SpringerBriefs in Computer Science

Series editors

Stan Zdonik, Brown University, Providence, Rhode Island, USA
Shashi Shekhar, University of Minnesota, Minneapolis, Minnesota, USA
Xindong Wu, University of Vermont, Burlington, Vermont, USA
Lakhmi C. Jain, University of South Australia, Adelaide, South Australia, Australia
David Padua, University of Illinois Urbana-Champaign, Urbana, Illinois, USA
Xuemin (Sherman) Shen, University of Waterloo, Waterloo, Ontario, Canada
Borko Furht, Florida Atlantic University, Boca Raton, Florida, USA
V.S. Subrahmanian, University of Maryland, College Park, Maryland, USA
Martial Hebert, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
Katsushi Ikeuchi, University of Tokyo, Tokyo, Japan
Bruno Siciliano, Università di Napoli Federico II, Napoli, Italy
Sushil Jajodia, George Mason University, Fairfax, Virginia, USA
Newton Lee, Newton Lee Laboratories, LLC, Tujunga, California, USA

More information about this series at http://www.springer.com/series/10028

Wei-Tek Tsai · Guanqiu Qi

Combinatorial Testing in Cloud Computing

Springer

Wei-Tek Tsai
Arizona State University
Tempe, AZ
USA

Guanqiu Qi
Arizona State University
Tempe, AZ
USA

ISSN 2191-5768 ISSN 2191-5776 (electronic)
SpringerBriefs in Computer Science
ISBN 978-981-10-4480-9 ISBN 978-981-10-4481-6 (eBook)
https://doi.org/10.1007/978-981-10-4481-6

Library of Congress Control Number: 2017955265

© The Author(s) 2017
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Traditional testing faces significant complexity issues due to the increasing number of data, paths, combinations, permutations, and so on. Various testing methods have been proposed and used to improve the quality and reliability of software. As one branch of software testing, combinatorial testing (CT) is one black-box testing method to identify faults caused by the interaction of a few components. CT is considered a hard problem due to its exponential complexity: A system with 30 choice items may need to explore 2^{30} combinations and this is time- and effort-consuming.

Many approaches have been proposed in the last 40 years using various theoretical models or techniques such as Latin square, orthogonal array, covering array, machine learning. In the past 20 years, the evolutionary solutions of combinatorial testing, such as AETG, IPO, have been proposed to generate a small set of test cases that achieves 100% test coverage. However, in spite of significant progress, it is still difficult to apply CT for a system of moderate sizes such as 100 choice items. These solutions mainly focus on test coverage and not much work on fault identification.

This book proposes a computational approach to address the CT problem; instead of general purpose machine learning algorithms to learn from cases, it explores the CT structure to eliminate combinations for consideration, and the exploration can be done in a parallel manner, and the process can be done in a cloud environment where parallel processing is a standard feature. This approach allows performing CT for a system with size of 2^{40}, with $2^{2^{40}}$ combinations to consider. Thus, effectively, CT can run on a large system.

This book intends to propose a faulty location analysis solution of CT as well as review existing CT solutions. Chapter 1 reviews existing combinatorial designs and CT solutions of test case generation. Chapter 2 discusses CT practical application in cloud computing and compares existing faulty location analysis solutions in CT. Chapter 3 introduces adaptive reasoning (AR) algorithm in multi-tenancy Software-as-a-Service (SaaS) system. In the next three chapters, it describes the formal definitions of test algebra (TA), discusses the related optimizations of TA, and simulates TA in cloud environment. The last three chapters propose an integrated Testing-as-a-Service (TaaS) design with AR and TA, discuss the related testing strategies, and simulate the proposed TaaS design to solve a large-scale CT problem in cloud environment.

This book can serve as reference text for graduate students and researchers who work in combinatorial testing area. It should also be interesting to those who work in fault identification, fault location, and other related fields in testing. Practitioners may be inspired by this book in testing plan design. Due to the limitation of our knowledge, we could not provide more details in relationship between existing faults and potential faults now. But our research moves forward in fault location analysis step-by-step. You are welcome to contact us, if you have any comments and suggestions.

This work is built on many outstanding CT work in the past, and their contributions are greatly appreciated. Prof. Charles Colbourn of Arizona State University, a leading expert on CT, was a co-author of some of research papers published. Prof. Wenjun Wu of Beihang University provided the initial computing environment for us to perform computational experiments, and our great friend Tony provided 40 large servers for us to perform CT experiments on a system of 2^{50} choice items. By the way, the data generated by the 2^{50} is so large that it will take months just to transfer data.

The editors at Springer, Celine Chang and Jane Li, are always helpful. We are grateful to them for their constructive suggestions and patience. Finally, we are also indebted to our family members who suffered through the writing of this work that seems to last forever.

Beijing, China and San Jose, USA
May 2017

Wei-Tek Tsai
Guanqiu Qi

Contents

1 Introduction 1
- 1.1 Software Testing 1
- 1.2 Cloud Testing 2
- 1.3 Combinatorial Designs 3
  - 1.3.1 Latin Square 3
  - 1.3.2 Orthogonal Array 3
  - 1.3.3 Covering Array 4
- 1.4 Combinatorial Testing 5
  - 1.4.1 Covering Array for Testing 6
  - 1.4.2 Automatic Efficient Test Generator 6
  - 1.4.3 In-Parameter-Order 8
  - 1.4.4 Genetic Algorithm 8
  - 1.4.5 Backtracking Algorithm 9
  - 1.4.6 Fault Detection 9
- 1.5 Structure of This Book 10
- References 11

2 Combinatorial Testing in Cloud Computing 15
- 2.1 Combinatorial Testing in Cloud Computing 15
- 2.2 Improvements of Combinatorial Testing in Cloud Environment . . . 16
- 2.3 Faulty Location Analysis in Combinatorial Testing 17
  - 2.3.1 Fault Localization Based on Failure-Inducing Combinations 17
  - 2.3.2 Identifying Failure-Inducing Combinations in a Combinatorial Test Set 18
  - 2.3.3 Faulty Interaction Identification via Constraint Solving and Optimization 19
  - 2.3.4 Characterizing Failure-Causing Parameter Interactions by Adaptive Testing 20

2.3.5 Comparisons of Existing Faulty Location Analysis Solutions 22
2.4 Related Work 22
References 23

3 Adaptive Fault Detection In Multi-tenancy Saas Systems 25
3.1 Adaptive Testing Framework 25
3.1.1 Learning from Earlier Test Results 25
3.1.2 AR Algorithm Framework 28
3.1.3 Relationship Between Faults, Final-Faults, and Candidate-Faults 29
3.2 Simulation of AR Algorithm 29
3.3 Incremental Testing to Allow New Components 32
References 36

4 Test Algebra for Concurrent Combinatorial Testing 37
4.1 Test Algebra 37
4.1.1 Learning from Previous Test Results 38
4.1.2 Changing Test Result Status 40
4.1.3 Matrix Representation 40
4.1.4 Relationship Between Configuration and Its Interactions 42
4.1.5 Merging Concurrent Testing Results 43
4.1.6 Distributive Rule 45
4.1.7 Incremental Development 45
4.2 Conclusion 46
Appendix 46
Reference 52

5 Concurrent Test Algebra Execution with Combinatorial Testing 53
5.1 TA Analysis Framework 53
5.1.1 The Role of N in Concurrent Combinatorial Testing 58
5.1.2 Modified Testing Process 58
5.2 TA Analysis Algorithm 59
5.3 TA Analysis Process and Related Considerations 60
5.3.1 Analysis Process 60
5.3.2 Adjustment in Analyzing 61
5.4 Test Database Design 62
5.4.1 X and F Table Design 62
5.4.2 P Table Design 62
5.4.3 N and U Table Design 63
5.5 Experiment 63
5.6 Conclusion 67
References 67

6 Test Algebra Execution in a Cloud Environment . . . 69
6.1 TA Concurrent Execution and Analysis . . . 69
6.1.1 TA Concurrent Execution . . . 69
6.1.2 NU Configuration . . . 71
6.1.3 NU Configuration Selection Algorithms . . . 72
6.1.4 Analysis Process of NU and U Configurations . . . 74
6.2 TA Experiments . . . 76
6.2.1 TA MapReduce Experiment Flowchart . . . 76
6.2.2 Different Configuration Numbers of TA Experiments . . . 76
6.2.3 Different Speedup Strategy for TA Experiments . . . 76
6.2.4 Different Fault Rates for TA Experiments . . . 77
6.2.5 Explanation on Simulated Data . . . 79
6.2.6 Simulation with Different Clusters . . . 79
6.2.7 Simulation using 37-node Cluster with Different Map Slots . . . 79
6.3 Conclusion . . . 81
Reference . . . 82

7 Adaptive Reasoning Algorithm with Automated Test Cases Generation and Test Algebra in Saas System . . . 83
7.1 Experimentation Using a MTA SaaS Sample . . . 83
7.2 SaaS Testing . . . 86
7.3 SaaS Test Case Generation . . . 88
7.4 Simulation and Analysis . . . 92
7.4.1 Simulation of ARP Algorithm . . . 92
7.4.2 Incremental Testing with Automatic Test Generation . . . 94
7.4.3 Simulation Experiments of ARP+PTR . . . 95
7.4.4 Analysis of the Strategy on Test Generation . . . 96
7.4.5 TA Simulation in SaaS . . . 99
References . . . 99

8 TaaS Design for Combinatorial Testing . . . 101
8.1 TaaS Introduction . . . 101
8.2 TaaS Design with TA and AR . . . 102
8.3 TaaS as SaaS . . . 105
8.3.1 GUIs . . . 107
8.3.2 Workflows . . . 107
8.3.3 Services . . . 108
8.3.4 Runtime Composition, Execution and Scalability . . . 108
8.4 Experimental Results . . . 110
8.5 Conclusion . . . 112
References . . . 112

9 Integrated Taas with Fault Detection and Test Algebra 115
9.1 Framework 115
9.1.1 Integrated Process 115
9.1.2 Framework Illustration 116
9.2 Experiments and Results 119
9.2.1 Experiment Setup 119
9.2.2 Experiment Results 122
9.2.3 Measurements 123
9.3 Conclusion 127
References 127