

Ravishankar K. Iyer | Zbigniew T. Kalbarczyk
Nithin M. Nakka

Dependable Computing

Design and Assessment

 **IEEE Press**



 **IEEE**
computer
society

WILEY

Dependable Computing

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Sarah Spurgeon, *Editor in Chief*

Moeness Amin	Ekram Hossain	Desineni Subbaram Naidu
Jón Atli Benediktsson	Brian Johnson	Tony Q. S. Quek
Adam Drobot	Hai Li	Behzad Razavi
James Duncan	James Lyke	Thomas Robertazzi
	Joydeep Mitra	Diomidis Spinellis

About IEEE Computer Society

IEEE Computer Society is the world's leading computing membership organization and the trusted information and career-development source for a global workforce of technology leaders including: professors, researchers, software engineers, IT professionals, employers, and students. The unmatched source for technology information, inspiration, and collaboration, the IEEE Computer Society is the source that computing professionals trust to provide high-quality, stage-of-the-art information on an on-demand basis. The Computer Society provides a wide range of forums for top minds to come together, including technical conferences, publication, and a comprehensive digital library, unique training webinars, professional training, and the Tech Leader Training Partner Program to help organizations increase their staff's technical knowledge and expertise, as well as the personalized information tool my Computer. To find out more about the community for technology leaders, visit <http://www.computer.org>.

IEEE/Wiley Partnership

The IEEE Computer Society and Wiley partnership allows the CS Press authored book program to produce a number of exciting new titles in areas of computer science, computing, and networking with a special focus on software engineering. IEEE Computer Society members receive a 35% discount on Wiley titles by using their member discount code. Please contact IEEE Press for details.

To submit questions about the program or send proposals, please contact Mary Hatcher, Editor, Wiley-IEEE Press: Email: mhatcher@wiley.com, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774.

Dependable Computing

Design and Assessment

Ravishankar K. Iyer

*Department of Electrical and Computer Engineering and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA*

Zbigniew T. Kalbarczyk

*Department of Electrical and Computer Engineering and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA*

Nithin M. Nakka

*Cisco Networking Engineering group
Cisco Systems, Inc.
San Jose, California, USA*

IEEE
 computer
society

 **IEEEPress**

WILEY

Copyright © 2024 by the IEEE Computer Society. All rights reserved.

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

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data for:

Hardback ISBN: 9781118709443

Cover Design: Wiley

Cover Image: © Yuichiro Chino/Getty Images

Set in 9.5/12.5pt STIXTwoText by Straive, Pondicherry, India

This book is dedicated to my wife Pamela for her unwavering support and encouragement in my academic pursuits, from my PhD to the present day.

– For Ravishankar K. Iyer

This book is dedicated to my wife, whose unfailing support kept me moving forward to the completion of this work.

– For Zbigniew T. Kalbarczyk

This book is dedicated to my parents, whose sacrifices have shaped what I am.

– For Nithin M. Nakka

Contents

About the Authors	<i>xxiii</i>
Preface	<i>xxv</i>
Acknowledgments	<i>xxvii</i>
About the Companion Website	<i>xxix</i>

1	Dependability Concepts and Taxonomy	1
1.1	Introduction	1
1.2	Placing Classical Dependability Techniques in Perspective	2
1.3	Taxonomy of Dependable Computing	4
1.3.1	Faults, Errors, and Failures	5
1.4	Fault Classes	6
1.5	The Fault Cycle and Dependability Measures	6
1.6	Fault and Error Classification	7
1.6.1	Hardware Faults	8
1.6.2	Software Faults and Errors	8
1.6.2.1	The GUARDIAN90 Operating System	9
1.6.2.2	IBM-MVS (zOS) and IBM Database Management Systems	9
1.7	Mean Time Between Failures	11
1.8	User-perceived System Dependability	13
1.9	Technology Trends and Failure Behavior	14
1.10	Issues at the Hardware Level	15
1.11	Issues at the Platform Level	17
1.12	What is Unique About this Book?	18
1.13	Overview of the Book	19
	References	20
2	Classical Dependability Techniques and Modern Computing Systems: Where and How Do They Meet?	25
2.1	Illustrative Case Studies of Design for Dependability	25
2.1.1	IBM System S/360	25

2.1.2	The Tandem Integrity System	26
2.1.3	Blue Waters	28
2.2	Cloud Computing: A Rapidly Expanding Computing Paradigm	31
2.2.1	Layered Architecture of Cloud Computing	32
2.2.2	Reliability Issues in Cloud Computing	34
2.3	New Application Domains	37
2.3.1	Smart Power Grid Application	38
2.3.2	Business Integrity Assurance Application	40
2.3.3	Medical Devices and Systems	42
2.3.3.1	Monitoring of Soldiers for Blast Impact in a Battlefield Scenario	44
2.3.3.2	Teleoperated Surgical Robots	46
2.3.4	Wireless Sensor Networks	46
2.3.5	Mobile Phones	47
2.3.6	Artificial Intelligence (AI) Systems	48
2.4	Insights	52
	References	52

3 Hardware Error Detection and Recovery Through Hardware-Implemented Techniques 57

3.1	Introduction	57
3.2	Redundancy Techniques	58
3.2.1	Comparing the Reliability of Simplex and TMR Systems	60
3.2.2	M -out-of- N Systems	61
3.2.3	The Effect of a Voter	63
3.2.4	Time Redundancy	66
3.3	Watchdog Timers	67
3.3.1	Example Applications of Watchdog Timers	67
3.3.2	Limitations of Watchdog Timers	68
3.4	Information Redundancy	69
3.4.1	A Brief History of Coding Theory	69
3.4.2	Outline of the Description of Coding Techniques	71
3.4.3	Fault Detection Through Encoding	72
3.4.4	Parity	72
3.4.5	Cyclic Redundancy Checks	75
3.4.6	Checksums	79
3.4.7	Arithmetic Codes	80
3.4.7.1	AN Codes	80
3.4.7.2	Berger Codes	80
3.4.8	Residue-Inverse Residue Codes	81
3.4.9	Reed-Solomon Codes	82
3.4.10	Communication Codes and Protocols	83
3.4.10.1	Convolutional Codes	84

3.4.10.2	Communication Protocols for Reliable Transmission	85
3.4.11	Two-Level Integrated Interleaved Codes	86
3.4.12	RAID: Redundant Array of Inexpensive Disks	88
3.4.12.1	A Commercial RAID-Based Storage System	90
3.5	Capability and Consistency Checking	93
3.5.1	Capability Checking	93
3.5.2	Consistency Checking	93
3.6	Insights	93
	References	96
4	Processor Level Error Detection and Recovery	101
4.1	Introduction	101
4.2	Logic-level Techniques	104
4.2.1	Radiation Hardening	104
4.2.2	Selective Node-Level Engineering	105
4.2.3	SEU Hardening for Memory Cells	108
4.2.4	SEU-tolerant Latch	110
4.2.4.1	Recovery from a Particle Strike	111
4.2.5	Razor	112
4.2.5.1	Pipeline Error Recovery	112
4.2.5.2	Discussion	113
4.2.6	Built-in Soft-Error Resilience Using Scan Flip-Flop Reuse	114
4.2.7	Discussion	115
4.3	Error Protection in the Processors	115
4.3.1	Reliability Features of Intel P6 Processor Family	115
4.3.1.1	Machine Check Architecture (MCA)	116
4.3.1.2	Functional Redundancy Checking (FRC)	116
4.3.2	Reliability Features in Itanium	116
4.3.2.1	Protection of On-Chip Memory Arrays	116
4.3.2.2	Error Containment	117
4.3.2.3	Data Poisoning	118
4.3.2.4	Error Promotion	118
4.3.2.5	Watchdog Timer	118
4.3.2.6	Error Detection and Correction Logging	119
4.3.3	POWER7	119
4.3.4	NonStop Himalaya Systems	120
4.4	Academic Research on Hardware-level Error Protection	122
4.4.1	SRTR: Transient Fault Recovery Using Simultaneous Multithreading	126
4.4.1.1	Discussion	128
4.4.2	DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design	128

- 4.4.2.1 Discussion 129
- 4.4.3 Microprocessor-based Introspection (MBI) 131
- 4.4.4 Phoenix: Detection and Recovery from Permanent Process Design Bugs 133
- 4.5 Insights 134
- References 137

5 Hardware Error Detection Through Software-Implemented Techniques 141

- 5.1 Introduction 141
- 5.2 Duplication-based Software Detection Techniques 142
 - 5.2.1 Examples of Software-based Duplication Techniques 144
 - 5.2.1.1 Duplication at the Level of Source Code 145
 - 5.2.1.2 ED⁴I 146
 - 5.3 Control-Flow Checking 146
 - 5.3.1 The State of the Art 147
 - 5.3.1.1 Hardware Schemes 147
 - 5.3.1.2 Software Schemes 152
 - 5.3.2 Enhanced Control-Flow Checking with Assertions (ECCA) 153
 - 5.3.2.1 Insertion of ECCA Assertions 153
 - 5.3.2.2 SET and TEST Assertions in ECCA 153
 - 5.3.2.3 ECCA Error Detection 155
 - 5.3.2.4 Experimental Evaluation of ECCA 156
 - 5.3.3 Preemptive Control Signature (PECOS) 158
 - 5.3.3.1 PECOS Error Detection 161
 - 5.3.3.2 Experimental Evaluation of PECOS 163
 - 5.4 Heartbeats 166
 - 5.4.1 Timeout Mechanism 167
 - 5.4.2 Limitations of Traditional Heartbeats 168
 - 5.4.3 Designing Adaptive, Smart Heartbeats 168
 - 5.4.4 Evaluation of Smart Heartbeats 171
 - 5.4.4.1 Experimental Methodology 171
 - 5.4.4.2 Experimental Results 172
 - 5.5 Assertions 173
 - 5.6 Insights 174
 - References 175

6 Software Error Detection and Recovery Through Software Analysis 179

- 6.1 Introduction 179
- 6.2 Diverse Programming 183
 - 6.2.1 N-Version Programming 183

6.2.1.1	Applications of N-Version Programming	185
6.2.2	Recovery Blocks	188
6.2.2.1	Sequential Recovery Block Scheme	189
6.2.2.2	Designing an Acceptance Test	189
6.2.2.3	Distributed Applications: Recovery Block Conversations	191
6.2.2.4	Advanced Recovery Block Models and Real-Time Systems	193
6.3	Static Analysis Techniques	194
6.3.1	ESP: Path-Sensitive Program Verification in Polynomial Time	196
6.3.2	PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code	199
6.3.3	Dynamic Derivation of Program Invariants	202
6.3.3.1	DAIKON	203
6.3.4	Statically Derived Application-Specific Detectors	206
6.3.4.1	Terms and Definitions	207
6.3.4.2	Steps in Detector Derivation	207
6.3.4.3	Example of Derived Detectors	208
6.3.4.4	Software Errors Covered	209
6.3.4.5	Hardware Errors Covered	212
6.3.4.6	Performance and Coverage Measurements	213
6.4	Error Detection Based on Dynamic Program Analysis	217
6.4.1	Fault Model	218
6.4.2	Derivation: Analysis and Design	219
6.4.2.1	Dynamic Derivation of Detectors	221
6.4.2.2	Detector Tightness and Execution Cost	221
6.4.2.3	Detector Derivation Algorithm	224
6.4.3	Experimental Evaluation	225
6.4.3.1	Application Programs	225
6.4.3.2	Infrastructure	225
6.4.3.3	Experimental Procedure	227
6.4.4	Results	228
6.4.4.1	Detection Coverage of Derived Detectors	228
6.4.4.2	False Positives	231
6.5	Processor-Level Selective Replication	233
6.5.1	Application Analysis	234
6.5.2	Overview of Selective Replication	235
6.5.3	Mechanism of Replication	236
6.6	Runtime Checking for Residual Software Bugs	239
6.6.1	Race Condition Checking in Multithreaded Programs	239
6.6.2	Array Bounds Checking	240
6.6.3	Runtime Verification	241
6.7	Data Audit	242
6.7.1	Static and Dynamic Data Check	243

6.7.2	Structural Check	243
6.7.3	Semantic Referential Integrity Check	244
6.7.4	Optimization Using Runtime Statistics	244
6.8	Application of Data Audit Techniques	246
6.8.1	Target System Software and Database Architecture	246
6.8.2	Audit Subsystem Architecture	247
6.8.2.1	The Heartbeat Element	249
6.8.2.2	The Progress Indicator Element	249
6.8.2.3	Audit Elements	249
6.8.3	Evaluating the Audit Subsystem	250
6.9	Insights	252
	References	253
7	Measurement-based Analysis of System Software: Operating System Failure Behavior	261
7.1	Introduction	261
7.2	MVS (Multiple Virtual Storage)	262
7.2.1	MVS Error Detection and Recovery Processing	263
7.2.2	MVS Error Detection	263
7.2.3	Recovery Processing	264
7.2.3.1	Hardware Error Recovery	265
7.2.3.2	MVS Software Error Recovery	266
7.2.4	Hardware-related Software Errors	266
7.2.4.1	Processing of Error Data	267
7.2.4.2	Analysis of Error Detection	269
7.2.4.3	Error Classification and Detection	269
7.2.4.4	Error Detection and Recovery	270
7.2.4.5	Detection of HW/SW Software Errors	271
7.2.5	Analysis of Hardware-related Software Errors	271
7.2.5.1	Recovery from HW/SW Errors	271
7.2.6	Summary of MVS Analysis	273
7.3	Experimental Analysis of OS Dependability	273
7.3.1	What to Measure and Why?	274
7.4	Behavior of the Linux Operating System in the Presence of Errors	275
7.4.1	Methodology	276
7.4.2	Error Injection Environment	276
7.4.2.1	Approach	276
7.4.2.2	Error Activation	278
7.4.2.3	Error Model	279
7.4.2.4	Outcome Categories	280
7.4.3	Overview of Experimental Results	282

7.4.4	Crash Cause Analysis	284
7.4.4.1	Stack Injection	284
7.4.4.2	System Register Injection	286
7.4.4.3	Code Injection	288
7.4.4.4	Data Injection	289
7.4.4.5	Summary	289
7.4.5	Crash Latency (<i>Cycles-to-Crash</i>) Analysis	290
7.4.6	Crash Severity	292
7.4.6.1	Lessons Learned	293
7.4.6.2	Value in Employing Fault Injection	294
7.4.6.3	Toolset and Benchmark Procedures	294
7.4.7	Summary	294
7.5	Evaluation of Process Pairs in Tandem GUARDIAN	295
7.5.1	Data Integrity	296
7.5.2	User Applications	296
7.5.3	Software Fault Tolerance of Process Pairs	297
7.5.3.1	Measure of Software Fault Tolerance	298
7.5.3.2	Outages Due to Software	299
7.5.3.3	Characterization of Software Fault Tolerance	300
7.5.4	Discussion	303
7.5.5	First Occurrences Versus Recurrences	304
7.5.6	Impact of Software Failures on Performance	305
7.5.7	Summary	308
7.6	Benchmarking Multiple Operating Systems: A Case Study Using Linux on Pentium, Solaris on SPARC, and AIX on POWER	308
7.6.1	Introduction of Case Study	309
7.6.2	Experimental Setup	310
7.6.2.1	Fault Model	310
7.6.2.2	Target Systems	311
7.6.2.3	Experimental Environment	311
7.6.3	Evaluation Procedure	313
7.6.3.1	Generation of Injection Targets	313
7.6.3.2	Execution of Fault Injection Campaigns	313
7.6.3.3	Collection and Analysis of Data	314
7.6.4	Results	315
7.6.4.1	Comparison of Target Platforms' Error Behavior	321
7.6.4.2	Feedback for Reliability Enhancements	322
7.6.5	Detailed Discussion and Analysis	323
7.6.5.1	Text Injection Analysis	323
7.6.5.2	Stack Injection Analysis	324
7.6.5.3	Register Injection Analysis	325

7.6.6	Conclusions	326
7.7	Dependability Overview of the Cisco Nexus Operating System	326
7.8	Evaluating Operating Systems: Related Studies	330
7.9	Insights	331
	References	332
8	Reliable Networked and Distributed Systems	337
8.1	Introduction	337
8.2	System Model	339
8.3	Failure Models	340
8.4	Agreement Protocols	342
8.4.1	Byzantine Agreement Problem: Solution	343
8.4.1.1	Oral Message Algorithm, $OM(f)$	344
8.4.2	Interactive Consistency Obtained by Running the Byzantine Agreement Protocol	345
8.5	Reliable Broadcast	346
8.5.1	Reliable Broadcast	348
8.5.2	FIFO (First-In-First-Out) Broadcast	348
8.5.3	Causal Broadcast	349
8.5.4	Total Order Broadcast	350
8.6	Reliable Group Communication	351
8.6.1	Specification of Group Communication Service	351
8.6.1.1	Specification of Group Membership Service	352
8.6.1.2	Specification of Reliable Multicast Service	354
8.6.2	Example Implementations of Group Communication Systems	357
8.7	Replication	358
8.7.1	Replication in Hardware	360
8.7.2	Replication in Software	363
8.7.2.1	Replication at the Level of the Operating System	363
8.7.2.2	Replication at the Level Between the Hardware and the Operating System	364
8.7.2.3	Replication at the Level Between the Operating System and the User Application	364
8.7.2.4	Replication at the User-Level	365
8.7.2.5	CORBA	366
8.7.3	The Problem of Nondeterminism	366
8.7.4	Paxos and Read-Write Quorums: A Practical Approach to Achieving Eventual Consistency	368
8.7.4.1	Paxos	368
8.7.5	Read-Write Quorums	369

8.8	Replication of Multithreaded Applications	370
8.8.1	System Model: Definitions and Assumptions	373
8.8.2	Specification of the LSA Algorithm	374
8.8.3	LSA Algorithm Overview	374
8.8.3.1	Failure Behavior with Error-Free Leader-to-Followers Communication	376
8.8.3.2	Failure Behavior with Byzantine Errors in Leader-to-Followers Communication	381
8.8.4	Specification of the PDS Algorithm	382
8.8.4.1	PDS-1 Algorithm Overview	382
8.8.4.2	PDS-2 Algorithm	384
8.8.5	Application-Transparent Replication Framework	385
8.8.5.1	Using the LSA and PDS Algorithms with Majority Voting	386
8.8.5.2	LSA and PDS Implementations	386
8.8.5.3	Virtual Socket Layer	386
8.8.5.4	Voter/Fanout Process	387
8.8.6	Performance-Dependability Trade-Offs	387
8.8.6.1	Performance Evaluation	388
8.8.6.2	Dependability Evaluation	390
8.8.6.3	Injecting into a Replica Process	391
8.8.6.4	Lessons Learned	395
8.8.7	Conclusions	395
8.9	Atomic Commit	396
8.9.1	The Two-Phase Commit Protocol	397
8.9.1.1	Assumptions	397
8.9.1.2	Basic Algorithm	398
8.9.1.3	Disadvantages	398
8.9.1.4	The Detailed Two-Phase Commit Protocol	398
8.10	Opportunities and Challenges in Resource-Disaggregated Cloud Data Centers	400
8.10.1	Data Movement	402
8.10.2	Data Consistency	403
8.10.3	Fault Tolerance	403
8.10.4	ML-based Orchestration and Validation	405
	References	405
9	Checkpointing and Rollback Error Recovery	413
9.1	Introduction	413
9.2	Hardware-Implemented Cache-Based Schemes Checkpointing	415
9.2.1	Cache-Aided Rollback Error Recovery (CARER) for Uniprocessors	415

9.2.2	Multiprocessor Cache-Based Schemes	418
9.2.3	ReVive: Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors	419
9.3	Memory-Based Schemes	421
9.3.1	Physical Memory-Based Schemes	421
9.3.2	Virtual Memory-Based Schemes	421
9.4	Operating-System-Level Checkpointing	424
9.4.1	<i>libckpt</i> : Transparent Checkpointing Under Unix	425
9.4.1.1	Incremental Checkpointing	425
9.4.1.2	Forked Checkpointing	425
9.4.2	Fine-Grained Rollback and Deterministic Replay for Software Debugging	426
9.4.2.1	Rollback of Multithreaded Processes	427
9.4.3	Transparent Application Checkpoint (TAC) Module	428
9.4.3.1	RMK Framework	428
9.4.3.2	RMK Pins: System-Level RMK Interface	429
9.4.3.3	Application-Level RMK Interface	430
9.4.3.4	RMK Core	430
9.4.3.5	An Example RMK Module: Transparent Application Checkpoint (TAC)	430
9.5	Compiler-Assisted Checkpointing	432
9.5.1	CATCH – Compiler-Assisted Techniques for Checkpointing	432
9.5.1.1	Potential Checkpoints	433
9.5.1.2	Sparse Potential Checkpoints	434
9.5.1.3	Adaptive Checkpointing	434
9.5.2	Compiler-Assisted Checkpointing Using <i>libckpt</i>	437
9.5.2.1	Compiler Directives	438
9.6	Error Detection and Recovery in Distributed Systems	438
9.6.1	Synchronous Checkpointing	440
9.6.2	Asynchronous Checkpointing: Message Logging	441
9.6.3	Sender-Based Message Logging	442
9.6.3.1	Design and Motivation	442
9.6.3.2	A Practical Implementation	445
9.7	Checkpointing Latency Modeling	451
9.8	Checkpointing in Main Memory Database Systems (MMDB)	455
9.8.1	Checkpointing of MMDB Control Structures	458
9.8.1.1	Checkpointing Framework	459
9.8.1.2	Incremental Checkpointing	461
9.8.1.3	Delta Checkpointing	462
9.9	Checkpointing in Distributed Database Systems	463
9.9.1	Definitions	464

9.9.2	The Algorithm	465
9.9.2.1	Failure Recovery	468
9.10	Multithreaded Checkpointing	468
9.10.1	Dealing with Nondeterminism	469
	References	470
10	Checkpointing Large-Scale Systems	475
10.1	Introduction	475
10.2	Checkpointing Techniques	476
10.2.1	Checkpoint Coordination Techniques	476
10.2.2	Shared Memory Systems	478
10.2.3	I/O Techniques	479
10.2.4	Recovery Techniques	481
10.2.4.1	Use of Spares	482
10.3	Checkpointing in Selected Existing Systems	484
10.3.1	Blue Gene	485
10.3.2	Brazos	487
10.3.3	Winckp	487
10.3.4	Condor	488
10.3.5	Libckpt	488
10.3.6	Classification of Checkpointing Approaches in Existing Systems	489
10.3.7	Example of Evaluation of Checkpointing Schemes for a Large-Scale System	490
10.3.8	Determining Optimal Checkpointing Interval	491
10.4	Modeling-Coordinated Checkpointing for Large-Scale Supercomputers	492
10.4.1	Failure and Recovery	493
10.4.2	SAN-Based Modeling	494
10.4.2.1	Modeling Compute and Checkpointing	496
10.4.2.2	Modeling Correlated Failures	499
10.4.2.3	Results	500
10.5	Checkpointing in Large-Scale Systems: A Simulation Study	502
10.6	Cooperative Checkpointing	506
10.6.1	Other Terms and Definitions	507
10.6.2	Cooperative Checkpointing vs. Periodic Checkpointing	507
	References	508
11	Internals of Fault Injection Techniques	511
11.1	Introduction	511
11.2	Historical View of Software Fault Injection	513
11.3	Fault Model Attributes	517

11.4	Compile-Time Fault Injection	517
11.4.1	Source Code Mutation	520
11.4.2	Bytecode Mutation	521
11.5	Runtime Fault Injection	521
11.5.1	Time Trigger Faults	521
11.5.2	Runtime Mutation	521
11.5.2.1	Mutation of APIs and System Call Parameters	522
11.5.2.2	Software Probe	522
11.5.2.3	Network Messaging Faults	522
11.5.3	Library-Based Faults	523
11.5.4	Performance/Timing Faults	523
11.5.5	User-Space <i>Ptrace</i> -Based Faults	524
11.5.5.1	Fault Injection Using Trap Instruction	524
11.5.5.2	Fault Injection Using Debug Register	524
11.5.6	Fault Injection Using GDB	524
11.5.7	Kernel Space	525
11.5.7.1	Kernel Fault Injection	526
11.5.7.2	Driver	526
11.5.7.3	User Virtual Address	526
11.5.8	Configurable FPGAs	527
11.5.9	Security Threats	527
11.6	Simulation-Based Fault Injection	529
11.7	Dependability Benchmark Attributes	530
11.8	Architecture of a Fault Injection Environment: NFTAPE Fault/Error Injection Framework Configured to Evaluate Linux OS	531
11.8.1	Fault Injection Environment	531
11.8.2	Approach Overview	532
11.8.3	Kernel Profiling	533
11.8.3.1	Workload	533
11.8.3.2	Profiling	534
11.8.4	Hardware Monitoring	535
11.8.5	Control Host Overview	535
11.8.5.1	Target Generator	538
11.8.5.2	Injector Manager	539
11.8.6	Kernel-Level Support	540
11.8.6.1	Injection Controller	540
11.8.7	Breakpoint Handler	543
11.8.8	Crash Handler	544
11.8.9	Crash Dumper	545
11.8.10	Component Interactions	545
11.9	ML-Based Fault Injection: Evaluating Modern Autonomous Vehicles	547

11.9.1	DriveFI: Bayesian Fault Injection Framework	548
11.9.1.1	Autonomous Driving System Overview	550
11.9.1.2	Defining Safety	551
11.9.1.3	Fault Injection	552
11.9.1.4	Case Studies	554
11.9.2	Bayesian Fault Injection	554
11.9.2.1	Kinematics-Based Model of Safety	555
11.9.2.2	Machine Learning Model Describing the System's Response Under Faults	556
11.9.3	The ADS Architecture and Simulation	560
11.9.3.1	Overview of ADS	560
11.9.3.2	Simulation Platform	561
11.9.4	DriveFI Architecture	561
11.9.4.1	Injecting into Computational Elements: GPU Fault Models	563
11.9.4.2	Injecting Faults into ADS Module Output Variables	563
11.9.5	Results	564
11.9.5.1	GPU-Level Fault Injection	566
11.9.5.2	Source-Level Fault Injections	567
11.9.5.3	Results of Bayesian FI-Based Injections	570
11.9.6	AV-Fuzzer: Fault Injection Framework Based on AI-Driven Fuzzing	572
11.9.7	Related Work	573
11.10	Insights and Concluding Remarks	574
	References	574

12 Measurement-Based Analysis of Large-Scale Clusters: Methodology

12.1	Introduction	585
12.2	Related Research	587
12.2.1	Failure Data Analysis in Specific Application Domains	590
12.2.2	Analysis of Data on Security Incidents	593
12.3	Steps in Field Failure Data Analysis	594
12.4	Failure Event Monitoring and Logging	597
12.4.1	Automated Error Logging	597
12.4.1.1	Syslog	598
12.4.1.2	Blue Waters Logs	598
12.4.1.3	IBM Z/OS Logs	601
12.4.1.4	IBM Blue Gene RAS Events	602
12.4.1.5	Windows Event Logging	604
12.4.2	Human-Generated Failure Reports	607
12.4.2.1	Bug Databases and Public User Forums	607
12.5	Data Processing	608

- 12.5.1 Data Filtering 611
 - 12.5.1.1 Example: Processing of Public Computer-Related Recalls Databases for Safety-Critical Medical Devices 612
- 12.5.2 Data Coalescence 614
 - 12.5.2.1 Time-Based Coalescence 616
 - 12.5.2.2 Problems with Time-Based Coalescence 618
 - 12.5.2.3 Example of Time-Based Spatial Coalescence of Failure Data from Blue Gene/L 620
 - 12.5.2.4 Content-Based Event Coalescence 620
- 12.6 Data Analysis 622
 - 12.6.1 Basic Statistics 622
 - 12.6.2 Repair Rates 624
 - 12.6.2.1 Example: Root Cause Analysis from 20 HPC Systems at LANL 624
 - 12.6.2.2 Example: Analysis of Smartphone Users' Failure Reports 627
 - 12.6.2.3 Example: Analysis of Failures from LANs of Windows NT Machines 628
- 12.7 Estimation of Empirical Distributions 634
 - 12.7.1 Hazard Rate Estimation 635
 - 12.7.1.1 Hazard Rate Estimation from VAXclusters 638
 - 12.7.1.2 Hazard Rate Estimation from a Software-as-a-Service Platform 639
- 12.8 Dependency Analysis 641
 - 12.8.1 Workload/Failure Dependency 642
 - 12.8.2 Failure Dependency Among Components 646
 - 12.8.2.1 Steps in Correlation Analysis 647
 - 12.8.3 Error Interaction Analysis 650
 - 12.8.3.1 Hardware-Related Software Errors 650
 - References 651
- 13 Measurement-Based Analysis of Large Systems: Case Studies 667**
 - 13.1 Introduction 667
 - 13.2 Case Study I: Failure Characterization of a Production Software-as-a-Service Cloud Platform 667
 - 13.2.1 Data Source 668
 - 13.2.2 Failure Analysis Workflow 669
 - 13.2.3 Failure Characterization 670
 - 13.2.3.1 Output of the Coalescence Process 671
 - 13.2.3.2 Key Factors Impacting Platform Failures 674
 - 13.2.3.3 Impact of Timeout Errors 676
 - 13.2.4 Failure Rate Analysis 678
 - 13.2.4.1 Trend Analysis of the Platform Failure Rate 678
 - 13.2.4.2 Impact of Platform Software Upgrades 681

13.2.4.3	Impact of the Workload Volume on the Platform Failure Rate	683
13.2.4.4	Impact of the Workload Intensity on the Platform Failure Rate	683
13.2.5	Conclusions	684
13.3	Case Study II: Analysis of Blue Waters System Failures	686
13.3.1	Data and Methodology	690
13.3.1.1	Characterization Methodology	692
13.3.2	Blue Waters Failure Causes	693
13.3.2.1	Breakdown of Failures	693
13.3.2.2	Effectiveness of Failover	698
13.3.3	Hardware Error Resiliency	700
13.3.3.1	Rate of Uncorrectable Errors Across Different Node Types	701
13.3.3.2	Hardware Failure Rates	703
13.3.3.3	Hardware Failure Trends	704
13.3.4	Characterization of Systemwide Outages	706
13.3.5	Conclusions	708
13.4	Case Study III: Autonomous Vehicles: Analysis of Human-Generated Data	710
13.4.1	Examples of AV-Related Accidents	711
13.4.2	AV System Description and Data Collection	714
13.4.2.1	AV Hierarchical Control Structure	715
13.4.2.2	Data Sources	717
13.4.3	Data-Analysis Workflow: Parsing, Filtering, Normalization, and NLP	718
13.4.4	Statistical Analysis of Failures in AVs	721
13.4.4.1	Analysis of AV Disengagement Reports	722
13.4.4.2	Analysis of AV Accident Reports	730
13.4.5	Discussion	733
13.4.6	Limitations of this Study	734
13.4.7	Related Work	735
13.4.8	Insights and Conclusions	736
	References	737
14	The Future: Dependable and Trustworthy AI Systems	745
14.1	Introduction	745
14.2	Building Trustworthy AI Systems	748
14.2.1	An AI System and Its Key Components	748
14.2.2	A System Perspective on Trust in AI Systems	751
14.3	Offline Identification of Deficiencies	753
14.3.1	Assessment and Validation of a System and Its Design	753
14.3.1.1	Formal Verification	753
14.3.1.2	Traditional End-to-End Random Fault Injection	758

14.3.1.3	Model-driven Fuzzing, Falsification, and Fault Injection	758
14.3.2	Post-Mortem Analysis to Track the Causes of Incidents Systematically	760
14.3.2.1	Adversarial Learning: A Red Team Approach	761
14.3.2.2	Adversarial Learning: A Systematic Approach to Mislead AI Systems	761
14.3.2.3	Generative Adversarial Networks	763
14.3.3	Smart Malware with Self-Learning Capabilities	768
14.4	Online Detection and Mitigation	769
14.4.1	Formalization	769
14.4.2	Monitoring	769
14.4.3	Mitigation	770
14.5	Trust Model Formulation	772
14.5.1	An Illustrative Trust Model	772
14.6	Modeling the Trustworthiness of Critical Applications	775
14.6.1	Autonomous Vehicles and Transportation	775
14.6.1.1	Addressing Uncertainty	777
14.6.2	Large-Scale Computing Infrastructure	778
14.6.2.1	Model Formulation	779
14.6.2.2	Addressing Uncertainty	780
14.6.3	Healthcare AI/ML	781
14.6.3.1	Model Formulation	782
14.6.3.2	Addressing Uncertainty	785
14.7	Conclusion: How Can We Make AI Systems Trustworthy?	786
	References	788

Index	797
--------------	-----

About the Authors

Professor Ravishankar K. Iyer is George and Ann Fisher Distinguished Professor of Engineering at the University of Illinois Urbana-Champaign. He holds joint appointments in the Departments of Electrical and Computer Engineering (ECE) and Computer Science, the Coordinated Science Laboratory (CSL), the National Center for Supercomputing Applications (NCSA), the Carle Illinois College of Medicine, and the Carl R. Woese Institute for Genomic Biology. He is also a faculty Research Affiliate at the Mayo Clinic. Professor Iyer was the founding Chief Scientist of the Information Trust Institute at UIUC – a campus-wide research center addressing security, reliability, and safety issues in critical infrastructures. He leads the DEPEND Group at CSL/ECE at Illinois, with a multidisciplinary focus on systems and software that combine deep measurement-driven analytics and machine learning with applications in two important domains: (i) management and control of large infrastructures including autonomous systems that span resilience, safety, and security and performance and (ii) health and personalized medicine that spans computational genomics and health analytics focused on neurological disorders, pharmacogenomics, and predicting cancer metastases. His group has developed a rich AI analytics framework that has been deployed on real-world applications in collaboration with industry, health providers, and government agencies including the National Science Foundation, the National Institutes of Health, the Department of Energy, the Defense Advanced Research Projects Agency, and the Department of Defense.

Professor Iyer is a Fellow of the American Association for the Advancement of Science (AAAS), the Institute of Electrical and Electronics Engineers (IEEE), and the Association for Computing Machinery (ACM). He has received several awards, including the Jean-Claude Laprie Award, IEEE Emanuel R. Piore Award, and the 2011 Outstanding Contributions Award by the Association of Computing Machinery. Professor Iyer is also the recipient of the degree of Doctor Honoris Causa from Toulouse Sabatier University in France.

Dr. Zbigniew T. Kalbarczyk is Research Professor at the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory of the UIUC. Dr. Kalbarczyk's research interests are in the design and validation

of reliable and secure computing systems. His current work explores: (i) emerging computing technologies, such as resource virtualization to provide redundancy and assure system resiliency to accidental errors and malicious attacks; (ii) machine learning-based methods for early detection of security attacks, including defense against smart malware; (ii) analysis of data on failures and security attacks in large computing systems to characterize system resiliency and guide development of methods for rapid diagnosis and runtime detection of problems; and (iv) development of techniques for automated validation and benchmarking of dependable and secure computing systems using formal (e.g., model checking) and experimental methods (e.g., fault/attack injection). Dr. Kalbarczyk led the design and commercialization of (i) ARMOR high-availability software middleware to support resilient distributed applications and (ii) NFTAPE software framework to support fault injection-based resiliency assessment. He served as a program chair of Dependable Computing and Communication Symposium (DCCS), a track of the International Conference on Dependable Systems and Networks (DSN) 2007, and Program Co-Chair of Computer Performance and Dependability Symposium, a track of the DSN 2002. He was Associate Editor of IEEE Transactions on Dependable and Secure Computing. He has published over 230 technical papers and is regularly invited to give tutorials and lectures on issues related to the design and assessment of complex computing systems. He is a member of the IEEE, the IEEE Computer Society, and IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance.

Dr. Nithin M. Nakka received his BTech (hons.) degree from the Indian Institute of Technology, Kharagpur and his MS and PhD degrees at the University of Illinois at UIUC. Currently, he is Technical Leader at Cisco Systems. While at Cisco, he has worked on most layers of the networking software stack, from network data-plane hardware, control plane at Layer-2 and Layer-3, network controllers, up to and including network fabric monitoring. This along with doing what he enjoys – sharing his expertise through mentoring many incoming employees. He has been leading the development of solutions for network day-2 operations to monitor fabrics and analyze, understand, troubleshoot network issues, and possibly predict impending network failures. His innovative approximation heuristics in Algorithms for Memory Array Redundancy Analysis during his work at Nextest Systems Corporation brought the company to world-class excellence in this niche computational problem domain, generally known to be NP-complete. He also worked for Motorola's mobile devices group, on pioneering efforts in Bluetooth stereo audio transmission (A2DP) and Bluetooth security. His areas of research interest include systems reliability, network telemetry, and hardware implemented fault tolerance. Dr. Nakka has previously held positions as a research faculty in UIUC, and Northwestern University in Evanston, Illinois, and contributed to the area of dependability in high-performance computing systems.

Preface

Dependability of systems has transitioned over the years from a feature to a necessity for end users, and from an add-on to a core design principle for those who are designing and implementing computing or computer-based systems. The need for dependability has grown not just in its breadth in terms of the areas where it is applicable but also in depth. Given any one of the many systems where dependability techniques are applied, their relevance is seen in every layer of the system stack. The aim of this book is to help readers navigate through the evolution of dependability, from taxonomy, mathematical concepts, and fundamental theory to design, implementation, validation, deployment, measurement, and monitoring. Finally, the book brings its audience right up to the modernity of the field by looking at critical societal applications such as autonomous vehicles, large-scale clouds, and engineering solutions for healthcare, illustrating the emerging challenges faced in making artificial intelligence (AI) and its applications dependable and trustworthy.

Sections of the book are intensely pedantic and technical. However, with the support of practical case studies and use cases from both academia and real-world deployments, we have attempted to guide our audience through their journey in fathoming the developments in this ever-growing field. For a beginner, a systematic study from the beginning will help in building strong foundations, but we encourage all readers to whet their appetite with any of the case studies that spark their interest. For seasoned designers and academicians in the area, we attempt to provide a near-current reference for dependability research and development.

The prerequisites for the content of this book are a basic understanding of statistical concepts, computer systems and organization, and, preferably, a course on distributed systems. Above all, a keen interest in delving into this exciting field to unravel and possibly discover new techniques will maintain a reader's enthusiasm, as it has done ours over the past years. Certainly, well-written texts are already available in this area. However, the authors felt that we lacked a single compendium spanning the myriad areas in which dependability has been applied,

providing theoretical concepts and applied knowledge with content that would excite a beginner yet rigor that would satisfy an expert. That feeling led us to embark on the long journey of bringing forth this book.

Chapters 1 and 2 describe dependability taxonomy and briefly compare and contrast classical techniques with their modern counterparts or extensions. Chapters 3–7 help the readers walk up the system stack, from the hardware logic via operating systems up to software applications, with respect to how those layers are hardened for dependability. Chapters 8–12 expand into the domain of distributed systems to explore the techniques and applications therein. Those chapters also delve a great deal into a measurement-based understanding of the systems being studied, an aspect that the authors feel honored to have had the opportunity to significantly contribute. Chapter 13 focuses on the most recent and upcoming trends that are shaping developments in dependability. Finally, looking into the future, Chapter 14 delves deeper into the novel challenges that are being faced in making AI systems dependable and trustworthy.

In summary, with the support of practical case studies and use cases from both academia and real-world deployments, we guide our audience through a journey of developments, including the impact of AI and machine learning on this ever-growing field.

Acknowledgments

In writing this book, we were inspired by Professor Dan Siewiorek's groundbreaking research and the unmatched book *The Theory and Practice of Reliable System Design* by Siewiorek and Swarz, now in its third edition, as well as the foundational work of Professors Ed McCluskey and Al Avizienis, which continues to impact the field today.

We are indebted to many of our current and former students, postdoctoral associates, and academic and industry colleagues whose research contributed in important ways to material in this book, including Karthik Pattabiraman, Lelio DiMartino, Bob Horst, Saurabh Bagchi, Homa Alemzadeh, Long Wang, Tim Tsai, Saurabh Jha, Phuong Cao, Keywhan Chung, Shengkun Cui, and Archit Patke. Some of our colleagues have also adopted a draft version of this book in teaching dependability courses to graduate and senior students in their respective institutions, which has bolstered our confidence in the usefulness of this content. The administrative and technical proofreading staff members, including, Carol Bosley, Heidi Leerkamp, Jenny Applequist, and Kathleen Atchley, have contributed immensely to this effort by their critical linguistic polish of this technical content and also by their logistical work in keeping the authors and publishers in synchrony to accomplish this massive task. We are grateful to all of them as well as many others who shared their insights.

Special thanks to our colleagues at the University of Illinois Urbana-Champaign who provided a rich, supportive environment that allowed us to pursue this project.

The research presented in this book was supported by numerous funding agencies and industry partners, including NSF, NIH, NASA, DoD, DARPA, DOE, IBM, Sandia National Lab, Nvidia, the Mayo Clinic, Infosys, and Xilinx.

Apart from the immense technical support we have received, we are very grateful to our families, who have been ever so patient in supporting us. They have transformed their "Are we there yet?" to "Looks like we are getting close" to keep our enthusiasm alive on an emotional front while we gave all we could to tame

this mammoth. In spite of all the support that we have received both professionally and personally, added to our over 100 years of combined experience in this area, we feel that our attempts to gather all that we could in this ever-expanding and interesting field may have fallen short in some application domains, not given enough justice to some, or even at times made unintentional errors in comprehending and explaining the content. A significant portion of our time was spent in making sure that we kept the content current and relevant for our audience. However, as the field is growing at the rate that it is, we had to reconcile ourselves to the hope that we may offer more in a future edition! We invite readers to send us their feedback on the content or any errors that may have escaped our scrupulous efforts to maintain relevance and correctness.