

Grid resources, 418–419

- aggregating, 418, 420
- allocation of, 419, 436, 439
- brokering, 440–443, 441

Grid Service Handles (GSHs), 286, 423–424, 423

Grid service migration, 423–424, 424

Grid service models, 425–427

Grid service protocol stack, 417–418, 418

Grid service providers (GSPs), 31

Grid Service Reference (GSF), 286, 423–424, 423

Grid services, 280, 283–287

accounting system for, 439–440, 439

monitoring and metering services, 439–440

Grid standards, 44, 444

Grid Workloads Archive (GWA), 458

Grid'5000, 244–246, 428, 429–430, 458, 459, 461, 563–564

Gridbus, 440–443

Gridbus Resource Broker, 442

GridChem, 295

GridFTP, 426, 447, 449, 451

Grids, 4

    architecture, 416–419

    business, 417

    vs. clouds, 40, 419

    of clusters, 369

    computational, 7, 30, 31, 71, 417

    data, 7, 31, 31

    features of, 336–339

    history and service categories, 416–419

    HPC, 419

    vs. HPC clusters, 417

    information (knowledge), 417

    InterGrid, 241

    international, 428–430, 429

    layered architecture for, 37–38, 37

    middleware applications, 291–294

    national, 428–430

    P2P, 31, 417

    programming environment, 338

    security, 455–470

    selfish, 457, 457

semantic, 309–312, 311, 417  
service modelings, 416–427  
vs. supercomputers, 419–422  
toolkits, 44  
trust models for, 461–463  
types of, 417  
virtual, 419–422  
workload and performance prediction, 457–461

GridWay, 442

GSI, *See Globus Security Infrastructure (GSI)*

GT4 library, 446–449, 447, 467

  data management using, 451–452  
  monitoring and discovery, 452, 452

Guest systems, 157

Guest-based manager, 159

Gustafson’s law, 47–48

## H

Hadoop, 43, 43–44, 342, 343, 354, 355–359  
  architecture, 357–358, 357  
  K means clustering for, 353  
  performance on ClueWeb data set, 355  
  running a job in, 358–359, 358  
  WebVisCounter program implementation, 364–365

Hadoop Distributed File System (HDFS), 242, 242, 355–357

Hardware, clusters and, 83

Hardware abstraction level virtualization, 132–134

Hardware as a Service (HaaS), 235

Hardware layer, 88

Hardware virtualization, 26, 221–223

Hardware-assisted virtualization, 135–138, 148

Hbase, 341

HBM module, 447, 447

Health care applications, of social networks, 606–607

Health monitoring, 586, 586

Heartbeats, 99

Heterogeneity, 45, 68

dynamic, 153

in peer resources, 486

Hierarchical access model, 426, 427

Hierarchical classification, 436

Hierarchically structured overlay networks, 501–505, 501

Hierarchy, virtualization, 153–154

High availability (HA), 48, 66, 83, 95–99

High availability (HA) clusters, 71

High availability (HA) systems, 456

High level language (HLL), 132

High throughput, 6

High-bandwidth interconnects, 80–81, 80

High-CPU instances, 382

High-flux computing, 6

High-level data analysis languages

comparison of, 366

DryadLINQ, 361–365

Pig Latin, 365–368

Sawzall, 365–368

High-memory instances, 381, 382

High-performance computing and communication (HPCC), 431

High-performance computing (HPC), 4–6, 5, 8

applications, 9–10

Cyclone Cloud for, 550–551

MOSIX applications for, 111

performance, 561–576

resource requirements, 169

High-performance virtual storage, 158–159

High-throughput computing (HTC), 4, 5, 6, 8, 66, 192

applications, 9–10

performance, 561–576

Himalaya Cluster, 67

Histogram

in DryadLINQ, 362

of word-count problem, 363–364

Historical interval, 458, 460

Home directory, 89–90

Home node, 88

Homogeneity, 68–69

Homogenous design, 77–79

Honeynets, 177

Honeypots, 177

Hopper, 74–75

Horizontal scalability, 570

Host channel adapter (HCA), 82

Host failure, 89

Host node, 88

Host systems, 157

Host-based IDS (HIDS), 176

Host-based manager, 159

Host-based virtualization, 141–143

Hot standby clusters, 99

HPC, *See* High-performance computing (HPC)

HPC grids, 419

HTC, *See* High-throughput computing (HTC)

HTTP, 274, 278

HUBzero, 294, 297–301

    architecture, 298–299, 298

    content tagging, 299

    future directions, 300–301

    operational features, 299

    ratings and citations, 299

    usage metrics, 300

    user support area, 299–300

    wikis and blogs, 300

Human resources (HR), 35

Hybrid access model, 426, 427

Hybrid clouds, 194, 194, 199, 217

Hybrid model (H-Model), 460, 460

Hybrid node design, 77–78

Hype cycle, 11, 12

Hypercalls, 140, 143

Hypervisors, 23, 134, 140–141

## I

IaaS, *See* Infrastructure as a Service (IaaS)

IBM, 228

    cloud projects, 547–548

    RC2 Cloud, 548–550

IBM 360, 4

IBM Blue Cloud, 193, 228, 238, 549, 550

IBM Ensembles, 547

IBM Grid Toolbox, 44

IBM Roadrunner, 73, 119–120

    architecture, 119

InfiniBand interconnect, 120  
message-passing performance, 120  
processor chip and compute blade design, 119–120

IBM SP2, 66, 67

IBM Sysplex, 66

ICE Cube container, 211, 211

Identifier space, 496

Identity authorities, 465

Identity spoofing attacks, 523

Identity-based signatures (IBSes), 533

Illegal file downloads, 490, 531, 533

ILP, *See* Instruction-level parallelism (ILP)

Image management, 387, 388

Images service, 373

Inactive state, 159

InChlGoogle, 288–289

Incremental checkpoint, 102

Independent checkpointing, 104

Independent cluster manager, 159

Independent scheduling, 106

Independent service management, 244

Index servers, 482, 486, 510

InfiniBand, 22, 80, 80–81

- architecture, 82, 82
- interconnect, 120

Information consumption, 547

Information grids, 417

Information leakage, 50

Information Power Grid, 417

Information services, 284

Infrastructure as a Service (IaaS), 35, 36, 49, 200–203, 201, 224, 227, 235–236, 336, 572–574

RFID and, 576

Infrastructure cloud features, 338

Infrastructure layer, of cloud architecture, 218, 218

Infrastructure services, 284

Initial reputation aggregation, 520

INRIA cluster, 165, 166

Instant messaging, 483

Instruction set architecture (ISA) level, 130, 131, 132

Instruction-level parallelism (ILP), 9, 14

Integrated clusters, 159

Integrity, 49–51, 50

Integrity protection, 467

Intel hardware-assisted CPU virtualization, 147–148, 148

Intel Pentium 4, 14

Intel x86 processor, *See X86 processors*

Intellectual property, 50

Interactive jobs, 107

InterCloud architecture, 247, 248

Inter-cloud resource management, 234–249

Inter-clouds, 40

Interconnection networks, 208–211, 214

Interconnects, 80, 80–83

Interface definition language (IDL), 37

Intergid gateway (IGG), 241, 241

InterGrid, 239, 241, 244–246

Inter-module connection networks, 213

International Electrotechnical Commission (IEC), 581

International grids, 428–430, 429

International Organization for Standardization (ISO), 581

# Internet

development of, 416

scalable computing over, 4–13

## Internet clouds, *See* Clouds

## Internet computing, 4–8

definition of, 7

## Internet conferencing, 293

## Internet micropayment systems, 11

## Internet of Things (IoT), 6, 11–13

applications of, 590–597, 591

architecture, 579, 579–580

definition of, 7

development of, 6, 577

enabling technologies for, 576–590, 578

GPS and, 587–590

mobility and security support for, 561, 561

RFID and, 579, 580–582

sensor networks and, 582–587, 586

technology roadmap for, 578

ubiquitous computing and, 576–580

## Internet Protocol Security (IPsec), 37

## Internet Suspend-Resume (ISR), 162

## Internet traffic, 480, 481

## Internode communication, 70

## Interoperability, 69, 226–227, 462

## Intrusion detection systems (IDSes), 50, 462

distributed, 252–253

host-based, 176

network-based, 176

VM-based, 176–179

I/O address space, single, 92, 93

I/O node, 88

I/O virtualization, 150–153, 152

IP addresses,  $2^{128}$ , 13

IPv6 protocol, 13

IRods, 295, 309

Isolated redundancy, 98

IT as a service, 200

IT growth, 29

Iterative MapReduce, 351–355, 353, 371

## J

Jaguar, *See Cray XT5 Jaguar*

Java, 37–38

Java Data Object (JDO), 370

Java Message Service (JMS), 37, 105, 291

Java Persistence API (JPA), 370

Java Virtual Machine (JVM), 133

JavaSpace, 313

JBoss, 280

JBPM project, 280

JDO, *See Java Data Object (JDO)*

JLP, *See Job-level parallelism (JLP)*

JMS, *See Java Message Service (JMS)*

JNDI (Jini and Java Naming Directory Interface), 38

Job execution environments, 312–314

Job management, 70

Job management systems (JMS), 93, 95, 107–109

  administration, 107

  cluster job types, 107

  cluster workload characteristics, 107–108

features of, 108–109

migration schemes, 108

Job response time, 45

Job scheduler, 107

Job scheduling methods, 104–106, 105

resource management and, 435–437

selfish grids and, 457

space sharing, 105–106

tiling, 105, 106

time sharing, 106

Job-level parallelism (JLP), 9

Joint ventures, 421

Joyent Cloud, 202

JPA, *See Java Persistence API (JPA)*

JSR-311 (JAX0RS), 275

JVM, *See Java Virtual Machine (JVM)*

## K

K means clustering, 353, 354

Kad network, 497

Kademlia, 499

KaZaA, 482, 483, 493–494, 494, 502–503, 502, 537

Kernel layer, 88

Kernel level, checkpointing at, 101–102

Kesselman, Carl, 416

Kimberley, 560

Kleinberg, Jon, 607–608

Klienrock, Leonard, 6

Knowledge grids, 417

Khronos Group, 86

KVM (kernel-based VM), 144, 146

# L

- Laboratory Information Systems (LIMS), 316
- LAMP architecture, 298
- Language Integrated Query (LINQ), 361–365
- Latency processor, 84
- LDAP (Lightweight Directory Access Protocol), 38
- Libraries, checkpointing at, 101–102
- Library support level virtualization, 132, 138–140
- Lightweight Directory Access Protocol (LDAP), 392, 430
- LimeWire, 537
- Linda programming model, 313
- Link layer, 418
- Linked Environments for Atmospheric Discovery (LEAD), 302
- LinkedIn, 606
- Linux clusters, MOSIX2 for, 41–42, 110
- Linux OS, 72, 137
- Linux vServer, 137
- Live migration, 159–162, 160, 165
- Load balancing, 158, 202, 380, 385
- Load index, 457
- Load information managers (LIMs), 109
- Load management, 107
- Load sharing, 107
  - Load sharing facility (LSF), 109–110, 554
    - application of, on cluster, 110
    - architecture, 109
    - utility commands, 109–110

Load-balancing clusters, 71  
Local area networks (LANs), 21–22, 27  
    virtual, 235  
Local bridge, 602  
Local files, 90  
Local jobs, 107  
    competition between cluster jobs and, 106  
Local node, 88  
Local Resource Management System (LRMS), 389  
Local scheduler, 448  
Local scheduling, 106  
Local storage, 90  
Local trust scores, 518, 520  
Location as a Service (LaaS), 235  
Location transparency, 88, 91  
Location-aware request distribution algorithms (LARD), 312  
Location-based service (LBS), 587  
Logistics services, 592  
Look-ahead random walk (LRW), 520  
Loosely synchronous applications, 369, 371  
Low-cost design philosophy, 25–26  
Lsbatch, 109  
Lshosts, 109  
LSLIB (load sharing library), 109  
Lsmake, 110  
Lsrun, 109  
Lstcsh, 109  
Lstools, 109  
Lxrun, 138, 139

Machine memory, 148

Machine size, 45

Magellan, 565

Mainframe computers, 4

Malware-based attacks, 249, 251

Man-in-the-middle attacks, 225, 251, 255, 255

Manjrasoft, 205, 393–398

Many-core chip multiprocessors (CMPs), 153, 155

Many-core GPU processors, 15

*Map* function, 43, 345–346, 348

Mapping, 344, 368–370

MapReduce, 26, 203, 312–313, 336

architecture, 357, 357–358

benchmarking, 574–576

comparisons with, 342

compute-data affinity, 351

control flow implementation, 352

data and control flow, 348–351

data flow implementation, 350

definition of, 346

description and features, 42, 43

formal notation of data flow, 348

framework, 345

Hadoop implementation, 355–359, 364–365

iterative, 351–355, 353, 371

logical data flow, 346–348, 347

performance, 575, 576

problem-solving strategy, 348

scalability, 558

MapReduce++, 353, 369, 370, 371

Mashups, *See* Cloud mashups

Massive parallelism, 17, 18, 19

clustering for, 66–75

GPU clusters for, 83–87

Massive systems, 27, 27

Massively distributed systems, 8

Massively parallel processors (MPPs), 5, 69, 369, 416, 546

architecture, 75–87

design issues, 69–71

packaging, 77–79

Master node, 99–100

Master-worker model, 349

Matchmaking, 312

Matrix multiplication, 86

Maya GUI, 398

MC Cluster, 67

MCAT, 309

M/DB, 341

MDCube (Modularized Datacenter Cube), 213, 214

MDS module, 447, 450, 452

Mean square error (MSE), 460

Mean time to failure (MTTF), 48, 98, 100–101, 102

Mean time to repair (MTTR), 48, 98, 100–101

Medical applications, of social networks, 606

Megabits per second (Mbps), 14

Megastations, 70

Memcache, 372

Memex, 547

Memory

machine, 148

physical, 148

shared, 77, 369, 370

virtual, 130, 148

Memory access time, 20

Memory bus, 77

Memory controllers (MCs), 18, 84

Memory management unit (MMU), 148

Memory migration, 162

Memory space, single, 92

Memory technology, 20, 21

Memory ushering, 111, 112

Memory virtualization, 148–150, 149–150

Mesh access model, 426

Mesh network sensors, 11

Message bus, 289–290, 290

Message filtering, 291

Message notification, 291

Message passing, 7

Message protection, 466–467

Message queuing, 308

Message-level security, 467

Message-oriented middleware, 289–294

enterprise bus, 289–290, 290

publish-subscribe model, 291

queuing and messaging systems, 291, 292

Message-passing interface (MPI), 42–43, 43, 71, 208, 352, 353, 371

benchmarking, 574–576

K means clustering, 354

Metadata, 304, 308–309

Metadata catalogs, 308–309

Metadata Catalog Service (MCS), [309](#)

METADATA table, [378–379](#)

Metaproblems, [369](#)

Micro instances, [381, 382](#)

Microblogging, [611–613](#)

Micro-kernel architecture, [140](#)

Microprocessors, advances in, [14](#)

Microsoft, [548, 573](#)

Microsoft Azure, [193, 203, 229, 233–234, 336, 548](#)

Affinity Groups, [340](#)

features of, [385](#)

platform, [234](#)

programming support, [384–387](#)

roles, [341](#)

service offerings, [228](#)

SQLAzure, [385](#)

tables, [386–387](#)

virtualization support, [224](#)

virtualized resources, [223](#)

Microsoft Dryad, *See* [Dryad](#)

Microsoft Sharepoint, [205](#)

Middleware

cloud or grid applications, [291–294](#)

clusters and, [83](#)

grid, [302, 443–455](#)

message-oriented, [289–294](#)

in multitier architecture, [283](#)

open source, [444–446](#)

support, [444–446](#)

support for SSI clustering, [94, 95](#)

support for virtualization, 138, 138–140

virtualization, 22–24

Middleware layer, 88, 52, 53

Migration

file system, 162–163

live, of virtual machines, 159–162, 160, 165

memory, 162

network, 163–164

using Xen, 164–165, 165

Migration overhead, 108

Migration schemes, 108

Millions of instructions per second (MIPS), 14, 45

MIMD (Multiple Instruction Multiple Data), 368

MIPS, *See Millions of instructions per second (MIPS)*

Mobile cloud computing, 558–561

Modular data centers, 211–212, 211

construction of, 211–212

interconnection networks, 212–213

Modular packaging, 78–79

Monadic access model, 426, 427

Monitoring systems, 313

Monolithic hypervisor architecture, 140

Moore's law, 8, 14, 22

MOSIX, 67, 110–112

MOSIX2, 41–42, 110

Mosso, 548

Motion Picture Association of America (MPAA), 533

MPI, *See Message-passing interface (MPI)*

MPPs, *See Massively parallel processors (MPPs)*

MQSeries, 290

MTTF, *See* Mean time to failure (MTTF)

MTTR, *See* Mean time to repair (MTTR)

MuleMQ, 291, 292

Multi-core chips, 26

Multicore CPUs, 14–16

    architecture, 15, 16

    schematic, 15

    virtualization in, 153–154, 154

Multiple connections, 89

Multiplexing, 24

Multiplicity of OS images, 46, 46

Multipoint Control Unit (MCU), 291

Multitasking scalability, 569–570

Multithreading, 9

Multithreading technologies, 14–16, 16

Multitier architecture, 282, 282–283

Multi-wavelength (MWL) detectors, 296

Mutual authentication, 468–469, 469

MyGrid Project, 312

Myrinet, 80–81, 80

MySpace, 599

## N

NanoHUB.org gateway, 296–297

Nanotechnology, 296

Napster, 482, 482, 493, 494

NaradaBrokering, 291, 292–293

NAS, *See* Network attached storage (NAS)

NAS Grid, 428

Nash equilibrium, 457

National Aeronautics and Space Administration (NASA), 196, 552–555

National grid projects, 428–430

Native mode, 110

Nebula, 552–553, 553

Nebulae, 18, 73, 74–75

NEES.org, 298

.NET, 229, 417

Net-centric services, 287–288

NetSuite, 548

Network address translator (NAT), 484

Network as a Service (NaaS), 235

Network attached storage (NAS), 21, 22, 163

Network bandwidth, 8, 45, 193

Network defense technologies, 50

Network File System (NFS), 90

Network for Computational Nanotechnology (NCN), 296

Network interface card (NIC), 82

Network latency, 45

Network layer, 52, 53–54

Network migration, 163–164

Network on chip (NoC), 18, 84

Network operating systems, 40

Network proximity, 487, 487, 507–509

Network threats, 49–51, 50

Network-based IDS (NIDS), 176

Network-based systems, technologies for, 13–26

Networking

  data-center, 194–196, 195

  single, 91, 92

Networks

expandability of, 209  
local area, 21, 22, 27, 235  
overlay, *See* Overlay networks  
peer-to-peer (P2P), *See* Peer-to-peer (P2P) networks  
sensor, 582–587  
social, *See* Online social networks (OSNs)  
storage area, 21, 22, 163  
structure of, 209  
virtual private network (VPN), 28, 433  
wide-area, 22

Networks of clusters, 7

New technologies, hype cycle of, 11, 12

*New York Times*, 205

Nexus, 447, 447

Nimbus, 172, 173, 336, 387–389, 388

Nimbus Context Broker, 555

Nimbus Web, 387

Nimrod-G, 442

Node activeness, 99

Node architecture, 74, 77–79

Node availability, 108

Node degree, 601

Node failures, 510

Nonrelational databases, 341

Nonuniform memory access (NUMA), 46

Nordic D-Grid, 429

NOSQL, 376–379

NOSQL nonrelational databases, 341, 370

No-trust systems, 522

Nova, 392

NOW project, 66, 67, 75

NSF TeraGrid, 30, 417

NUMA, *See* Nonuniform memory access (NUMA)

N-version programming, 98–99

NVIDIA, 17, 18, 87

NVIDIA Fermi GPU, 18, 19

## O

Object Management Group, 444

Observational Medical Outcomes Partnership (OMOP), 340

OGCE Gadget container, 301, 303

OGSA, *See* Open Grid Services Architecture (OGSA)

OGSA-DAI (Globus Data Access and Integration), 452

Onion routing, 488

Online piracy, 50, 490

Online social networks (OSNs), 546, 597–600

    attack tracking, 608

    benefits of, 600

    characteristics of, 597–600

    communities and applications of, 603–608

    Facebook, 608–611

    graph-theoretic analysis of, 600–603, 601, 603

    privacy breaches, 607

    representative, 599, 599

    structure of, 598

    Twitter, 611–613

Online transaction processing (OLTP), 66

*OnStart()* method, 384

*OnStop()* method, 384

Open Cirrus, 565

Open Cloud Consortium (OCC), 565

Open Cloud Testbed, 565

Open Computing Language, 86

Open Gateway Computing Environments (OGCE) project, 294, 301–303

- gadget container, 303
- messaging service, 301
- packaging, 303
- scientific application management, 302–303
- workflows, 302

Open Grid Forum (OGF), 438, 444

Open Grid Services Architecture (OGSA), 44, 44, 280, 283, 285, 422–424, 444

- framework, 422
- Grid Service Handles (GSHs), 423
- grid service migration, 423–424
- interfaces, 422–423, 423
- security models, 424, 425
- Semantic OGSA (S-OGSA), 312

Open Grid Services Infrastructure (OGSI), 284

Open MPI Project, 120

Open Science Grid, 294, 303

Open Social standards, 303

Open Software Foundation (OSF), 41

Open source grid middleware, 444–446

Open Systems Interconnection (OSI) layers, 37

Open Virtualization Format (OVF), 226

OpenBabel service, 288–289

OpenMP, 338

OpenNebula, 173, 200, 244, 389–393, 390

OpenStack, 391–392, 392

OpenStack Compute, 392–393

OpenStack Storage, 393

OpenVZ, 136–137, 137, 299

Operate-repair cycle, 96

Operating system level virtualization, 132, 135

Operating systems (OSs)

cloud, for virtualized data centers, 172–176

cluster, 76–77

distributed, 42–44

extensions, 83

network, 41

of Top 500, 72–73

Oracle, 308

Oracle Streams Advanced Queuing, 308

Orchestration and Director Engine (ODE), 302

OS extensions, 135–136

OS image count, 46–47, 46

OSCAR3 service, 288–289

OSF, *See Open Software Foundation (OSF)*

Overlay networks, 32, 32–33, 492–496

complexity analysis, 503–505

hierarchically structured, 501–505

network proximity in, 507–509

routing in, 505–513

security, 522–523

structured, 490–491, 498–501, 507, 524

trust overlay networks, 517–519, 518

unstructured, 490, 492–496, 509, 524

OVirt, 199

OWL, 310

OWL-S ontology, 310

PaaS, *See Platform as a Service (PaaS)*

Packaging, 68

Packet switching, 83

Page blobs, 385

Pan-STARRS workflow, 318–319

Parallax, 171–172, 172

Parallel and distributed programming

advantages of, 343–370

definition of, 343

Dryad, 359–365

DryadLINQ, 361–365

Hadoop, 355

MapReduce, 345–355

motivation for, 344–345

paradigms, 344–345

Pig Latin, 365–368

Sawzall, 365–368

Twister, 351–355

Parallel and distributed systems, mapping applications to, 368–?370

Parallel computer, 7

Parallel computing, 6

system classification, 27

Parallel data transfer, 427

Parallel jobs, 107

Parallel processing, 7

Parallel programming, 7

in clusters, 77

CUDA, 86

models, 42–44, 43

with Sawzall and Pig Latin, 367–368

Parallel programs, 7, 103, 103

Parallel Virtual Machine (PVM), 43, 71, 111

Parallelism, 8

*See also* Massive parallelism

bit-level, 9

data-level, 9, 319

degree of parallelism (DoP), 9

functional, 319

instruction-level, 9, 14

job-level, 9

massive, 17, 18, 19

task-level, 9

Para-virtualization, 143–145

architecture, 143–144, 143

with compiler support, 144–145

for I/O virtualization, 150

KVM (kernel-based VM), 144

VMware ESX Server for, 144–145, 145

Partial failures, 97

Partitioning, 344, 348, 604

*Partitioning* function, 349, 350

*PartitionKey* property, 386

Passwords, 464, 468–470

PAST storage utility, 497

Pastry, 498, 508–509

Path length, 601

Patterson, David, 6

Paused state, 160

Peak speed ( $R_{\text{peak}}$ ), 73

Peer Authorization Protocol (PAP), 534, 536

Peer nodes, search of, 486

Peer trust and reputation systems, 514–517

Peer-to-peer (P2P) computing, 456, 480–492

basic concepts, 480–485

challenges in, 486–490

Peer-to-peer (P2P) grids, 31, 417

Peer-to-peer (P2P) networks, 5, 8, 27, 28, 32, 32, 417, 480–?492

applications, 33, 483, 483–485

architecture, 32

challenges, 33–34

characteristics of, 480

vs. client/server networks, 482, 482

complexity analysis, 503–505, 504

content delivery networks, 529–532

copyright protection and, 533–534

DHT implementation, 519–520, 519

for distributed file sharing, 493–496

distributed hash tables (DHTs), 496–497

effective search in, 486

examples of, 483–484

failure recovery, 509–511

fault tolerance, 509–511

file sharing, 523–538

free-riding problem in, 488

hierarchically structured, 501–505

models, 482–483, 482

network proximity in, 507–509

overlay networks, 32–33, 492–505

peer trust and reputation systems, 514–517, 518, 520–522

performance, 34

piracy prevention in, 535–538

proximity aware, 487

routing in, 505–507

search algorithms, 524–528

security, 522

structured, 498–501, 524

taxonomy of, 490–492, 491

trust and reputation management in, 489

trust overlay networks, 517–519

unstructured, 492–496, 524

PeerTrust, 517, 518

Performance

benchmarking, 574–576

bottlenecks, 226, 242

of clouds, 561–576

of distributed systems, 561–576

of HPC/HTC systems, 568–572

prediction, for grids, 457–461

scalable, 46, 70

of Top 500, 72

unpredictable, 226

Performance metrics, 45–48

Peripheral component interconnect (PCI), 82

Permanent failures, 96–97

Personal computers (PCs), 4

Petri Nets, 316

P-Grid, 497

PharmaHUB.org, 300

Physical clusters

definition of, 155

vs. virtual clusters, 156–159, 156

Physical memory, 148

Pig Latin, 365–368, 366–367

Piracy, 50

Piracy prevention, 490, 535–538

Pirate download time, 536–538

Pirate success rate, 536–538

Pitney Bowes, 206

PKI-based trust model, 462, 464–465, 464

Planned shutdowns, 96, 98

Platform Abstraction Layer (PAL), 396

Platform as a Service (PaaS), 35, 36, 49, 200, 201, 203–206, 204, 227, 235–236, 336, 573

Platform evolution, 4–5

Platform layer, of cloud architecture, 218, 218

Platforms, 483

Pleasingly parallel applications, 369

Plumtree, 513

Pods, 210

Policy authorities, 465

Popularity-driven resource provisioning, 238–239, 240

Portals, 294–303, 338

PortTypes, 284

POST, 274

PostgreSQL, 307

Power consumption, 21, 224

Power efficiency

- of CPU, 19–20
- of GPU, 19–20

Power management, 585–586

PowerTrust, 517, 518, 520, 520–522

Presentation layer, 282

Prestige, 602

Primary components, 98

Privacy, 252, 339, 341, 607

Privacy protection, 255–256

Private clouds, 193–194, 194, 196–200, 552–555

Private key generator (PKG), 533

Private keys, 464, 469

Private registries, 304

Privileged instructions, 147

Proactive content poisoning, 535, 535

Problem size scalability, 45

Process space, 93

Processing daemon, 360

Process-level virtualization, 132

Processor speed, 8, 14, 14, 45

Producer/consumer model, 437, 437–438

Production grids, 444

Professional networking, 606

Program library, 340

ProgrammableWeb.com, 306

Programming models

Amazon Web Services (AWS), 379–387

cloud, 343

Google App Engine (GAE), 370–379

Microsoft Azure, 384–387

parallel and distributed, 42–44, 343–370

Proximity neighbor selection, 509

Proximity routing, 508–509

Proximity subgroups, 508

Proxy certificates, 466, 468–469

Proxy credentials, 464–465  
Public clouds, 193, 194, 199  
    providers, 202–203  
    service offerings, 227–229, 228  
    virtualization support in, 224  
Public Key Infrastructure (PKI), 252, 424, 464–465, 561  
Public keys, 464, 468–469  
Public registries, 304  
PublisherAssertion, 305  
Publish-subscribe model, 291, 307–308, 308, 312  
PubMed, 288–289  
PUT, 274  
*Put* operation, 497  
PVM, *See* Parallel Virtual Machine (PVM)  
Python, 370

## Q

Quadrics, 80, 80–81  
QuakeSim, 293–294  
Quality of cloud service (QoCS), 572, 574  
Quality of Service (QoS), 196, 221, 247–248, 487  
Queuing, 291  
Queuing services, 341  
Queuing systems, 292  
Queuing time, 436

## R

Rackspace, 548  
Rackspace Cloud, 202  
Radiality, 602  
Radio-frequency identification (RFID), 6, 11, 576, 580–582, 581

Random graphs, 493  
Random walk algorithms, 525  
Randomized colluder detection, 535–536, 536  
Rappture toolkit, 299  
Rarest first (RF), 526  
RC2, *See* Research Compute Cloud (RC2)  
RDF, *See* Resource Description Framework (RDF)  
Reach, 601  
Read-only RFID tags, 580  
Read-write RFID tags, 581  
Record Industry Association of America (RIAA), 533  
Recovery, 70–71, 83, 100–101  
Recovery time, 99  
Recruitment threshold, 108  
*Reduce* function, 43, 345–347, 349, 351, 365  
Redundancy, 70–71, 95–99  
    isolated, 98  
    techniques, 98  
Registries  
    categories of information in, 304  
    private, 304  
    public, 304  
    service registries, 304–306  
    shared (semiprivate), 304  
    UDDI, 304–306  
Regular random walk, 520  
Relational Database Service (RDS), 232, 379  
Relational databases, 340–341  
Reliability, 8, 10, 98–99  
Reliability, availability, and serviceability (RAS), 95–96

Remote execution server (RES), 109

Remote files, 90

Remote Method Invocation (RMI), 38

Remote nodes, 88

Remote Procedure Call (RPC), 37

Remote storage, 90

Replay prevention, 467

Replica Location Service (RLS), 309

Replication, 528

Representational State Transfer (REST), 273, 273–277

    architecture, 275

    principles of, 273

    queuing services and, 341

Reputation management, in P2P networks, 489

Reputation sharing, 227

Reputation systems, 257–260, 456, 456

    for clouds, 258–259

    design options, 257–258, 258

    in P2P networks, 514, 518, 520

    trust overlay networks, 259–260, 260

Reputation vector, 520

Reputation-based trust model, 257–260, 462–463

Research clouds, 562–566

Research Compute Cloud (RC2), 194–196

Research grids, 444

Resource Description Framework (RDF), 310

Resource layer, 52, 53, 419

Resource management

*See also* Grid resource management

    inter-cloud, 234–249

    optimization of, 170

services, 284

virtual clusters and, 155–169

Resource management layer, 282, 283

Resource management system (RMS), 388, 435–437, 435

Resource manager, 107

Resource monitoring, 437, 437

Resource provisioning, 237–243, 238

of compute resources, 237–238

demand-driven, 239, 240

dynamic, 239–241

event-driven, 238–239, 240

methods, 238–239

popularity-driven, 238–239, 240

of storage resources, 242–243

underprovisioning, 238

Resource sharing, 7, 77, 78

Resource-pulling model, 465, 465

REST, *See* Representational State Transfer (REST)

REST systems, 38

RESTful Web services

in Amazon S3, 275–276

interaction with, 274

Web 2.0 applications and, 276

Restlet, 275

Retailing, IoT and, 591–594

RFID, *See* Radio-frequency identification (RFID)

RFID tags, 580–582

RFT (Reliable File Transfer), 448, 451

Ringside social application servers, 610

RISC processors, 15

RLS (Replica Location Service), 452

Roadrunner, *See IBM Roadrunner*

Robotic garden project, 596–597

Rollback recovery, 70, 100

Root directories, 137

Rootkits, 251

Routing, in P2P networks, 505–507

Routing efficiency, 487

*RowKey* property, 386

*Run()* method, 384

*RunJob(conf)* function, 358

Runtime environment, 451

Runtime support services, 237

## S

SaaS, *See Software as a Service (SaaS)*

SAGA (Simple API for Grid Applications), 444

Salesforce.com, 205, 228, 229, 548, 552

SAML protocol, 466

SAN, *See Storage area networks (SANs)*

Savitsky-Golay filter, 460

Sawzall, 365–366, 365–368

SAXPY, 86

Saxpy\_serial routine, 86

Scalability, 8, 10, 68, 176, 558

application, 45

cloud, 226–227

dimensions of, 45

horizontal, 570

multitasking, 569–570

vs. OS image count, 46, 46–47

of P2P networks, 481, 486

size, 45

software, 45

vertical, 570

Scalable coherence interface (SCI), 77

Scalable computing, 4–13

Scalable parallel computing environments, 338

Scalable performance, 46, 70

Scaled-workload speedup, 47

Scale-free graphs, 493

Scheduling, 344

Science and research clouds, 562–566

Science Clouds, 566

Science gateways, 294, 294–303, 338

exemplars, 295–297

HUBzero, 297–301

Open Gateway Computing Environments (OGCE) project, 301–303

Scratch space, 91

SDKs (software development kits), 419

Seagate Barracuda XT, 20, 21

Search algorithms, 524–528

Sector/Sphere, 336, 390–391, 391

Secure Overlay Services (SOS), 523

Secure Shell (SSH), 245

Secure Sockets Layer (SSL), 468

Security, 176

authentication, 464–465, 468–469

authorization methods, 464–465

cloud, 225, 249–260, 339, 561

cluster, 69

Globus Security Infrastructure (GSI), 466–470

grid, 455, 461–463

infrastructure, 50–51

Internet Protocol Security (IPsec), 37

message-level, 467

network threats and data integrity, 49–51

OGSA security model, 424, 425

P2P networks, 489, 522–523

piracy prevention, 535–538

responsibilities, 49

services, 284

transport-level, 467, 469

trust models, 461–463

trusted zones, 177–179

VM-based intrusion detection systems, 176–179

Security as a Service (SaaS), 235, 260

Security demand (SD), 462–463

Security index, 571

Security resilience, 45

SEG (Scheduler Event Generator), 448

Selfish grids, 457

Self-management services, 285

Self-organization, 481

Semantic grids, 309–312, 311, 417

Semantic OGSA (S-OGSA), 312

Semantic Web, 309–312, 417

Semiprivate registries, 304

Sensor networks, 582–586

Sensor service (SS), 39

Sensor technologies, 6

Sequential bottleneck, 47

SERENDIP, 485

Serial jobs, 107

Server clusters, 194–195

Server consolidation, 169–170

Server nodes, 109

Server-centric design, 210, 212–213

Servers

    energy consumption of unused, 51

    underutilization of, 170

Service accounting, 439–440

Service availability, 225

Service computing, 7

Service grounding, 310

Service models, 310

    IaaS, 200–203

    PaaS, 200, 203–205, 204

    SaaS, 200, 205–206

Service nodes, 77

Service profile, 310

Service registries, 304–306

Serviceability, 95

Service-level agreements (SLAs), 10, 193, 200, 229, 237, 248, 558

Service-oriented architecture (SOA), 26, 37–40, 216, 271–322, 422

    characteristics of, 272

    discovery, registries, metadata, and databases, 304–314

    DoD net-centric services, 287–288

    enterprise multitier architecture, 282–283

    evolution of, 39, 39–40

    layered architecture for Web services and grids, 37, 37–38

REST, 273–277

science gateways, 294–303

services and, 272–289

Web services and, 38, 277–281

workflow in, 314–322

## Services

CICC, 288–289

discovery, 304

grid, 279, 283–287

net-centric, 287–288

OGSA, 284

queuing, 341

RESTful Web services, 273–277

Web, *See* Web services

## ServiceTypes, 284

Session Initiation Protocol (SIP), 483

SETI@Home, 30, 87, 419, 429, 485, 492

SGE, *See* Sun Grid Engine (SGE)

SGI Cyclone, 550–552, 551

SGI Origin 3000, 66

SGI ProPack5, 72

Shadow page table, 150

Shared memory, 77, 78, 369, 369

Shared registries, 304

Shared-disk architecture, 77, 78

Signature matching, 253

SIMD (single instruction, multiple data), 9, 368, 369

Simon, Herbert, 547

Simple Cloud APIs, 341

Simple Network Management Protocol (SNMP), 92

Simple Notification Service (SNS), [379](#)  
Simple Object Access Protocol (SOAP), [38](#), [278–281](#), [382](#), [466](#)  
Simple Queue Service (SQS), [379](#)  
Simple Storage Service, *See* [Amazon S3 \(Simple Storage Service\)](#)  
SimpleDB, [336](#), [379](#), [383–384](#)  
SimpleGrid project, [294](#)  
Simultaneous multithreaded (SMT) processor, [16](#)  
Single Cloud File Storage API, [340](#)  
Single entry point, [89](#)  
Single file hierarchy, [90](#)  
Single instruction, multiple data (SIMD), [71](#)  
Single memory space, [92](#), [92](#)  
Single networking, [91–93](#), [92](#)  
Single point of control, [88](#), [92](#), [92](#)  
Single point of failure, [48](#), [97](#), [97](#)  
Single point of management, [92](#)  
Single-chip cloud computer (SCC), [87](#)  
Single-precision (SP) arithmetic, [83](#)  
Single-system image (SSI), [29](#), [41](#), [70](#)  
    features, [87–95](#)  
    in MOSIX2, [110–111](#)  
Size scalability, [45](#)  
Skip-list-based systems, [498](#)  
SkipNet, [504](#), [505](#)  
Sky Computing, [566](#)  
Skype, [483–484](#)  
Slack clusters, [68](#)  
SLEs10, [72](#)  
Sloan Digital Sky Survey, [567](#)  
Small-world graphs, [493](#)

Smart buildings, 594

Smart power grids, 594, 595

SMP, *See* Symmetric multiprocessor (SMP)

Snapshots, 103–104

SOA, *See* Service-oriented architecture (SOA)

SOAP, *See* Simple Object Access Protocol (SOAP)

SOAP messages, 280, 280

SOAP-based Web services, 277–282

Social circles/clusters, 602

Social engine, 610

Social network analysis, 600–603

Social networks, *See* Online social networks (OSNs)

Software

- clusters and, 83
- for grid computing, 443–455
- licensing, 227
- support, 444–446
- virtualization, 221

Software as a Service (SaaS), 35, 36, 49, 192, 200, 205–206, 227, 229, 235, 343, 572, 574

Software bugs, 226

Software layer, for virtualization, 130

Software scalability, 45

Software stacks, for cloud computing, 236

Solid-state drives (SSDs), 20

SourceForge.net, 297, 303

SP2 Server Cluster, 67, 98

Space, 45

Space sharing, 105–106

Special function units (SFUs), 18

Speed performance, 5, 72

Speedup factor, 47

Sphere, 336, 390–391

SPMD (single-program and multiple data), 368

SPRESI services, 288–289

Spyware, 251

SQL, 340–341

SQL Data Services (SDS), 386

SQL queries, 313

SQLAzure, 385

SSDs, *See* Solid-state drives (SSDs)

SSI, *See* Single-system image (SSI)

Stable storage, 90

Standard instances, 381, 382

Standardization, 226–227, 257, 278

Stateless interactions, 274

Static mode, 104

Static priority, 104

Static replication, 425

Stay scheme, 106

Storage

Azure, 384–387

cloud, 340

distributed, 226

local, 90

provisioning of, 242–243, 242

remote, 90

stable, 90

virtual, 171–172

Storage area networks (SANs), 21, 22, 163

Storage Resource Broker (SRB), 309

Storage technology, 20

Storage virtualization, 171–172, 224

Storm botnet, 497

Streaming multiprocessors (SMs), 18, 84

Streaming processors (SPs), 18, 19

Striped data transfer, 427

Structural cohesion, 602–603

Structural equivalence, 602

Structural holes, 602

Structured overlay networks, 33, 490–491, 498, 498–501, 502–503

- hierarchical, 501–505
- network proximity in, 507–509
- search algorithms, 524–528

Subgrids, 314

Subject-push model, 465, 465

*SubmitJob()* function, 358

Subscription, 305

Sun Grid Engine (SGE), 440, 444, 446

Sun Niagara 2, 14

Sun One, 417

Super nodes, 502–503, 510

Supercomputers, 5

- See also* Top 500 computer systems
- case studies, 112–120
- data centers and, 195
- vs. DISC systems, 567, 568
- top five in 2010, 73, 74
- virtual, 416, 419–422

Superscalar processor, 15

Supply chain management, 592–594, 593

Suspended state, 159

Sustained speed ( $R_{\max}$ ), 73  
Swarm-based delivery methods, 532  
Swift workflow system, 321–322, 322  
Switch-centric design, 210–211  
Symmetric multiprocessor (SMP), 46, 144  
Symmetry, 88, 91  
Synchronization, 344, 349  
Synchronous applications, 369  
Synergistic technologies, 577–579, 578  
System availability, 45, 48, 49, 570–571  
System defense technologies, 50  
System directories, 90  
System efficiency, 47, 73, 570  
System of systems, 40  
System throughput, 45, 569–570  
System-area interconnects, 21, 22  
Systems of systems, 273–277

## T

Table-lookup search, 506, 506–507  
Tables, 341  
    Azure, 386–387  
    finger, 499, 506  
Tablet location hierarchy, 378–379, 378  
Target channel adapter (TCA), 82  
Task-level parallelism (TLP), 9, 14  
Taverna, 318, 319  
Tcsh, 109  
Technologies  
    convergence of, 26

fusion, 456–457

hype cycle of new, 11, 12

scalability, 45–48

Telnet command, 29

Template VMs, 157–158, 244–245

Tera-floating-point operations per second (Tflops/second), 485

TeraGrid, 30, 301, 303, 428, 430–431, 432

TeraGrid Science Gateway, 295

TerraDir, 504, 505

Tesla, 17

Tesla GPU, 18, 87

Tflops (tera floating-point operations per second), 45

Third-party applications, 244

Throughput per watt, 8

Tianhe-1A, 18, 73, 74–75, 78, 112–116

  ARCH fat-tree interconnect, 113

  architecture, 112, 113

  hardware implementation, 112–114

  Linpack Benchmark results, 115–116

  planned applications, 115–116

  power consumption, space, and cost, 115

  software stack, 115, 115

Tiling, 105–106, 105

Time, 45

Time sharing, 106

Tit-for-tat (TFT), 526

Tivoli Provisioning Manager, 550

Tivoli Service Automation Manager (TSAM), 547

TLP, *See* Task-level parallelism (TLP)

TModel, 305

Top 500 computer systems, 5, 66

analysis of, 71–75

application share, 74

ARCH fat-tree interconnect, 114

architectural evolution, 71–72, 72

case studies, 112–120

cluster architecture, 28

country share, 74

Cray XT5 Jaguar, 116–119

IBM Roadrunner, 119–120

operating systems, 72

performance, 72–73

power versus performance, 75, 75

share of system interconnects in, 81–83, 81

Tianhe-1A, 112

top five in 2010, 73, 74–75

Topic-based systems, 307

Total execution time, 569

Total failures, 97

ToxTree service, 288–289

TPS (transactions per second), 45

Transaction servers, 533

Transient failures, 96

Translation lookaside buffer (TLB), 148

Transparency

location, 88, 91

in programming environments, 42, 42

Transport-level security (TLS), 466–467, 469

TreadMarks, 67

Tree-based delivery methods, 532

Tree-structure systems, 498

Trident, 337

Trojan horses, 251

Trust delegation, 469–470, 470

Trust delegation chain, 465

Trust index (TI), 462–463

Trust management

  clouds, 249–260

  in P2P networks, 489, 514–523

  in virtualized data centers, 176–179

Trust matrix, 515–516

Trust models, 461–463

  fuzzy, 463

  generalized model, 462, 463

  PKI-based, 464–465

  reputation-based, 257–260, 463

Trust overlay networks (TONs), 259–260, 260, 517–519, 518

Trusted zones, 177–179, 178

Tsubame, 18, 75

TSUBAME 2.0, 73, 74

Tuecke, Steve, 416

Twister, 342, 343, 351–355, 353–355

Twitter, 480, 597, 599, 611–613, 612

Two-core CMP, 15

## U

Ubiquitous computing, 546–561

  clouds for, 546–561

  definition of, 546

  Internet of Things and, 576–590

  wireless sensor networks and, 582

UDDI (Universal Description, Discovery, and Integration), 38, 304–306, 305  
UDDI Custody and Ownership Transfer API, 306  
UDDI Inquiry API, 306  
UDDI Publisher API, 306  
UDDI Replication API, 306  
UDDI Security API, 306  
UDDI Subscription API, 306  
UltraScan Biophysics Gateway, 295–296  
UNICORE, 444  
Unified namespace, 425–426  
Uniform Resource Identifiers (URIs), 273, 275  
Uninterruptible power supply (UPS), 25  
Universal Business Registry, 306  
Universal Description, Discovery, and Integration (UDDI), 278  
UNIX semantics, 91  
Unplanned failures, 96  
Unstructured overlay networks, 33, 73, 509  
URL Fetch operation, 372  
User authentication, 373  
User Datagram Protocol (UDP), 390  
User interface, 77, 93  
User server, 107  
User-application level virtualization, 132–133  
User-defined type (UDT), 390  
User-directed checkpointing, 103  
Usernames, 468–469  
User-oriented reputation systems, 258  
User-space (US) protocol, 98  
Utility computing, 7, 10–11, 26  
UUHash, 502

# V

Varuna database service, 288–289

VAX Series, 4

VAXcluster, 67

VCenter, 174

VCompute, 175

VCUDA, 138–139

VCuda, 139–140

Vector machines, 9

Venus-C, 566

Vertical scalability, 570

VIOLIN cluster, 165, 166, 167–169

Virtual appliances, 23, 398

Virtual clusters

application partitioning and, 157

cloud computing and, 194

dynamic deployment of, 165–169

fast deployment and effective scheduling, 158

management of, 159

migration of memory, files, and network resources, 162–?165

vs. physical clusters, 156, 156–159

resource management and, 155–169

trusted zones for, 177–179

Virtual disk images (VDIs), 172

Virtual execution environment (VE), 135

Virtual grids, 419

Virtual hierarchies, 153–154

Virtual infrastructures, 24

Virtual infrastructures (VI) managers, 172, 173, 199

Virtual instruction set architecture (V-ISA), 132

Virtual LANs, 235

Virtual machine monitor (VMM), 23, 130, 131, 237, 244

binary translation using, 142

comparison of, 134

design requirements and providers, 133–135

trust management and, 176–179

Virtual machine (VM) managers, 243, 244

Virtual machines (VMs), 22–24, 130

architectures, 23

bare-metal, 23

cloning, 224–225

cloud computing and, 193, 222

deployment, 158

initialization of, 156

management of, 243–246

migration, 159–162, 160–161

Parallel Virtual Machine (PVM), 43

performance effects, 159–162

primitive operations, 23–24, 24

provisioning of, 237–243

provisioning of, to virtual clusters, 156

states, 159

storage, 171

synthesis, 560

template, 157–158, 244–245

Virtual memory, 130, 148

Virtual Network Computing (VNC), 298

Virtual organizations (VOs), 30, 338, 416, 420–422, 421

Virtual platforms, 30

Virtual Private Cloud (VPC), 201–203, 202

Virtual private networks (VPNs), 28, 433

Virtual Processor ID (VPID), 149

Virtual reality (VR) applications, 13

Virtual root directories, 137

Virtual storage, 158–159

management of, 171–172

Virtual supercomputers, 416, 419–420

Virtualization, 6

binary translation with full, 141–143

cloud architecture and, 221–225

cloud security and, 251

CPU, 147–148

data center, 25–26

for data-center automation, 169–179

hardware-assisted, 145, 148

hardware-level, 132–133

host-based, 141–143

for IaaS, 224

implementation levels of, 130–140, 131, 133

I/O, 150–153, 152

on Linux or Windows platforms, 137–138

memory, 148–150, 149–150

middleware support for, 138, 138–140

in multicore processors, 153–154, 154

at OS level, 135–138

para-virtualization, 143–145

in public clouds, 224

storage, 224

structures/tools and mechanisms, 140–145

VMM design requirements and providers, 133–135

Virtualization layer, 130

Virtualization middleware, 22–24

Virtualization software, 221

Virtualized data centers

- automation, 169–179

- cloud OSes for, 172–176

- server consolidation, 169–170

- storage management, 171–172

- trust management in, 176–179

Virtualizing integration managers, 223

Viruses, 49

Visual MainWin, 138, 139

Visual Studio, 386

VLIW (very long instruction word) architectures, 9

VLSI microprocessors, 4

VM image management, 387, 388

VM Orchestrator, 199

VMApp, 153

VMDriver, 152

VMNet, 153

VMware ESX Server, 144–145, 145, 156

VMware vSphere 4, 173, 174–176

VMware Workstation, 146, 152–153, 174

VNetwork, 175

Voice over IP (VoIP), 483

VOTables, 288–289

VSphere 4, 173, 174–176, 199

VStorage, 175

VT-i technique, 146

VT-x technique, 146

# W

- WABI, *See Windows Application Binary Interface (WABI)*
- Wal-Mart data warehouse, 567
- Warps, 18
- Watermarking, 256, 256–257, 531
- Web 2.0 services, 5, 6, 26
- Web Application Description Language (WADL), 274
- Web pages, 242
- Web roles, 341–342, 384
- Web Service Choreography Description Language (WS-CDL), 317
- Web Service Choreography Interface (WSCI), 317
- Web service core, 451
- Web services, 277, 422
- application-specific, 301
  - definition of, 277
  - layered architecture for, 37–38, 37
  - multitier architecture, 282–283
  - RESTful, 273–277
  - SOAP-based, 278
  - standardization, 278
  - tools and, 38
  - WS-\* standards, 281–282
  - WS-I protocol stack, 278–281
- Web Services Atomic Transaction (WS-Atomic Transaction), 317
- Web Services Composite Application Framework (WS-CAF), 317
- Web Services Context (WS-CTX), 317
- Web Services Conversation Language (WSCL), 317
- Web Services Coordination (WS-Coordination), 317
- Web Services Description Language (WSDL), 37, 277–278, 286
- Web Services Flow Language (WSFL), 317

Web Services Reliable Messaging (WSRM), 37  
Web Services Resource Framework (WSRF), 286, 286, 451  
Web traffic, 480  
WebMDS service, 452  
WebSphere, 417  
WebSphere CloudBurst, 547  
WebSphereMQ, 290, 292  
WebVisCounter program, 364–365, 364  
White pages, 304  
Wide-area networking (WAN), 22, 27  
Windows, virtualization on, 137–138  
Windows Application Binary Interface (WABI), 138–139, 138  
Windows Azure, *See Microsoft Azure*  
Wine, 138, 139  
Wireless home area network (WHAN), 582  
Wireless personal area network (WPAN), 584  
Wireless sensor applications, 591  
Wireless sensor networks (WSNs), 582, 584  
Worker roles, 341–342, 384  
Workflow Management Coalition, 316  
Workflows  
    architecture and specification, 317–319  
    basic concepts, 315–316  
    clouds and, 337  
    execution engine, 319–321  
    grids and, 338  
    in SOA, 314–322  
    specification, 318–319  
    standards, 316, 317  
Swift workflow system, 321–322, 322

Workload management, 107

Workload prediction

adaptive, 458, 458–459

adaptive hybrid model (AH-Model), 460

auto-regression (AR) method, 459, 461

for grid computing, 457–461

on Grid'5000, 459, 461

H-Model for, 460

Workstations, availability, 98

World Wide Web Consortium (W3C), 272

World Wide Web (WWW), 417

WorldGrid, 429

Worms, 251

Write-once-read-many (WORM) tags, 581

WS containers, 451

WS mashups, 26

WS-\* standards, 281

WS-Addressing, 286

WS-BusinessActivity, 317

WSDL, *See* Web Services Description Language (WSDL)

WS-Eventing, 291

WS-I protocol stack, 278–281, 279

WS-Notification (WSN), 286, 291

WSRE (Web Service Resource Framework), 444

WS-RM (Reliable Messaging), 281

WS-Secure Conversation, 466–467

WS-Security, 466–467, 469

WS-Trust, 466, 469

X

X.509 certificates, 252, 468

X.509 standard, 466, 468

X86 processors, 15, 25, 66, 146

XBaya, 301

Xen, 140–141, 141, 146, 164–165, 238, 244

XenServer, 156

Xeon X5670, 17

Xing.com, 606

XLANG, 317

XML, 278–279

XML schemas, 38

XRegistry, 301

## Y

YaCy, 497

Yahoo! Pipes, 252

Yellow pages, 304

YouTube, 480, 599

## Z

ZigBee coordinator, 584

ZigBee end device, 584

ZigBee networks, 582–587, 585

ZigBee router, 584

Zolera Soap Infrastructure (ZSI), 280

Zukerberg, Mark, 611

# Distributed and Cloud Computing

From Parallel Processing to the Internet of Things

Kai Hwang · Geoffrey C. Fox · Jack J. Dongarra

**MK**  
MORGAN KAUFMANN