

Springer Series in Reliability Engineering

Lance Fiondella
Antonio Puliafito *Editors*

Principles of Performance and Reliability Modeling and Evaluation

Essays in Honor of Kishor Trivedi on his
70th Birthday

 Springer

Springer Series in Reliability Engineering

Series editor

Hoang Pham, Piscataway, USA

More information about this series at <http://www.springer.com/series/6917>

Lance Fiondella · Antonio Puliafito
Editors

Principles of Performance and Reliability Modeling and Evaluation

Essays in Honor of Kishor Trivedi
on his 70th Birthday

 Springer

Editors

Lance Fiondella
University of Massachusetts Dartmouth
Dartmouth, MA
USA

Antonio Puliafito
Università degli studi di Messina
Messina
Italy

ISSN 1614-7839 ISSN 2196-999X (electronic)
Springer Series in Reliability Engineering
ISBN 978-3-319-30597-4 ISBN 978-3-319-30599-8 (eBook)
DOI 10.1007/978-3-319-30599-8

Library of Congress Control Number: 2016933802

© Springer International Publishing Switzerland 2016

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.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

Non soffocare la tua ispirazione e la tua
immaginazione, non diventare lo schiavo del
tuo modello.

(Do not stifle your inspiration and your
imagination, do not become the slave of your
model.)

—Vincent Van Gogh

Sbagliando si impara

-proverbio italiano

(You learn by making mistakes

-Italian proverb)

*To my wife Maria and my beloved children,
Carlo and Andrea, for their unconditioned
love. To my friends, colleagues
and students representing the second half
of my existence.*

—Antonio Puliafito

I dedicate this book to కిశోర త్రివేది (Kishor Trivedi) and one branch of his legacy, including my doctoral advisor స్వప్నా గోఖలే (Swapna Gokhale) and my graduate students అనుష కృష్ణ మూర్తి (Anusha Krishna Murthy), بنت الهدى جعفرى (Bentolhoda Jafary), జెయేష్ కుమార్ (Jayesh Kumar), కార్తిక్ కాటిపల్లి (Karthik Katipally), ప్రాగ్నా స్రీనివాసన్ (Pragnya Srinivasan), సైకత భట్టాచార్య (Saikath Bhattacharya), వీరేశ్ వరాద్ బసవరాజ్ (Veeresh Varad Basavaraj), వేంకటేశ్వరన్ శేకర్ (Venkateswaran Shekar), and విద్యాశ్రీ నాగరాజు (Vidhyashree Nagaraju).

—Lance Fiondella

Contents

Part I Phase Type Distributions, Expectation Maximization Algorithms, and Probabilistic Graphical Models

Phase Type and Matrix Exponential Distributions in Stochastic Modeling	3
Andras Horvath, Marco Scarpa and Miklos Telek	
An Analytical Framework to Deal with Changing Points and Variable Distributions in Quality Assessment.	27
Dario Bruneo, Salvatore Distefano, Francesco Longo and Marco Scarpa	
Fitting Phase-Type Distributions and Markovian Arrival Processes: Algorithms and Tools	49
Hiroyuki Okamura and Tadashi Dohi	
Constant-Stress Accelerated Life-Test Models and Data Analysis for One-Shot Devices.	77
Narayanaswamy Balakrishnan, Man Ho Ling and Hon Yiu So	
Probabilistic Graphical Models for Fault Diagnosis in Complex Systems	109
Ali Abdollahi, Krishna R. Pattipati, Anuradha Kodali, Satnam Singh, Shigang Zhang and Peter B. Luh	

Part II Principles of Performance and Reliability Modeling and Evaluation

From Performability to Uncertainty.	143
Raymond A. Marie	
Sojourn Times in Dependability Modeling	169
Gerardo Rubino and Bruno Sericola	

Managed Dependability in Interacting Systems	197
Poul E. Heegaard, Bjarne E. Helvik, Gianfranco Nencioni and Jonas Wäfler	
30 Years of GreatSPN	227
Elvio Gilberto Amparore, Gianfranco Balbo, Marco Beccuti, Susanna Donatelli and Giuliana Franceschinis	
WebSPN: A Flexible Tool for the Analysis of Non-Markovian Stochastic Petri Nets	255
Francesco Longo, Marco Scarpa and Antonio Puliafito	
Modeling Availability Impact in Cloud Computing	287
Paulo Romero Martins Maciel	
Scalable Assessment and Optimization of Power Distribution Automation Networks	321
Alberto Avritzer, Lucia Happe, Anne Koziolok, Daniel Sadoc Menasche, Sindhu Suresh and Jose Yallouz	
Model Checking Two Layers of Mean-Field Models	341
Anna Kolesnichenko, Anne Remke, Pieter-Tjerk de Boer and Boudewijn R. Haverkort	
Part III Checkpointing and Queueing	
Standby Systems with Backups	373
Gregory Levitin and Liudong Xing	
Reliability Analysis of a Cloud Computing System with Replication: Using Markov Renewal Processes	401
Mitsutaka Kimura, Xufeng Zhao and Toshio Nakagawa	
Service Reliability Enhancement in Cloud by Checkpointing and Replication	425
Subrota K. Mondal, Fumio Machida and Jogesh K. Muppala	
Linear Algebraic Methods in RESTART Problems in Markovian Systems	449
Stephen Thompson, Lester Lipsky and Søren Asmussen	
Vacation Queueing Models of Service Systems Subject to Failure and Repair	481
Oliver C. Ibe	
Part IV Software Simulation, Testing, Workloads, Aging, Reliability, and Resilience	
Combined Simulation and Testing Based on Standard UML Models	499
Vitali Schneider, Anna Deitsch, Winfried Dulz and Reinhard German	

Workloads in the Clouds 525
Maria Carla Calzarossa, Marco L. Della Vedova, Luisa Massari,
Dana Petcu, Momin I.M. Tabash and Daniele Tessera

Reproducibility of Software Bugs 551
Flavio Frattini, Roberto Pietrantuono and Stefano Russo

Constraint-Based Virtualization of Industrial Networks 567
Waseem Mandarawi, Andreas Fischer, Amine Mohamed Houyou,
Hans-Peter Huth and Hermann de Meer

**Component-Oriented Reliability Assessment Approach
Based on Decision-Making Frameworks for Open
Source Software** 587
Shigeru Yamada and Yoshinobu Tamura

**Measuring the Resiliency of Extreme-Scale
Computing Environments** 609
Catello Di Martino, Zbigniew Kalbarczyk and Ravishankar Iyer

Contributors

Ali Abdollahi Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA

Elvio Gilberto Amparore Dipartimento di Informatica, Università di Torino, Turin, Italy

Søren Asmussen Department of Mathematics, Aarhus University, Aarhus C, Denmark

Alberto Avritzer Siemens Corporation, Corporate Technology, Princeton, NJ, USA

Narayanaswamy Balakrishnan Department of Mathematics and Statistics, McMaster University, Hamilton, ON, Canada

Gianfranco Balbo Dipartimento di Informatica, Università di Torino, Turin, Italy

Marco Beccuti Dipartimento di Informatica, Università di Torino, Turin, Italy

Dario Bruneo Dipartimento di Ingegneria, Università degli Studi di Messina, Messina, Italy

Maria Carla Calzarossa Dipartimento di Ingegneria Industriale e dell'Informazione, Università degli Studi di Pavia, Pavia, Italy

Pieter-Tjerk de Boer Department of Computer Science, University of Twente, Enschede, The Netherlands

Hermann de Meer Chair of Computer Networks and Computer Communications, University of Passau, Passau, Germany

Anna Deitsch Computer Science 7, Friedrich-Alexander-University of Erlangen-Nürnberg, Erlangen, Germany

Marco L. Della Vedova Dipartimento di Matematica e Fisica, Università Cattolica del Sacro Cuore, Brescia, Italy

Catello Di Martino Coordinated Science Laboratory, University of Illinois at Urbana Champaign, Urbana, IL, USA

Salvatore Distefano Higher Institute for Information Technology and Information Systems Kazan Federal University, Kazan, Russia; Dipartimento di Scienze Matematiche e Informatiche, Scienze Fisiche e Scienze della Terra, Università degli Studi di Messina, Messina, Italy

Tadashi Dohi Department of Information Engineering, Graduate School of Engineering, Hiroshima University, Higashihiroshima, Japan

Susanna Donatelli Dipartimento di Informatica, Università di Torino, Turin, Italy

Winfried Dulz Computer Science 7, Friedrich-Alexander-University of Erlangen-Nürnberg, Erlangen, Germany

Andreas Fischer Chair of Computer Networks and Computer Communications, University of Passau, Passau, Germany

Giuliana Franceschinis Dipartimento di Scienze e Innovazione Tecnologica, Università del Piemonte Orientale, Alessandria, Italy

Flavio Frattini Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, Napoli, Italy

Reinhard German Computer Science 7, Friedrich-Alexander-University of Erlangen-Nürnberg, Erlangen, Germany

Lucia Happe Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Boudewijn R. Haverkort Department of Computer Science, University of Twente, Enschede, The Netherlands

Poul E. Heegaard Department of Telematics, Norwegian University of Science and Technology, Trondheim, Norway

Bjarne E. Helvik Department of Telematics, Norwegian University of Science and Technology, Trondheim, Norway

Andras Horvath Dipartimento di Informatica, Università degli Studi di Torino, Turin, Italy

Amine Mohamed Houyou Siemens AG, Corporate Technology, Munich, Germany

Hans-Peter Huth Siemens AG, Corporate Technology, Munich, Germany

Oliver C. Ibe Department of Electrical and Computer Engineering, University of Massachusetts, Lowell, MA, USA

Ravishankar Iyer Bell Labs—Nokia, New Providence, NJ, USA

- Zbigniew Kalbarczyk** Bell Labs—Nokia, New Providence, NJ, USA
- Mitsutaka Kimura** Department of Cross Cultural Studies, Gifu City Women's College, Gifu, Japan
- Anuradha Kodali** University of California Santa Cruz, NASA Ames Research Center, Moffett Field, CA, USA
- Anna Kolesnichenko** UL Transaction Security Division, Leiden, The Netherlands
- Anne Koziolk** Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
- Gregory Levitin** The Israel Electric Corporation, Haifa, Israel
- Man Ho Ling** Department of Mathematics and Information Technology, The Hong Kong Institute of Education, Hong Kong Sar, China
- Lester Lipsky** Department of Computer Science and Engineering, 371 Fairfield Way, Unit 4155, University of Connecticut, Storrs, CT, USA
- Francesco Longo** Dipartimento di Ingegneria, Università degli Studi di Messina, Messina, Italy
- Peter B. Luh** Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA
- Fumio Machida** Laboratory for Analysis of System Dependability, Kawasaki, Japan
- Waseem Mandarawi** Chair of Computer Networks and Computer Communications, University of Passau, Passau, Germany
- Raymond A. Marie** IRISA Campus de Beaulieu, Rennes University, Rennes Cedex, France
- Paulo Romero Martins Maciel** Centro de Informática, Universidade Federal de Pernambuco, Av. Jornalista Anibal Fernandes, s/n - Cidade Universitária, Recife, Brazil
- Luisa Massari** Dipartimento di Ingegneria Industriale e dell'Informazione, Università degli Studi di Pavia, Pavia, Italy
- Daniel Sadoc Menasche** Department of Computer Science, Federal University of Rio de Janeiro, Rio de Janeiro (UFRJ), RJ, Brazil
- Subrota K. Mondal** The Hong Kong University of Science and Technology, Kowloon, Hong Kong
- Jogesh K. Muppala** The Hong Kong University of Science and Technology, Kowloon, Hong Kong

Toshio Nakagawa Department of Business Administration, Aichi Institute of Technology, Toyota, Japan

Gianfranco Nencioni Department of Telematics, Norwegian University of Science and Technology, Trondheim, Norway

Hiroyuki Okamura Department of Information Engineering, Graduate School of Engineering, Hiroshima University, Higashihiroshima, Japan

Krishna R. Pattipati Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA

Dana Petcu Departament Informatica, Universitatea de Vest din Timișoara, Timișoara, Romania

Roberto Pietrantuono Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, Napoli, Italy

Antonio Puliafito Dipartimento di Ingegneria, Università degli Studi di Messina, Messina, Italy

Anne Remke Department of Computer Science, University of Münster, Münster, Germany

Gerardo Rubino Inria Rennes—Bretagne Atlantique, Campus de Beaulieu, Rennes Cedex, France

Stefano Russo Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, Napoli, Italy

Marco Scarpa Dipartimento di Ingegneria, Università degli Studi di Messina, Messina, Italy

Vitali Schneider Computer Science 7, Friedrich-Alexander-University of Erlangen-Nürnberg, Erlangen, Germany

Bruno Sericola Inria Rennes—Bretagne Atlantique, Campus de Beaulieu, Rennes Cedex, France

Satnam Singh Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA

Hon Yiu So Department of Mathematics and Statistics, McMaster University, Hamilton, ON, Canada

Sindhu Suresh Siemens Corporation, Corporate Technology, Princeton, NJ, USA

Momin I.M. Tabash Dipartimento di Ingegneria Industriale e dell'Informazione, Università degli Studi di Pavia, Pavia, Italy

Yoshinobu Tamura Department of Electronic and Information System Engineering, Yamaguchi University, Ube-shi, Yamaguchi, Japan

Miklos Telek MTA-BME Information Systems Research Group, Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary

Daniele Tessera Dipartimento di Matematica e Fisica, Università Cattolica del Sacro Cuore, Brescia, Italy

Stephen Thompson Department of Computer Science and Engineering, 371 Fairfield Way, Unit 4155, University of Connecticut, Storrs, CT, USA

Jonas Wäfler Department of Telematics, Norwegian University of Science and Technology, Trondheim, Norway

Liudong Xing University of Massachusetts Dartmouth, Dartmouth, MA, USA

Jose Yallouz Department of Electrical Engineering, Israel Institute of Technology (Technion), Haifa, Israel

Shigeru Yamada Department of Social Management Engineering, Tottori University, Tottori-shi, Japan

Shigang Zhang Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA

Xufeng Zhao Department of Mechanical and Industrial Engineering, Qatar University, Doha, Qatar

Review of Prof. Trivedi's Contributions to the Field

Kishor S. Trivedi holds the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He has been on the Duke faculty since 1975.

Professor Trivedi is a leading international expert in the domain of reliability and performability evaluation of fault-tolerant systems. He has made seminal contributions to stochastic modeling formalisms and their efficient solution. He has written influential textbooks that contain not only tutorial material but also the latest advances. He has encapsulated the algorithms developed into usable and well-circulated software packages. He has made key contributions by applying the research results to practical problems, working directly with industry and has been able to not only solve difficult real-world problems but also develop new research results based on these problems.

Professor Trivedi is the author of the well-known text, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, published by Prentice-Hall, which has allowed hundreds of students, researchers, and practitioners to learn analytical modeling techniques. A thoroughly revised second edition (including its Indian edition) of this book was published by John Wiley in 2001. A comprehensive solution manual for the second edition containing more than 300 problem solutions is available from the publisher. This book is the first of its kind to present a balanced treatment of reliability and performance evaluation, while introducing the basic concepts of stochastic models. This book has made a unique contribution to the field of reliability and performance evaluation.

Professor Trivedi has also authored two books: *Performance and Reliability Analysis of Computer Systems*, published by Kluwer Academic Publishers in 1995, and *Queueing Networks and Markov Chains*, published by John Wiley in 1998, which are also well known and focused on performance and reliability evaluation techniques and methodologies. His second book has already made inroads for self-study with practicing engineers and has been adopted by several universities. The third book is considered to be an important book in queueing theory with several interesting example applications. The second edition of this book was published in 2006. He has also edited two books, *Advanced Computer System*

Design, published by Gordon and Breach Science Publishers, and *Performability Modeling Tools and Techniques*, published by John Wiley & Sons.

He is a Fellow of the Institute of Electrical and Electronics Engineers and a Golden Core Member of the IEEE Computer Society. He has published over 500 articles and has supervised 44 Ph.D. dissertations and 30 postdoctoral associates. He is on the editorial boards of *IEEE Transactions on Dependable and Secure Computing*, *Journal of Risk and Reliability*, *International Journal of Performability Engineering*, and *International Journal of Quality, Reliability and Safety Engineering*.

Professor Trivedi has taken significant steps to implement his modeling techniques in tools, which is necessary to the transition from state-of-the-art research to best practices for industry. These tools are used extensively by practicing engineers, researchers, and instructors. Tools include HARP (Hybrid Automated Reliability Predictor) which has been installed at nearly 120 sites; SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) is used at over 550 sites; and SPNP (Stochastic Petri Net Package) has been installed at over 400 sites. Trivedi also helped design and develop IBM's SAVE (System Availability Estimator), Software Productivity Consortium's DAT (Dynamic Assessment Toolset), Boeing's IRAP (Integrated Reliability Analysis Package), and SoHar's SDDS. Graphical user interfaces for these tools have been developed. These packages have been widely circulated and represent a reference point for all researchers in the field of performance evaluation. Most researchers in the field are probably aware of such tools and have very likely used one or more of them.

Trivedi has helped several companies carry out reliability and availability prediction for existing products as well as the ones undergoing design. A partial list of companies for which he has provided consulting services includes 3Com, Avaya, Boeing, DEC, EMC, GE, HP, IBM, Lucent, NEC, TCS, and Wipro. Notable among these is his help to model the reliability of the current return network subsystem of the Boeing 787 for FAA certification. The algorithm he developed for this problem has been jointly patented by Boeing. He led the reliability and availability model of SIP on IBM Websphere, a model that facilitated its sale of the system to a telco giant.

Kishor has developed polynomial time algorithms for performability analysis, numerical solution techniques for completion time problems, algorithms for the numerical solution of the response time distribution in a closed queueing network, techniques to solve large and stiff Markov chains, and algorithms for the automated generation and solution of stochastic reward nets, including sensitivity and transient analysis. His contributions to numerical solution and decomposition techniques for large and stiff stochastic models have considerably relaxed the limits on the size and stiffness of the problems that could be solved by alternative contemporary techniques. He has also developed fast algorithms for the solution of large fault trees and reliability graphs, including multistate components and phase mission systems analysis. He has defined several new paradigms of stochastic Petri nets (Markov regenerative stochastic Petri nets and fluid stochastic Petri nets), which

further extend the power of performance and reliability modeling into non-Markovian and hybrid (mixed, continuous-state, and discrete-state) domains.

His recent work on architecture-based software reliability is very well cited. He and his group have pioneered the areas of software aging and rejuvenation. His group has not only made many theoretical contributions to the understanding of software aging and rejuvenation scheduling but has also been the first to collect data and develop algorithms for the prediction of time to resource exhaustion, leading to adaptive and on-line control of rejuvenation. His methods of software rejuvenation have been implemented in the IBM X-series servers three years after the research was first done; a record time in technology transfer. In recognition of his pioneering work on the aging and rejuvenation of software, Kishor was awarded the title of Doctor Honoris Causa from the Universidad de San Martín de Porres, Lima Peru' on August 12, 2012. He has now moved on to experimental research into software reliability during operation where he is studying affordable software fault tolerance through environmental diversity.

Professor Trivedi developed solution methods for Markov regenerative processes and used them for performance and reliability analysis. He has applied his modeling techniques to a variety of real-world applications, including performance analysis of polling systems and client-server systems, wireless handoff, connection admission control in CDMA systems, reliability analysis of RAID and FDDI token rings, availability analysis of Vaxcluster systems, transient performance analysis of leaky bucket rate control scheme, and the analysis of real-time systems.

The contributions of Trivedi's work are fundamental to the design of reliable digital systems. His papers on modeling fault coverage are recognized as milestones. His papers on hierarchical modeling and fixed-point iteration represent breakthroughs to solve large reliability models. His advances in transient analysis are recognized as leading the state of the art by computer scientists. His contributions to the field of stochastic Petri nets and his work on performability modeling appear as benchmark references on these topics.

Among the enormous scientific production of Prof. Trivedi, the following five papers are representative of his contribution to the scientific community:

- [1] Gianfranco Ciardo and Kishor S. Trivedi. "A decomposition approach for stochastic reward net models." *Performance Evaluation* 18(1) (1993): 37–59. Develops a decomposition approach to large Markovian Petri nets; both theoretical and practical aspects are considered.
- [2] Hoon Choi, Vidyadhar G. Kulkarni, and Kishor S. Trivedi. "Markov regenerative stochastic Petri nets." *Performance Evaluation* 20(1) (1994): 337–357. Develops a new formalism of MRSPN that allows generally distributed firing times concurrently with exponentially distributed firing times.
- [3] Vittorio Castelli, Richard E. Harper, Philip Heidelberger, Steven W. Hunter, Kishor S. Trivedi, Kalyanaraman Vaidyanathan, and William P. Zeggert. "Proactive management of software aging." *IBM Journal of Research and Development* 45(2) (2001): 311–332. Details the implementation of software

rejuvenation in IBM X-series as an example of tech transfer from academia to industry.

- [4] Sachin Garg, Aad van Moorsel, Kalyanaraman Vaidyanathan, Kishor S. Trivedi. "A methodology for detection and estimation of software aging." *Proc. Ninth International Symposium on Software Reliability Engineering* (1998): 283–292. First experiment to demonstrate the existence of software aging and prediction of time to resource exhaustion.
- [5] Andrew Reibman, and Kishor S. Trivedi. "Numerical transient analysis of Markov models." *Computers & Operations Research* 15(1) (1988): 19–36. Benchmark paper on numerical transient analysis of Markov models.

Part I
**Phase Type Distributions, Expectation
Maximization Algorithms, and
Probabilistic Graphical Models**

Phase Type and Matrix Exponential Distributions in Stochastic Modeling

Andras Horvath, Marco Scarpa and Miklos Telek

Abstract Since their introduction, properties of Phase Type (PH) distributions have been analyzed and many interesting theoretical results found. Thanks to these results, PH distributions have been profitably used in many modeling contexts where non-exponentially distributed behavior is present. Matrix Exponential (ME) distributions are distributions whose matrix representation is structurally similar to that of PH distributions but represent a larger class. For this reason, ME distributions can be usefully employed in modeling contexts in place of PH distributions using the same computational techniques and similar algorithms, giving rise to new opportunities. They are able to represent different dynamics, e.g., faster dynamics, or the same dynamics but at lower computational cost. In this chapter, we deal with the characteristics of PH and ME distributions, and their use in stochastic analysis of complex systems. Moreover, the techniques used in the analysis to take advantage of them are revised.

1 Introduction

Stochastic modeling has been used for performance analysis and optimization of computer systems for more than five decades [19]. The main analysis method behind this effort was the continuous time Markov chains (CTMC) description of the sys-

A. Horvath (✉)

Dipartimento di Informatica, Università degli Studi di Torino, C.so Svizzera 185,
10149 Turin, Italy
e-mail: horvath@di.unito.it

M. Scarpa

Dipartimento di Ingegneria, Università degli Studi di Messina, C.da di Dio,
98166 Messina, Italy
e-mail: mscarpa@unime.it

M. Telek

Department of Networked Systems and Services, MTA-BME Information
Systems Research Group, Budapest University of Technology and Economics,
Magyar Tudosok krt. 2, Budapest 1117, Hungary
e-mail: telek@hit.bme.hu

© Springer International Publishing Switzerland 2016

L. Fiondella and A. Puliafito (eds.), *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering,
DOI 10.1007/978-3-319-30599-8_1

tem behavior and the CTMC-based analysis of the performance measures of interest. With the evolution of computing devices, model description languages (e.g., queueing systems, Petri nets, process algebras), and model analysis techniques (a wide range of software tools with efficient analysis algorithm using adequate data representation and memory management) the analysis of more and more complex systems has become possible. One of main modeling limitations of the CTMC-based approach is the limitation on the distribution of the random time durations, which is restricted to be exponentially distributed. Unfortunately, in a wide range of practical applications, the empirical distribution of field data differs significantly from the exponential distribution. The effort to relax this restriction of the CTMC-based modeling on exponentially distributed durations resulted in the development of many alternative stochastic modeling methodologies (semi-Markov and Markov regenerative processes [11], analysis with the use of continuous system parameters [8]), yet all of the alternative modeling methodologies quickly suffers from infeasible computational complexity when the complexity of the system considered increases beyond basic examples.

It remains a significant research challenge to relax the modeling restriction of the exponentially distributed duration time and still evaluate complex model behaviors. To this end, one of the most promising approaches is the extension of CTMC-based analysis to non-exponentially distributed durations. Initial steps in this direction date back to the activity of A.K. Erlang in the first decades of the twentieth century as reported in [10]. These initial trials were referred to as the method of phases, which influenced later terminology. M.F. Neuts characterized a set of distributions which can be incorporated into CTMC-based analysis by introducing the set of phase type (PH) distributions [16].

The extension of CTMC-based analysis (where the durations are exponentially distributed) with PH distributed durations requires the generation of a large CTMC, referred to as extended Markov chain (EMC), which combines the system behavior with the description of the PH distributions. In this chapter, we summarize the basics of EMC-based stochastic analysis and provide some application examples. Finally, we note that in this chapter we restrict our attention to continuous time stochastic models, but that the same approach applies for discrete time stochastic models as well.

1.1 Structure of the Chapter

The next two sections, Sects. 2 and 3, summarize the basic information on PH and ME distributions, respectively. The following two sections, Sects. 4 and 5, discuss the analysis procedure for complex stochastic systems with PH and ME distributed durations, respectively. The tools available to support EMC-based analysis of stochastic systems is presented in Sect. 6. Numerical examples demonstrate the modeling and analysis capabilities of the approach are discussed in Sect. 7 and the main findings and conclusions are given in Sect. 8.

2 PH Distributions and Their Basic Properties

2.1 Assumed Knowledge

Transient behavior of a finite state Markov chain with generator \mathbf{Q} and initial distribution π , specifically, the transient probability vector $p(t)$, satisfies the ordinary differential equation

$$\frac{d}{dt} p(t) = p(t)\mathbf{Q}, \text{ with initial condition } p(0) = \pi,$$

whose solution is a matrix exponential function

$$p(t) = \pi e^{\mathbf{Q}t}, \quad (1)$$

where the matrix exponential term is defined as

$$e^{\mathbf{Q}t} = \sum_{i=0}^{\infty} \frac{t^i}{i!} \mathbf{Q}^i.$$

The properties of generator \mathbf{Q} and initial distribution π are as follows. The elements of π are probabilities, i.e., nonnegative numbers not greater than one. The off-diagonal elements of \mathbf{Q} are transition intensities, i.e., nonnegative numbers. The diagonal elements of \mathbf{Q} are such that each row sum is zero, i.e., the diagonal elements are non-positive. The elements of π sum to one, that is $\sum_i \pi_i = \pi \mathbf{1} = 1$. Each row of a generator matrix sums to zero, that is $\sum_j Q_{ij} = 0$, or equivalently, in vector form, we can write $\mathbf{Q}\mathbf{1} = \mathbf{0}$, where $\mathbf{1}$ is a column vector of ones and $\mathbf{0}$ is a column vector of zeros. Hereafter, the sizes of vector $\mathbf{1}$ and $\mathbf{0}$ are defined by the context such that the dimensions in the vector expressions are compatible.

The stationary distribution of an irreducible finite state Markov chain with generator \mathbf{Q} , $p \triangleq \lim_{t \rightarrow \infty} p(t)$, can be computed as the unique solution of the linear system of equations

$$p\mathbf{Q} = \mathbf{0}, \quad p\mathbf{1} = 1. \quad (2)$$

In this chapter, we focus on the computation of the initial distribution and the generator matrix of the EMC and do not discuss the efficient solution methods for solving (1) and (2).

2.2 Phase Type Distributions

PH distributions are defined by the behavior of a Markov chain, which is often referred to as the background Markov chain behind a PH.

Let $X(t)$ be a Markov chain with n transient and one absorbing states, meaning that the absorbing state is reachable (by a series of state transitions) from all transient states, but when the Markov chain moves to the absorbing state it remains there forever. Let π be the initial distribution of the Markov chain, that is $\pi_i = P(X(0) = i)$. Without loss of generality, we number the states of the Markov chain such that state $1, \dots, n$ are transient states and state $n + 1$ is the absorbing state. The generator matrix of such a Markov chain has the following structure

$$\mathbf{Q} = \begin{bmatrix} \mathbf{A} & \mathbf{a} \\ \mathbf{0} & 0 \end{bmatrix},$$

where \mathbf{A} is a square matrix of size n and \mathbf{a} is a column vector of size n . Since the rows of the generator matrix sum to zero, the elements of \mathbf{a} can be computed from \mathbf{A} , that is $\mathbf{a} = -\mathbf{A}\mathbf{1}$. Similarly, the first n elements of the initial vector π , denoted by $\boldsymbol{\alpha}$, completely defines the initial vector, since the $(n + 1)$ st element of π is $1 - \boldsymbol{\alpha}\mathbf{1}$. We note that $\boldsymbol{\alpha}$ defines the initial probabilities of the transient states. With the help of this Markov chain, we are ready to define PH distributions.

Definition 1 The time to reach the absorbing state of a Markov chain with a finite number of transient and an absorbing state

$$T = \min\{t : X(t) = n + 1, t \geq 0\},$$

is phase type distributed.

Throughout this document, we assume that the Markov chain starts from one of the transient states and consequently $\boldsymbol{\alpha}\mathbf{1} = 1$, i.e., there is no probability mass at zero and T has a continuous distribution on \mathbb{R}^+ . Since the time to reach the absorbing state is a transient measure of the Markov chain, we can evaluate the distribution of random variable T , based on the transient analysis of the Markov chain with initial distribution π and and generator matrix \mathbf{Q}

$$F_T(t) = P(T < t) = P(X(t) = n + 1) = \pi e^{\mathbf{Q}t} e_{n+1},$$

where e_{n+1} is the $(n + 1)$ st unit vector (the column vector with zero elements except in position $n + 1$ which is one).

This straight forward description of the distribution of T is not widely used due to the redundancy of matrix \mathbf{Q} and vector π . Indeed, matrix \mathbf{A} and the initial vector associated with the transient states, $\boldsymbol{\alpha}$, define all information about the distribution of T and the analytical description based on $\boldsymbol{\alpha}$ and \mathbf{A} is much simpler to use in more complex stochastic models. To obtain the distribution based on $\boldsymbol{\alpha}$ and \mathbf{A} , we carry on the block structure of matrix \mathbf{Q} in the computation.

$$\begin{aligned}
F_T(t) &= P(T < t) = P(X(t) = n + 1) = 1 - \sum_{i=1}^n P(X(t) = n + 1) \\
&= 1 - [\alpha, 0] e^{Q^t} \begin{bmatrix} \mathbf{1} \\ 0 \end{bmatrix} = 1 - [\alpha, 0] \sum_{i=0}^{\infty} \frac{t^i}{i!} \begin{bmatrix} \mathbf{A} & \mathbf{a} \\ \mathbf{0} & 0 \end{bmatrix}^i \begin{bmatrix} \mathbf{1} \\ 0 \end{bmatrix} \\
&= 1 - [\alpha, 0] \sum_{i=0}^{\infty} \frac{t^i}{i!} \begin{bmatrix} \mathbf{A}^i & \bullet \\ \mathbf{0} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ 0 \end{bmatrix} = 1 - \alpha \sum_{i=0}^{\infty} \frac{t^i}{i!} \mathbf{A}^i \mathbf{1} = 1 - \alpha e^{A^t} \mathbf{1},
\end{aligned}$$

where \bullet indicates irrelevant matrix block whose elements are multiplied by zero. The PDF of T can be obtained from the derivative of its CDF.

$$\begin{aligned}
f_T(t) &= \frac{d}{dt} F_T(t) = \frac{d}{dt} \left(1 - \alpha \sum_{i=0}^{\infty} \frac{t^i}{i!} \mathbf{A}^i \mathbf{1} \right) = -\alpha \sum_{i=0}^{\infty} \frac{d}{dt} \frac{t^i}{i!} \mathbf{A}^i \mathbf{1} \\
&= -\alpha \sum_{i=1}^{\infty} \frac{t^{i-1}}{(i-1)!} \mathbf{A}^{i-1} \mathbf{A} \mathbf{1} = -\alpha e^{A^t} \mathbf{A} \mathbf{1} = \alpha e^{A^t} \mathbf{a},
\end{aligned}$$

where we used $\mathbf{a} = -\mathbf{A} \mathbf{1}$ in the last step.

Before computing the remaining properties of PH distributions we need to classify the eigenvalues of \mathbf{A} . The i, j element of matrix e^{A^t} contains the probability that starting from transient state i the Markov chain is in transient state j at time t . If states $1, \dots, n$ are transient states then as t tends to infinity e^{A^t} tends to zero, which means that the eigenvalues of \mathbf{A} have negative real part and, as a consequence, \mathbf{A} is non-singular.

The Laplace transform of T , $E(e^{-sT})$, can be computed as

$$\begin{aligned}
f_T^*(s) &= E(e^{-sT}) = \int_{t=0}^{\infty} e^{-st} f_T(t) dt = \int_{t=0}^{\infty} e^{-st} \alpha e^{A^t} \mathbf{a} dt \\
&= \alpha \int_{t=0}^{\infty} e^{(-s\mathbf{I} + \mathbf{A})t} dt \mathbf{a} = \alpha (s\mathbf{I} - \mathbf{A})^{-1} \mathbf{a},
\end{aligned}$$

where we note that the integral surely converges for $\mathcal{R}(s) \geq 0$ because in this case the eigenvalues of $-s\mathbf{I} + \mathbf{A}$ also possess a negative real part.

To compute the k th moment of T , $E(T^k)$, we need the following integral relation

$$\left[t^k e^{A^t} \right]_0^{\infty} = \int_{t=0}^{\infty} k t^{k-1} e^{A^t} dt + \int_{t=0}^{\infty} t^k e^{A^t} A dt,$$

whose left-hand side is zero because the eigenvalues of \mathbf{A} possess a negative real part. Multiplying both side with $(-\mathbf{A})^{-1}$ we get

$$\int_{t=0}^{\infty} t^k e^{At} dt = k \int_{t=0}^{\infty} t^{k-1} e^{At} dt (-A)^{-1}.$$

Using this relation, the k th moment of T is

$$\begin{aligned} E(T^k) &= \int_{t=0}^{\infty} t^k f_T(t) dt = \alpha \int_{t=0}^{\infty} t^k e^{At} dt (-A) \mathbf{1} = k\alpha \int_{t=0}^{\infty} t^{k-1} e^{At} dt \mathbf{1} \\ &= k(k-1)\alpha \int_{t=0}^{\infty} t^{k-2} e^{At} dt (-A)^{-1} \mathbf{1} = \dots = k! \alpha (-A)^{-k} \mathbf{1}. \end{aligned}$$

These four properties of PH distributions (CDF, PDF, Laplace transform, and moments) have several interesting consequences and some of which we summarize below.

- Matrix $(-A)^{-1}$ has an important stochastic meaning. Let T_{ij} be the time spent in transient state j before moving to the absorbing state when the Markov chain starts from state i . For $E(T_{ij})$, we have

$$E(T_{ij}) = \frac{\delta_{ij}}{-A_{ii}} + \sum_{k, k \neq i} \frac{A_{ik}}{-A_{ii}} E(T_{kj}),$$

where δ_{ij} is the Kronecker delta symbol. The first term of the left-hand side is the time spent in state j while the Markov chain is in the initial state, and the second term is the time spent in state j during later visits to j . Multiplying both sides by $-A_{ii}$ and adding $E(T_{ij}) A_{ii}$ gives

$$0 = \delta_{ij} + \sum_k A_{ik} E(T_{kj}),$$

whose matrix form is

$$\mathbf{0} = \mathbf{I} + A\bar{\mathbf{T}} \quad \longrightarrow \quad \bar{\mathbf{T}} = (-A)^{-1},$$

where $\bar{\mathbf{T}}$ is the matrix composed of the elements $E(T_{ij})$. Consequently, the (ij) element of $(-A)^{-1}$ is $E(T_{ij})$, which is a nonnegative number.

- $f_T^*(s)$ is a rational function of s whose numerator is at most order $n-1$ and denominator is at most order n . This is because

$$\begin{aligned} f_T^*(s) &= \alpha (s\mathbf{I} - A)^{-1} \mathbf{a} = \sum_i \sum_j \alpha_i (s\mathbf{I} - A)^{-1}_{ij} \mathbf{a}_j \\ &= \sum_i \sum_j \alpha_i \left[\frac{\det_{ji}(s\mathbf{I} - A)}{\det(s\mathbf{I} - A)} \right] \mathbf{a}_j = \frac{\sum_i \sum_j \alpha_i \mathbf{a}_j \det_{ji}(s\mathbf{I} - A)}{\det(s\mathbf{I} - A)}. \end{aligned}$$

$\det_{ji}(\mathbf{M})$ denotes the determinant of the matrix obtained by removing row j and column i of matrix \mathbf{M} . The denominator of the last expression is an order n polynomial of s , while the numerator is the sum of order $n - 1$ polynomials, which is at most an order $n - 1$ polynomial of s .

- This rational Laplace transform representation indicates that a PH distribution with n transient state can be represented by $2n - 1$ independent parameters. A polynomial of order n is defined by $n + 1$ coefficients, and a rational function of order $n - 1$ numerator, and order n denominator is defined by $2n + 1$ parameters. Normalizing the denominator such that the coefficient of s^n is 1 and considering that $\int_0^\infty f_T(t)dt = \lim_{s \rightarrow 0} f_T^*(s) = 1$ adds two constraints for the coefficients, from which the number of independent parameters is $2n - 1$.
- The PDF of a PH distribution is the sum of exponential functions. Let $\mathbf{A} = \mathbf{B}^{-1} \Delta \mathbf{B}$ be the Jordan decomposition¹ of \mathbf{A} and let $u = \alpha \mathbf{B}^{-1}$ and $v = \mathbf{B} \mathbf{a}$. Then,

$$f_T(t) = \alpha e^{\mathbf{A}t} \mathbf{a} = \alpha \mathbf{B}^{-1} e^{\Delta t} \mathbf{B} \mathbf{a} = u e^{\Delta t} v.$$

At this point, we distinguish two cases.

- The eigenvalues of \mathbf{A} are different and Δ is a diagonal matrix. In this case, $f_T(t)$ is a sum of exponential functions because

$$f_T(t) = u e^{\Delta t} v = \sum_i u_i v_i e^{\lambda_i t} = \sum_i c_i e^{\lambda_i t},$$

where $c_i = u_i v_i$ is a constant coefficient of the exponential function.

Here the eigenvalues (λ_i) as well as the associated coefficients (c_i) can be real or complex conjugate pairs. For a complex conjugate pair of eigenvalues, we have

$$c_i e^{\lambda_i t} + \bar{c}_i e^{\bar{\lambda}_i t} = 2|c_i| e^{\mathcal{R}(\lambda_i)t} \cos(\mathcal{I}(\lambda_i)t - \varphi_i),$$

where $c_i = |c_i| e^{i\varphi_i}$, $\mathcal{R}(\lambda_i)$ and $\mathcal{I}(\lambda_i)$ are the real and the imaginary part of λ_i and i is the imaginary unit.

- There are eigenvalues of \mathbf{A} with higher multiplicity and Δ contains real Jordan blocks. The matrix exponent of a Jordan block is

$$\exp \left[\left(\begin{pmatrix} \lambda & 1 & & \\ & \lambda & 1 & \\ & & \ddots & \ddots \\ & & & \lambda \end{pmatrix} t \right) \right] = \begin{pmatrix} e^{\lambda t} & t e^{\lambda t} & \frac{1}{2!} t^2 e^{\lambda t} & \frac{1}{3!} t^3 e^{\lambda t} \\ & e^{\lambda t} & t e^{\lambda t} & \frac{1}{2!} t^2 e^{\lambda t} \\ & & \ddots & \ddots \\ & & & e^{\lambda t} \end{pmatrix}.$$

¹The case of different Jordan blocks with identical eigenvalue is not considered here, because it cannot occur in non-redundant PH representations.

Consequently, the density function takes the form

$$f_T(t) = \sum_{i=1}^{\#\lambda} \sum_{j=1}^{\#\lambda_i} c_{ij} t^{j-1} e^{\lambda_i t},$$

where $\#\lambda$ is the number of different eigenvalues and $\#\lambda_i$ is the multiplicity of λ_i . Similar to the previous case, the eigenvalues (λ_i) as well as the associated coefficients ($c_{i,j}$) can be real or complex conjugate pairs. For a complex conjugate pair of eigenvalues, we have

$$c_{i,j} t^{j-1} e^{\lambda_i t} + \bar{c}_{i,j} t^{j-1} e^{\bar{\lambda}_i t} = 2|c_{i,j}| t^{j-1} e^{\mathcal{R}(\lambda_i)t} \cos(\mathcal{I}(\lambda_i)t - \varphi_{i,j}),$$

where $c_{i,j} = |c_{i,j}| e^{i\varphi_{i,j}}$.

As a result of all of these cases, the density function of a PH distribution possesses the form

$$f_T(t) = \sum_{i=1}^{\#\lambda_R} \sum_{j=1}^{\#\lambda_i^R} c_{ij} t^{j-1} e^{\lambda_i^R t} + \sum_{i=1}^{\#\lambda_C} \sum_{j=1}^{\#\lambda_i^C} 2|c_{i,j}| t^{j-1} e^{\mathcal{R}(\lambda_i^C)t} \cos(\mathcal{I}(\lambda_i^C)t - \varphi_{i,j}) \quad (3)$$

where $\#\lambda_R$ is the number of different real eigenvalues and $\#\lambda_C$ is the number of different complex conjugate eigenvalue pairs.

- In general, infinitely many Markov chains can represent the same PH distribution.
 - The following *similarity transformation* generates representations with identical size.

Let \mathbf{T} be a non-singular matrix with unit row sums ($\mathbf{T}\mathbf{1} = \mathbf{1}$). The vector–matrix pairs $(\boldsymbol{\alpha}, \mathbf{A})$ and $(\boldsymbol{\alpha}\mathbf{T}, \mathbf{T}^{-1}\mathbf{A}\mathbf{T})$ are two different vector–matrix representations of the same PH distribution, since

$$F_T(t) = 1 - \boldsymbol{\alpha}\mathbf{T}e^{\mathbf{T}^{-1}\mathbf{A}\mathbf{T}t}\mathbf{1} = 1 - \boldsymbol{\alpha}\mathbf{T}\mathbf{T}^{-1}e^{\mathbf{A}t}\mathbf{T}\mathbf{1} = 1 - \boldsymbol{\alpha}e^{\mathbf{A}t}\mathbf{1}.$$

- Representations with different sizes can be obtained as follows.

Let matrix \mathbf{V} of size $m \times n$ be such that $\mathbf{V}\mathbf{1} = \mathbf{1}$.

The vector–matrix pairs $(\boldsymbol{\alpha}, \mathbf{A})$ of size n and $(\boldsymbol{\gamma}, \mathbf{G})$ of size m are two different vector–matrix representations of the same PH distribution if $\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{G}$ and $\boldsymbol{\alpha}\mathbf{V} = \boldsymbol{\gamma}$ because

$$F_T(t) = 1 - \boldsymbol{\gamma}e^{\mathbf{G}t}\mathbf{1} = 1 - \boldsymbol{\alpha}\mathbf{V}e^{\mathbf{G}t}\mathbf{1} = 1 - \boldsymbol{\alpha}e^{\mathbf{A}t}\mathbf{V}\mathbf{1} = 1 - \boldsymbol{\alpha}e^{\mathbf{A}t}\mathbf{1}$$

in this case.

3 Matrix Exponential Distributions and Their Basic Properties

In the definition of PH distributions, vector α is a probability vector with nonnegative elements and matrix A is a generator matrix with negative diagonal and nonnegative off-diagonal elements. Relaxing these sign constraints for the vector and matrix elements and maintaining the matrix exponential distribution (and density) function results in the set of matrix exponential (ME) distributions.

Definition 2 Random variable T with distribution function

$$F_T(t) = 1 - \alpha e^{At} \mathbf{1},$$

where α is a finite real vector and A is a finite real matrix, is matrix exponentially distributed.

The size of α and A plays the same role as the number of transient states in case of PH distributions. By definition, the set of PH distributions with a given size is a subset of the set of PH distributions with the same size.

ME distributions share the following basic properties with PH distributions: matrix exponential distribution function, matrix exponential density function, moments, rational Laplace transform, the same set of functions as in (3), and non-unique representation. The main difference between the matrix exponential and the PH classes comes from the fact that the sign constraints on the elements of generator matrixes restrict the eigenvalue structure of such matrixes, while such restrictions do not apply in case of ME distributions. For example, the eigenvalues of an order three PH distribution with dominant eigenvalue θ satisfy $\mathcal{R}(\lambda_i) \leq \theta$ and $|\mathcal{I}(\lambda_i)| \leq (\theta - \mathcal{R}(\lambda_i))/\sqrt{3}$, while the eigenvalues of an order three ME distribution with dominant eigenvalue θ satisfy $\mathcal{R}(\lambda_i) \leq \theta$ only. This flexibility of the eigenvalues has significant consequence on the flexibility of the set of order three PH and ME distributions. For example, the minimal squared coefficient of variation among the order three PH and ME distributions are 1/3 and 0.200902, respectively.

The main difficulty encountered when working with ME distributions is that a general vector–matrix pair does not always define a nonnegative density function, while a vector–matrix pair with the sign constraints of PH distributions does. Efficient numerical methods have been proposed recently to check the nonnegativity of a matrix exponential function defined by a general vector–matrix pair, but general symbolic conditions are still missing.

4 Analysis of Models with PH Distributed Durations

If all durations (service times, interarrival times, repair times, etc.) in a system are distributed according to PH distributions, then its overall behavior can be captured by a continuous time Markov chain, referred to as extended Markov chain (EMC).

In this section, we show how to derive the infinitesimal generator of this EMC using Kronecker operations. The methodology here described has been originally presented in the case of Discrete PHs in [17] and more recently in the case of Continuous PHs in [13]

To this end we first introduce the notation used to describe the model. By \mathcal{S} , we denote the set of states and by $N = |\mathcal{S}|$ the number of states. The states themselves are denoted by s_1, s_2, \dots, s_N . The set of activities is denoted by \mathcal{A} and the set of those that are active in state s_i is denoted by \mathcal{A}_i . The activities are denoted by a_1, a_2, \dots, a_M with $M = |\mathcal{A}|$. When activity a_i is completed in state s_j then the system moves from state s_j to state $n(j, i)$, i.e., n is the function that provides the next state. We assume that the next state is a deterministic function of the current state and the activity that completes. We further assume that there does not exist a triple, k, i, j , for which $s_k \in \mathcal{S}$, $a_i \in \mathcal{A}$, $a_j \in \mathcal{A}$ and $n(k, i) = n(k, j)$. These two assumptions, which make the formulas simpler, are easy to relax in practice. There can be activities that end when the system moves from state s_i to state s_j even if they do not complete and are active both in s_i and in s_j . These activities are collected in the set $e(i, j)$. The PH distribution that is associated with activity a_i is characterized by the initial vector α_i and matrix A_i . As before, we use the notation $\mathbf{a}_i = -A_i \mathbf{1}$ to refer to the vector containing the intensities that lead to completion of activity a_i . The number of phases of the PH distribution associated with activity a_i is denoted by n_i .

Example 1 PH/PH/1/K queue with server break-downs. As an example, we consider, using the above-described notation, a queue in which the server is subject to failure only if the queue is not empty. The set of states is $\mathcal{S} = \{s_1, s_2, \dots, s_{2K+1}\}$ where s_1 represents the empty queue, s_{2i} with $1 \leq i \leq K$ represents the state with i clients in the queue and the server up, and s_{2i+1} with $1 \leq i \leq K$ represents the state with i clients and the server down. There are four activities in the system: a_1 represents the arrival activity, a_2 the service activity, a_3 the failure activity,² and a_4 the repair activity. The vectors and matrices that describe the associated PH distributions are $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and A_1, A_2, A_3, A_4 . In this example, we assume that the arrival activity is active if the system is not full and it is inactive if the system is full. The service and the failure activities are active if the queue is not empty and the server is up. The repair activity is active if the queue is not empty and the server is down. Accordingly, we have $\mathcal{A}_1 = \{a_1\}$, $\mathcal{A}_{2i} = \{a_1, a_2, a_3\}$ for $1 \leq i \leq K - 1$, $\mathcal{A}_{2i+1} = \{a_1, a_4\}$ for $1 \leq i \leq K - 1$, $\mathcal{A}_{2K} = \{a_2, a_3\}$, and $\mathcal{A}_{2K+1} = \{a_4\}$. The next state function is as follows: for arrivals we have $n(1, 1) = s_2$ and $n(i, 1) = s_{i+2}$ with $2 \leq i \leq 2K - 1$; for services $n(2, 2) = s_1$ and $n(2i, 2) = s_{2i-2}$ with $2 \leq i \leq K$; for failures $n(2i, 3) = s_{2i+1}$ with $1 \leq i \leq K$; for repairs $n(2i + 1, 4) = s_{2i}$ with $1 \leq i \leq K$. We assume that the failure activity ends every time when a service activity completes, i.e., failure is connected to single jobs and not to the aging of the server. Other activities end only when they complete or when such a state is reached in which they are not active. Accordingly, $e(2i, 2i - 2) = \{a_3\}$ for $2 \leq i \leq K$.

²Failure is more like an event than an activity but, in order to keep the discussion clearer, we refer to it as failure activity.

Based on the description of the ingredients of the model, it is possible to derive blocks of the initial probability vector and the blocks of the infinitesimal generator of the corresponding CTMC. Let us start with the infinitesimal generator, which we denote by \mathbf{Q} , composed of $N \times N$ blocks. The block of \mathbf{Q} that is situated in the i th row of blocks and in the j th column of blocks is denoted by \mathbf{Q}_{ij} . A block in the diagonal, \mathbf{Q}_{ii} describes the parallel execution of the activities that are active in s_i . The parallel execution of CTMCs can be captured by the Kronecker-sum operator (\oplus), and thus we have

$$\mathbf{Q}_{ii} = \bigoplus_{j:s_j \in \mathcal{A}_i} \mathbf{A}_j .$$

An off-diagonal block, \mathbf{Q}_{ij} , is not a zero matrix only if there exists an activity whose completion moves the system from state s_i to state s_j . Let us assume that the completion of activity a_k moves the system from state s_i to state s_j , i.e., $n(i, k) = s_j$. The corresponding block, \mathbf{Q}_{ij} , must

- reflect the fact that activity a_k completes and restarts if a_k is active in s_j ,
- reflect the fact that activity a_k completes and does not restart if a_k is not active in s_j ,
- end activities that are active in s_i but not in s_j ,
- start those activities that are not active in s_i but are active in s_j ,
- end and restart those activities that are active both in s_i and in s_j but are in $e(i, j)$,
- and maintain the phase of those that are active both in s_i and in s_j and are not in $e(i, j)$.

The joint treatment of the above cases can be carried out by the Kronecker-product operator and thus we have

$$\mathbf{Q}_{ij} = \bigotimes_{l:1 \leq l \leq M} \mathbf{R}_l$$

with

$$\mathbf{R}_l = \begin{cases} \mathbf{a}_k & \text{if } l = k \text{ and } k \notin \mathcal{A}_j \\ \mathbf{a}_k \boldsymbol{\alpha}_k & \text{if } l = k \text{ and } k \in \mathcal{A}_j \\ \mathbf{1}_{n_l} & \text{if } l \neq k \text{ and } k \in \mathcal{A}_i \text{ and } k \notin \mathcal{A}_j \\ \boldsymbol{\alpha}_l & \text{if } l \neq k \text{ and } k \notin \mathcal{A}_i \text{ and } k \in \mathcal{A}_j \\ \mathbf{1}_{n_l} \boldsymbol{\alpha}_l & \text{if } l \neq k \text{ and } k \in \mathcal{A}_i \text{ and } k \in \mathcal{A}_j \text{ and } k \in e(i, j) \\ \mathbf{I}_{n_l} & \text{if } l \neq k \text{ and } k \in \mathcal{A}_i \text{ and } k \in \mathcal{A}_j \text{ and } k \notin e(i, j) \\ 1 & \text{otherwise} \end{cases}$$

where the subscripts to $\mathbf{1}$ and \mathbf{I} indicate their size.

The initial probability vector of the CTMC, $\boldsymbol{\pi}$, is a row vector composed of N blocks which must reflect the initial probabilities of the states of the system and the initial probabilities of the PH distributions of the active activities. Denoting by π_i the initial probability of state s_i , the i th block of the initial probability vector, $\boldsymbol{\pi}_i$, is given as

$$\pi_i = \bigotimes_{j:s_j \in \mathcal{A}_i} \alpha_j .$$

Example 2 For the previous example, the diagonal blocks, which must reflect the ongoing activities, are the following:

$$\begin{aligned} \mathcal{Q}_{1,1} &= A_1, \quad \mathcal{Q}_{2i,2i} = A_1 \bigoplus A_2 \bigoplus A_3, \quad \mathcal{Q}_{2i+1,2i+1} = A_1 \bigoplus A_4, \\ \mathcal{Q}_{2K,2K} &= A_2 \bigoplus A_3, \quad \mathcal{Q}_{2K+1,2K+1} = A_4 \quad \text{with } 1 \leq i \leq K-1 \end{aligned}$$

Arrival in state s_1 takes the system to state s_2 . The corresponding block must complete and restart the arrival activity and must restart both the service and the failure activity:

$$\mathcal{Q}_{12} = a_1 \alpha_1 \bigotimes \alpha_2 \bigotimes \alpha_3 \quad (4)$$

Arrival in state s_{2i} (server up) takes the system to state s_{2i+2} . If the system does not become full then the corresponding block must complete and restart the arrival activity and must maintain the phase of both the service and the failure activity. If the system becomes full, the arrival activity is not restarted. Accordingly, we have

$$\begin{aligned} \mathcal{Q}_{2i,2i+2} &= a_1 \alpha_1 \bigotimes \mathbf{I}_{n_2} \bigotimes \mathbf{I}_{n_3} \quad \text{with } 1 \leq i \leq K-2 \\ \mathcal{Q}_{2K-2,2K} &= a_1 \bigotimes \mathbf{I}_{n_2} \bigotimes \mathbf{I}_{n_3} \end{aligned}$$

An arrival in state s_{2i+1} (server down) takes the system to state s_{2i+3} . If the system does not become full then the corresponding block must complete and restart the arrival activity and must maintain the phase of the repair activity. If the system becomes full, the arrival activity is not restarted. Accordingly, we have

$$\begin{aligned} \mathcal{Q}_{2i+1,2i+3} &= a_1 \alpha_1 \bigotimes \mathbf{I}_{n_4} \quad \text{with } 1 \leq i \leq K-2 \\ \mathcal{Q}_{2K-1,2K+1} &= a_1 \bigotimes \mathbf{I}_{n_4} \end{aligned}$$

Service completion can take place in three different situations. If the system becomes empty then the phase of the arrival activity is maintained, the service activity is completed and the failure activity is put to an end. If the system neither becomes empty nor was full then the phase of the arrival activity is maintained, the service activity is completed and restarted, and the failure activity ends and restarts. Finally, if the queue was full then the arrival activity is restarted, the service activity is completed and restarted, and the failure activity is put to an end and restarted. Accordingly, we have

$$\mathcal{Q}_{2,1} = \mathbf{I}_{n_1} \bigotimes a_2 \bigotimes \mathbf{1}_{n_3}$$

$$\begin{aligned}
Q_{2i,2i-1} &= \mathbf{I}_{n_1} \otimes a_2 \alpha_2 \otimes \mathbf{1}_{n_3} \alpha_3 \quad \text{with } 1 < i < K \\
Q_{2K,2K-2} &= \alpha_1 \otimes a_2 \alpha_2 \otimes \mathbf{1}_{n_3} \alpha_3
\end{aligned} \tag{5}$$

The failure activity can be completed in two different situations. If the system is not full, then the phase of the arrival activity is maintained. If the system is full then the arrival activity is not active. In both cases, the service activity ends, the failure activity is completed and the repair activity is initialized.

$$\begin{aligned}
Q_{2i,2i+1} &= \mathbf{I}_{n_1} \otimes \mathbf{1}_{n_2} \otimes a_3 \otimes \alpha_4 \quad \text{with } 1 \leq i < K \\
Q_{2K,2K+1} &= \mathbf{1}_{n_2} \otimes a_3 \otimes \alpha_4
\end{aligned}$$

Similarly to the failure activity, also the repair activity can be completed in two different situations because the arrival activity can be active or inactive. In both cases, the service activity and the failure activity must be initialized and the repair activity completes.

$$\begin{aligned}
Q_{2i+1,2i} &= \mathbf{I}_{n_1} \otimes \alpha_2 \otimes \alpha_3 \otimes a_4 \quad \text{with } 1 \leq i < K \\
Q_{2K+1,2K} &= \alpha_2 \otimes \alpha_3 \otimes a_4
\end{aligned}$$

5 Analysis of Stochastic Systems with ME Distributed Durations

The most important observation to take from this section is that all steps of the method of EMCs (as explained in the previous section) remain directly applicable in case of ME distributed durations (where the (α_i, A_i) vector–matrix pairs describe ME distributions). In that case, the only difference is that the signs of the vector and matrix elements are not restricted to be nonnegative in case of the vector elements and off-diagonal matrix elements and to be negative in case of the diagonal matrix elements. Consequently, the model description does not allow a probabilistic interpretation via Markov chains.

This general conclusion was obtained through serious research efforts. Following the results in [12], it was suspected that in a stochastic model ME distributions could be used in place of PH distributions and several results would carry over, but it was not easy to prove these conjectured results in the general setting because probabilistic arguments associated with PH distributions no longer hold. In [1], it was shown that matrix geometric methods can be applied for quasi-birth–death processes (QBDs) with rational arrival processes (RAPs) [3], which can be viewed as an extension of ME distributions to arrival processes. To prove that the matrix geometric relations hold, the authors of [1] use an interpretation of RAPs proposed in [3]. However, the models considered are limited to QBDs. For the model class of SPNs with ME

distributed firing times, the applicability of the EMC-like analysis was proved in [2] and refined for the special case when the ME distribution has no PH representation in [4].

6 Analysis tools

Based on the common representation of the EMC through the Kronecker algebra, smart algorithms have been developed recently to optimize memory usage. These algorithms build the EMC in a completely symbolic way, both at the process state space level and at the expanded state space level, as deeply explained in [13] that we use as reference.

The algorithm presented in [13] is based on two high level steps:

1. to generate the reachability graph of the model (which collects the system states in a graph according to their reachability from an initial set of states) using a symbolic technique;
2. to enrich the symbolically stored reachability graph with all the necessary information to evaluate Kronecker expressions representing the expanded state space.

Step 1 is performed using symbolic technique based on complex data structures like Multi-Valued Decision Diagram (MDD) [18] to encode the model state space; step 2 adds information related to each event memory policy to the encoded state space. In manner it is possible to use on the fly expressions introduced in Sects. 4 and 5 to compute various probability measures of the model.

6.1 Symbolic Generation of Reachability Graph

Both traditional performance or dependability evaluation techniques and more recent model checking-based approaches are grounded in the knowledge of the set of states that the system considered can reach starting from a particular initial state (or in general from a set of initial states). Symbolic techniques [5] focus on generating a compact representation of huge state spaces by exploiting a model's structure and regularity. A model has a structure when it is composed of K sub-models, for some $K \in \mathbb{N}$. In this case, a global system state can be represented as a K -tuple (q^1, \dots, q^K) , where q^k is the local state of sub-model k (having some finite size n^k).

The use of (MDDs) for the encoding of model state spaces was introduced by Miner and Ciardo in [14]. MDDs are rooted, directed, acyclic graphs associated with a finite ordered set of integer variables. When used to encode a state space, an MDD has the following structure:

- nodes are organized into $K + 1$ levels, where K is the number of sub-models;
- level K contains only a single non-terminal node, the root, whereas levels $K - 1$ through 1 contain one or more non-terminal nodes;
- a non-terminal node at level k has n^k arcs pointing to nodes at level $k - 1$;

A state $s = (q^1, \dots, q^K)$ belongs to S if and only if a path exists from the root node to the terminal node 1 such that, at each node, the arc corresponding to the local state q^k is followed. In [6], and then in [7], Ciardo et al. proposed the *Saturation* algorithm for the generation of reachability graphs using MDDs. Such an iteration strategy improves both memory and execution time efficiency.

An efficient encoding of the reachability graph is built in the form of a set of Kronecker matrices $\mathbf{W}_{e,k}$ with $e \in \mathcal{A}$ and $k = 1, \dots, K$, where \mathcal{A} is the set collecting all the system events or activities. $\mathbf{W}_{e,k}[i_k, j_k] = 1$ if state j_k of sub-model k is reachable from state i_k due to event e . According to such a definition, the next state function of the model can be encoded as the incidence matrix given by the boolean sum of Kronecker products $\sum_{e \in \mathcal{A}} \bigotimes_{K \geq k \geq 1} \mathbf{W}_{e,k}$. As a consequence, the matrix representation \mathbf{R} of the reachability graph of the model can be obtained by filtering the rows and columns of such a matrix corresponding to the reachable global states encoded in the MDD and replacing each non-null element with the labels of the events that cause the corresponding state transition.

Saturation Unbound is a very effective way to represent the model state space and the related reachability graph of a model. In any case, the methodology we are dealing with is not strictly dependent on any particular algorithm to efficiently store the reachability graph. We refer to the *Saturation Unbound* algorithm simply because its efficiency is well known [7].

6.2 Annotating the Reachability Graph

The use of Saturation together with the Kronecker representation presented in previous sections enable solution of the derived stochastic process. However, knowledge of the reachability graph of the untimed system as produced by Saturation is not sufficient to manage the infinitesimal generator matrix \mathbf{Q} on the fly according to the symbolic representation. Considering that the information about the enabled events for all the system states is contained in the high level description of the model and it can be evaluated on the fly when needed with a negligible overhead, the only additional information needed is knowledge about the sets of active but not enabled events in each state s ($T_a^{(s)}$). Using Saturation for the evaluation of the reachability graph requires an additional analysis step for the computation of such an information and use of a different data structure for storage. Multi Terminal Multi-Valued Decision Diagram (MTMDD) [15] is used for this purpose.

The main differences with respect to MDDs are that: (1) more than two terminal nodes are present in an MTMDD and (2) such nodes can be labeled with arbitrary integer values, rather than just 0 and 1. An MTMDD can efficiently store both the

system state space S and the sets $T_a^{(s)}$ of active but not enabled events for all $s \in S$; this is necessary, in our approach, to correctly evaluate non-null blocks of \mathbf{Q} matrix. In fact, while an MDD is only able to encode a state space, an MTMDD is also able to associate an integer to each state. Thus, the encoding of sets $T_a^{(s)}$ can be done associating to each possible set of events an integer code that unambiguously represents it. Let us associate to each event an unique index n such that $1 \leq n \leq \|\mathcal{A}\|$. Then the integer value associated to one of the possible sets $T_a^{(s)}$ is computed starting from the indices associated with the system events that belong to it in the following way:

$$b_M \cdot 2^A + \dots + b_n \cdot 2^n + \dots + b_1 \cdot 2^1 + 1 = \sum_{i=1}^M b_i 2^i + 1$$

where

$$b_i = \begin{cases} 1, & \text{if event } e_i \in T_a^{(s)} \\ 0, & \text{otherwise} \end{cases}$$

In this manner all the necessary information to apply the Kronecker-based expressions on the fly are provided; the only remaining need is a method to evaluate the set $T_a^{(s)}$ given a referring state s .

In [13], the following theorem has been proved.

Theorem 1 *Given a model \mathcal{M} , a state $s_0 \in S$ and an event $\bar{e} \in \mathcal{A}$ with an age memory policy associated, then $\bar{e} \in T_a^{(s_0)}$ iff $\bar{e} \notin T_e^{(s_0)}$ and one of the following statements holds:*

1. $\exists s_1 \in S, \exists e_1 \in \mathcal{A}, s_1 \neq s_0, e_1 \neq \bar{e} \mid s_0 \in \mathcal{N}_{e_1}(s_1) \wedge \bar{e} \in T_e^{(s_1)}$
2. $\exists s_1 \in S, s_1 \neq s_0 \mid s_0 \in \mathcal{N}(s_1) \wedge \bar{e} \in T_a^{(s_1)}$

where \mathcal{N}_{e_1} is the next state function associated to event e_1 .

Note that function \mathcal{N} is the equivalent to the $n(\cdot, \cdot)$ defined in Sect. 4; function \mathcal{N}_e instead differs for the restriction to the firing of a specific event e . We use this notation because it is less cumbersome in this specific context.

Theorem 1 gives a way to evaluate if an event e belongs to the set $T_a^{(s_0)}$ or not. In fact, according to the statements proved, it is possible to characterize a state s_0 with respect to the system event memory policies by exploring its reachability graph. Exploration can be performed using classical bread-first search and depth-first search algorithms, easily applicable to an explicitly stored reachability graph; it is more complicated to apply classical search algorithms when the graph is stored in implicit manner as is the case when MTMDD data structures are used.

In this case, a different approach can be used by resorting to Computational Tree Logic (CTL) formulas that have been shown to be very efficient for data structures like MDD and MTMDD. The use of CTL formulas to evaluate sets $T_a^{(s)}$ is justified by a theorem introduced in [13]. Before recalling this theorem, we need to introduce a CTL operator.

Definition 3 Let $s_0 \in \mathcal{S}$ be a state of a discrete state process with state space \mathcal{S} , and let p and q be two logical conditions on the states. Let also $\mathcal{F}(s) \subseteq \mathcal{N}(s) \cup \mathcal{N}^{-1}(s)$ be a reachability relationship between two states in \mathcal{S} that defines a desired condition over the paths. Then s_0 satisfies the formula $E_{\mathcal{F}}[pUq]$, and we will write $s_0 \models E_{\mathcal{F}}[pUq]$, iff $\exists n \geq 0, \exists s_1 \in \mathcal{F}(s_0), \dots, \exists s_n \in \mathcal{F}(s_{n-1}) \mid (s_n \models q) \wedge (\forall m < n, s_m \models p)$.

In definition above, we used the path quantifier E with the meaning *there exists a path* and the tense operator U with the meaning *until*, as usually adopted in CTL formulas.

Given Definition 3, the following theorem holds:

Theorem 2 An event $\bar{e} \in \mathcal{E}$, with an age memory policy associated, belongs to $T_a^{(s_0)}$, with $s_0 \in \mathcal{S}$, iff $s_0 \models E_{\mathcal{F}}[pUq]$ over a path at least one long, where p and q are the statements “ \bar{e} is not enabled” and “ \bar{e} is enabled,” respectively, and $\mathcal{F}(s) = \mathcal{N}^{-1}(s) \setminus \mathcal{N}_{\bar{e}}^{-1}(s)$.

Thanks to Theorem 2, evaluation of the CTL formula $E_{\mathcal{F}}[pUq]$ makes possible to evaluate whether an event \bar{e} is active but not enabled in state s_0 or not by setting condition p as \bar{e} is not enable and q as \bar{e} is enabled. This is the last brick to build an algorithm able to compute state probabilities of a model, where the event are PH or ME distributed; in fact, it is possible to characterize all the active and/or enabled events in all the different states and to apply the Kronecker expressions with this information to solve the derived EMC.

7 Examples

In this section, we present two examples where non-exponentially distributed durations are present. In the first example, these durations are approximated by PH distributions, while in the second example they are described by ME distributions.

7.1 Reliability Model of Computer System

We introduce a reliability model where we use PH distributions as failure times. The model is specified according to the Petri net depicted in Fig. 1, where the usual graphical notation for the places, transitions, and arcs has been adopted.

The system under study is a distributed computing system composed of a cluster of two computers. Each of them has three main weak points: the motherboard, CPU, and disk. Interconnections inside the cluster are provided by a manager in such a way that the overall system is seen as a single unit. In the distributed system, the two computers work independently, driven by the manager that acts as a load balancer to split the work between them. Since the manager represents a single point of failure, a second instance is deployed for redundancy in the system; this latter instance operates in cold standby when the main computer manager works and it is powered on when it fails.

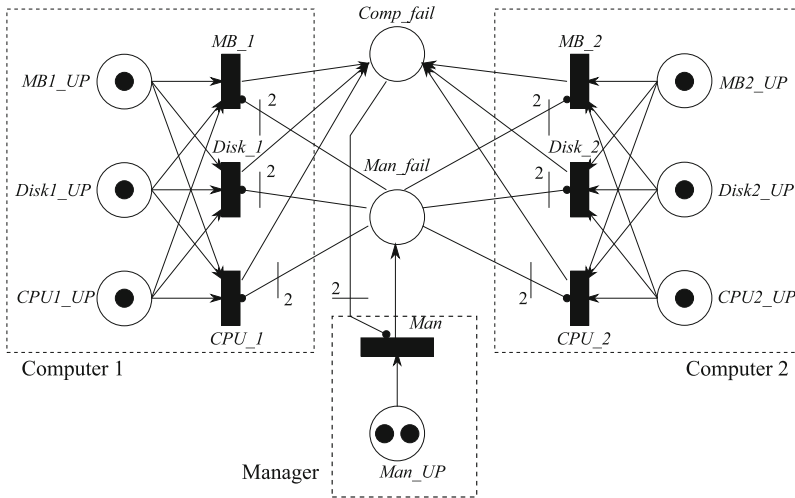


Fig. 1 Computer system reliability model

Due to this configuration, the distributed system works when at least one of the two computers works and the computer manager properly operates. The main components of each computational unit (CPU, motherboard, and disk) may fail rendering the unit inoperable. In the Petri net model, faults in the CPU, motherboard, and disk are modeled by the timed transitions MB_i , $Disk_i$, and CPU_i whose firing represents the respective faulty event in the i -th Computer; the operating conditions of components are represented by a token in the places $CPUi_UP$, MBi_UP , and $Diski_UP$. When one of the transitions above fires a token is flushed out of the place and a token is put in the place $Comp_fail$. At the same time, all the other transitions related to the faulty events in the same unit become disabled because the unit is considered down and thus no more faults can occur. Two tokens in the place $Comp_fail$ means that the two computational units are both broken and the overall distributed system is not operational. Similarly, transition Man models the fault of a manager unit. Its firing flushes a token out of the place Man_UP and puts a token in the place Man_fail . Thanks to the redundancy, the first manager unit fault is tolerated whereas the system goes down when a second fault occurs. This state is represented in the Petri net by two tokens in the place Man_fail . In both faulty states, all the transitions are disabled and an absorbing state is reached. In terms of Petri net objects, the not operational condition is expressed by the following statement:

$$(\#Comp_fail = 2) \vee (\#Man_fail = 2), \quad (6)$$

where the symbol $\#P$ states the number of token in place P .

Table 1 Failure time distribution parameters

Transition	Weibull			
	β_f	η_f	E	λ
<i>MB_1, MB_2</i>	0.5965	1.20	1.82	0.55
<i>Disk_1, Disk_2</i>	0.5415	1.00	1.71	0.59
<i>CPU_1, CPU_2</i>	0.399	1.46	3.42	0.29
<i>Man</i>	0.5965	1.20	1.82	0.55

As usual in reliability modeling, the time to failure of the components has been modeled using Weibull distributions whose cumulative distribution function is

$$F(t) = 1 - e^{-(t/\eta_f)^{\beta_f}}.$$

This choice has been also supported by measures done on real systems such as those analyzed in [9]. The parameters of the Weibull distributions used for the Petri net transitions of Fig. 1 are reported in Table 1.

Weibull distributions have been introduced in the model through the use of 10-phase PH distributions, approximating them by evaluating the formula (6). The results obtained are depicted in Fig. 2. To better highlight the usefulness of the modeling approach presented here, the Petri net model was solved by imposing exponential distributions as transition firing times. In fact, the use of exponential distributions is quite common to obtain a more tractable model. The value of the parameters λ used in this second run was computed as the reciprocal of the expected value, E , of the corresponding Weibull distributions (listed in Table 1). The result obtained are also depicted in Fig. 2. As can be easily noted, the use of exponential distributions produces optimistic results compared to the use of Weibull distributions, making the system appear more reliable than it is in reality.

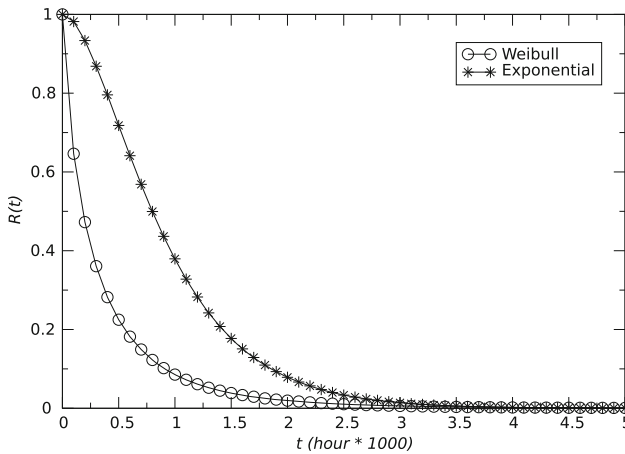


Fig. 2 Computer system reliability $R(t)$

7.2 Numerical Example with “Oscillating” ME Distribution

For our second example, we consider the Activity Network depicted in Fig. 3, which represents a “mission” composed of five activities and the constraints on the order in which the five activities can be carried out. Initially, activities 1 and 2 are active. If activity 1 finishes then activities 3 and 4 start and thus there are three activities under execution, namely, activities 2, 3, and 4. If activity 3 is the first first among these three activities to finish then no new activity starts because in order to start activity 5, both activity 2 and 3 must finish. The graph of all the possible states of the Activity Network is shown in Fig. 4, where in every node we report the activities that are under execution in the node. The label on the edges indicates the activity whose completion triggers the edge. The duration of the activities are modeled with ME distributions and we denote the vector and matrix that represent the duration of activity i by α_i and A_i , respectively. Further, we use the notation $a_i = (-A_i)\mathbf{1}$ and denote by I_i the identity matrix whose dimension is equal to that of A_i .

Following the approach described in Sect. 4, one can determine the infinitesimal generator of the model. Its first seven block-columns are given as (the left side of the matrix)

$A_1 \oplus A_2$	$a_1 \otimes I_2 \otimes A_3 \otimes A_4$	0	0	$I_1 \otimes a_2$	0	0
0	$A_2 \oplus A_3 \oplus A_4$	$I_2 \otimes I_3 \otimes a_4$	0	0	$I_2 \otimes a_3 \otimes I_4$	$a_2 \otimes I_3 \otimes I_4$
0	0	$A_2 \oplus A_3$	$I_2 \otimes a_3$	0	0	0
0	0	0	A_2	0	0	0
0	0	0	0	A_1	0	$a_1 \otimes A_3 \otimes A_4$
0	0	0	$I_2 \otimes a_4$	0	$A_2 \oplus A_4$	0
0	0	0	0	0	0	$A_3 \oplus A_4$
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

and the remaining five block-columns are (the right side of the matrix)

Fig. 3 An activity network

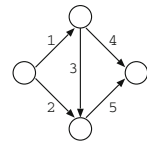
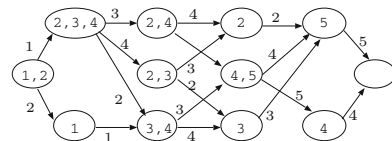


Fig. 4 CTMC of the activity network in Fig. 3



$$\begin{array}{cccc|c}
 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 \\
 a_2 \otimes I_3 & 0 & 0 & 0 & 0 \\
 0 & a_2 \otimes A_5 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & a_2 \otimes I_4 \otimes A_5 & 0 & 0 \\
 I_3 \otimes a_4 & 0 & a_3 \otimes I_4 \otimes A_5 & 0 & 0 \\
 A_3 & a_3 \otimes A_5 & 0 & 0 & 0 \\
 0 & A_5 & 0 & 0 & a_5 \\
 0 & a_4 \otimes I_5 & A_4 \oplus A_5 & I_4 \otimes a_5 & 0 \\
 0 & 0 & 0 & A_4 & a_4 \\
 0 & 0 & 0 & 0 & 0
 \end{array}$$

The vector that provides the initial configuration is $|A_1 \otimes A_2, 0, \dots, 0|$.

In order to illustrate a feature of ME distributions that cannot be exhibited by PH distributions, we applied an ME distribution with “oscillating” PDF to describe the duration of activities 1, 2, 4, and 5. The vector–matrix pair of this ME distribution is

$$A_1 = A_2 = A_4 = A_5 = |1.04865, -0.0340166, -0.0146293| ,$$

$$A_1 = A_2 = A_4 = A_5 = \begin{vmatrix} -1 & 0 & 0 \\ 0 & -1 & -20 \\ 0 & 20 & -1 \end{vmatrix} ,$$

and its PDF is depicted in Fig. 5. The duration of the remaining activity, namely activity 3, is distributed according to an Erlang distribution with four phases and average execution time equal to 1, i.e.,

$$A_3 = |1, 0, 0, 0| , \quad A_3 = \frac{1}{4} \begin{vmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{vmatrix} .$$

Fig. 5 Oscillating activity duration pdf

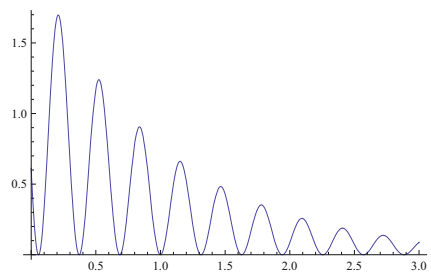
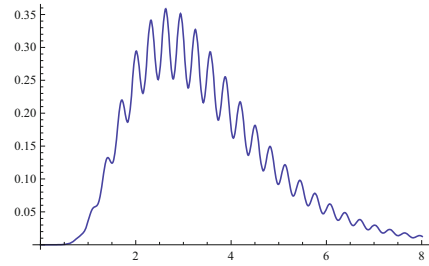


Fig. 6 Overall accomplishment time pdf



The model was then used to characterize the PDF of the time that is needed to accomplish the whole mission. The resulting PDF is shown in Fig. 6 and one can observe that the oscillating nature of the distribution of the activity durations carries over into the overall completion time distribution.

8 Conclusions

While the evolution of computing devices and analysis methods resulted in a sharp increase in the complexity of computable CTMC models, CTMC-based analysis had been restricted to the analysis of stochastic models with exponentially distributed duration times. A potential extension of CTMC-based analysis is the inclusion of PH distributed duration times, which enlarges the state space, but still possesses feasible computational complexity. We surveyed the basics of PH distributions and the analysis approach to generate the EMC.

A more recent development in this field is the extension of the EMC-based analysis with ME distributed duration times. With respect to the steps of the analysis method, the EMC-based analysis and its extension with ME distributions are identical. However, because ME distributions are more flexible than the PH distributions (more precisely, the set of PH distributions of a given size is a subset of the set of ME distributions of the same size) this extension increases the modeling flexibility of the set of models which can be analyzed with a given computational complexity.

Apart of the steps of the EMC-based analysis method, we discussed the tool support available for the automatic execution of the analysis method. Finally, application examples demonstrated the abilities of the modeling and analysis methods.

References

1. Asmussen S, Bladt M (1999) Point processes with finite-dimensional conditional probabilities. *Stoch Process Appl* 82:127–142
2. Bean NG, Nielsen BF (2010) Quasi-birth-and-death processes with rational arrival process components. *Stoch Models* 26(3):309–334

3. Buchholz P, Horvath A, Telek M (2011) Stochastic Petri nets with low variation matrix exponentially distributed firing time. *Int J Perform Eng* 7:441–454, 2011 (Special issue on performance and dependability modeling of dynamic systems)
4. Buchholz P, Telek M (2010) Stochastic petri nets with matrix exponentially distributed firing times. *Perform Eval* 67:1373–1385
5. Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ (1990) Symbolic model checking: 1020 states and beyond. In: Fifth annual IEEE symposium on logic in computer science, 1990. LICS '90, Proceedings, pp 428–439
6. Ciardo G, Luttgen G, Siminiceanu R (2001) Saturation: an efficient iteration strategy for symbolic state space generation. In: Proceedings of tools and algorithms for the construction and analysis of systems (TACAS), LNCS 2031. Springer, pp 328–342
7. Ciardo G, Marmorstein R, Siminiceanu R (2003) Saturation unbound. In: Proceedings of TACAS. Springer, pp 379–393
8. Cox DR (1955) The analysis of non-markovian stochastic processes by the inclusion of supplementary variables. *Proc Cambridge Philos Soc* 51(3):433–441
9. Distefano S, Longo F, Scarpa M, Trivedi KS (2014) Non-markovian modeling of a blade-center chassis midplane. In: Computer performance engineering, vol 8721 of Lecture Notes in Computer Science. Springer International Publishing, pp 255–269
10. Kleinrock L (1975) Queuing systems, vol 1: theory. Wiley Interscience, New York
11. Kulkarni VG (1995) Modeling and analysis of stochastic systems. Chapman & Hall
12. Lipsky L (2008) Queueing theory: a linear algebraic approach. Springer
13. Longo F, Scarpa M (2015) Two-layer symbolic representation for stochastic models with phase-type distributed events. *Int J Syst Sci* 46(9):1540–1571
14. Miner AS, Ciardo G (1999) Efficient reachability set generation and storage using decision diagrams. In: Application and Theory of Petri Nets 1999 (Proceedings 20th international conference on applications and theory of Petri Nets. Springer, pp 6–25)
15. Miner A, Parker D (2004) Symbolic representations and analysis of large state spaces. In: Validation of stochastic systems, LNCS 2925, Dagstuhl (Germany). Springer, pp 296–338
16. Neuts M (1975) Probability distributions of phase type. In: Amicorum L, Florin EH (eds) University of Louvain, pp 173–206
17. Scarpa M, Bobbio A (1998) Kronecker representation of stochastic petri nets with discrete ph distributions. In: Proceedings of IEEE international computer performance and dependability symposium, 1998. IPDS'98. pp 52–62
18. Srinivasan A, Ham T, Malik S, Brayton RK (1990) Algorithms for discrete function manipulation. In: IEEE international conference on computer-aided design, 1990. ICCAD-90. Digest of technical papers, pp 92–95
19. Trivedi K (1982) Probability and statistics with reliability, queueing and computer science applications. Prentice-Hall, Englewood Cliffs

An Analytical Framework to Deal with Changing Points and Variable Distributions in Quality Assessment

Dario Bruneo, Salvatore Distefano, Francesco Longo
and Marco Scarpa

Abstract Nonfunctional properties such as dependability and performance have growing impact on the design of a broad range of systems and services, where tighter constraints and stronger requirements have to be met. This way, aspects such as dependencies or interference, quite often neglected, now have to be taken into account due to the higher demand in terms of quality. In this chapter, we associate such aspects with operating conditions for a system, proposing an analytical framework to evaluate the effects of condition changing to the system quality properties. Starting from the phase type expansion technique, we developed a fitting algorithm able to catch the behavior of the system at changing points, implementing a codomain memory policy forcing the continuity of the observed quantity when operating conditions change. Then, to also deal with the state-space explosion problem of the underlying stochastic process, we resort to Kronecker algebra providing a tool able to evaluate, both in transient and steady states, nonfunctional properties of systems affected by variable operating conditions. Some examples from different domains are discussed to demonstrate the effectiveness of the proposed framework and its suitability to a wide range of problems.

D. Bruneo (✉) · F. Longo · M. Scarpa
Dipartimento di Ingegneria, Università degli Studi di Messina,
C.da di Dio, 98166 Messina, Italy
e-mail: dbruneo@unime.it

F. Longo
e-mail: flongo@unime.it

M. Scarpa
e-mail: mscarpa@unime.it

S. Distefano
Higher Institute for Information Technology and Information Systems
Kazan Federal University, 35 Kremlevskaya Street, 420008 Kazan, Russia
e-mail: sdistefano@kpfu.ru; sdistefano@unime.it

S. Distefano
Dipartimento di Scienze Matematiche e Informatiche, Scienze Fisiche e Scienze della Terra,
Università degli Studi di Messina, Viale F. Stagno d'Alcontres, 31, 98166 Messina, Italy

1 Introduction

Current research trends move toward new technologies and solutions that have to satisfy tight requirements raising the quality standards and calling for adequate techniques, mechanisms, and tools for measuring, modeling, and assessing nonfunctional properties and parameters. Aspects so far coarsely approximated or neglected altogether must now be taken into account in the quantitative evaluation of modern systems. This encompasses a wide range of phenomena, where it is often necessary to consider the relationships among different quantities in terms of aggregated metrics (e.g., dependability, performability, sustainability, and quality of service).

A particularly interesting aspect that often emerges while dealing with the continuously growing complexity of modern systems is the fallouts on observed quantities due to *changes*. The notion of change that we consider in the present work takes into account both the interdependencies among internal quantities, parts, or subparts, and the external environment interactions. Workload fluctuations, energy variations, environmental phenomena, user-related changing behaviors, and changing system operating conditions are only few examples of such aspects that can no longer be neglected in quantitative assessment [1, 2].

When non-exponentially distributed events are associated with system quantities, an important issue to address is the *memory* representation. In fact, in presence of changes, a wide class of phenomena requires that the quantity distribution may change its behavior still preserving its continuity. More specifically, the stochastic characterization at changing points, where the distributions governing the underlying phenomena affecting the observed quantity could change, is strategic to have a good and trustworthy representation of the actual system through a model. Stochastic processes more complex and powerful than Markov models are necessary in order to model this kind of memory, such as *semi-Markov processes* [3] or those deriving from *renewal theory* [4], but these techniques are sometimes not able to cover some specific aspects. Other modeling formalisms, such as fluid models [5], can deal with more complex phenomena but their analytic solution cannot always be easily automated, thus often requiring simulation [6].

In this chapter, we collect the results obtained in some previous works dealing with the above discussed aspects and phenomena, thus proposing a detailed problem analysis, a well-established analytical framework, as well as a comprehensive set of case studies with examples coming from ICT areas (network, Cloud, IoT) and science (physics).

The key aspects of the proposed modeling technique are: (i) the stochastic characterization of the observed quantity in different working conditions; (ii) the specification of the working condition process governing the corresponding variations; (iii) the modeling of the behavior of the observed quantity at changing points.

In particular, at changing points, the quantity distribution may change its behavior, preserving, under specific assumptions, its continuity. In order to implement the continuity constraints imposed by the changing working conditions in the presence

of non-exponential distributions, we propose a solution technique based on phase type distributions, providing an ad hoc fitting algorithm based on codomain discretization.

The chapter is organized as follows. Section 2 introduces the scenario considered also presenting background details that can be useful in order to understand the proposed approach. Section 3 presents our ad hoc fitting algorithm, while Sect. 4 explains how to adopt this algorithm in the evaluation of a system affected by variable operating conditions through stochastic models. Section 5 shows how the proposed framework can be exploited to deal with a variety of application domains. Section 6 concludes the work with some final remarks.

2 Preliminary Concepts

2.1 Problem Rationale

Let us consider a continuous random variable X defined on the sample space Ω stochastically characterizing a generic event. Moreover, let us assume such an event is also characterized and influenced by a specific *condition* belonging to a numerable set of conditions $c_X^i \in \mathbf{C}_X$ identified by $i \in \mathbb{N}$. Thus, the values of $X \in \Omega$ also depend on the condition in a way that X in the i th specific condition is characterized by the cumulative distribution function (CDF) $F_X^i(x) = Pr\{X \leq x \mid \text{The condition is } c_X^i\}$ and the probability density function (PDF) $f_X^i(x) = d(F_X^i(x))/dx$. Let us assume $F_X^i(x) \in \mathbf{F}_X$ is *continuous* and *strictly increasing* for all i .

To stochastically characterize conditions, let us consider another continuous random variable $Y_{i,j}$ on Ω . Specifically, $Y_{i,j}$ triggers the switching of X from condition c_X^i to c_X^j , with $c_X^i \neq c_X^j \in \mathbf{C}_X$. Assuming that $Y_{i,j}$ is characterized by the $F_{Y_{i,j}}(y)$ CDF (and by the $f_{Y_{i,j}}(y)$ PDF), we can identify a set \mathbf{Y} of condition switching random variables and a set \mathbf{F}_Y of CDFs. Moreover, considering the following CDF stochastically characterizing the event related to X when condition c_X varies:

$$F_X(x, c_X) = Pr\{X \leq x \wedge \text{The condition is } c_X \in \mathbf{C}_X\}$$

since the condition changes are triggered by the corresponding random variables in \mathbf{Y} , such a CDF can be treated as a joint CDF of X and \mathbf{Y} , $F_X(x, c_X) = F_{X,\mathbf{Y}}(x, \mathbf{y})$. We also impose that $F_X(x, c_X)$ is a continuous function with respect to c . This choice reflects the behavior of many physical systems where quantities under study have no sudden changes. The main consequence of this continuity is that at the condition changing points there is no probability mass for $F_X(x, c)$.

Our aim is to express $F_X(x, c)$ (or equivalently, $F_{X,\mathbf{Y}}(x, \mathbf{y})$) in terms of its marginal distributions $F_X^i(x) \in \mathbf{F}_X$ and $F_{Y_{i,j}}(y) \in \mathbf{F}_Y$. This is an undefined problem if no further hypotheses are assumed. To proceed with our discussion, let us consider that just two conditions for X are possible and that they are characterized by the CDFs $F_X^1(x)$ and $F_X^2(x)$ respectively. Moreover, let us assume we start with condition c_1 and then to

switch to condition c_2 . Thus, we are considering just one change point from c_1 to c_2 with $F_{Y_{1,2}}(y) = F_Y(y)$. We can express $F_{X,Y}(x, y)$ as

$$F_{X,Y}(x, y) = \begin{cases} F_X^1(x) & x \leq y \\ F_X^2(x + \tau) & x > y \end{cases} \quad (1)$$

where $\tau \in \Omega$ is a constant depending on the change point such that $x, y, x + \tau \in \Omega$ and $F_{X,Y}(x, y)$ is continuous in y . To quantify τ , we need to understand what happens at the change point. Given that $F_{X,Y}(x, y)$ must be continuous and strictly increasing (and thus invertible) by Eq. (1),

$$F_X^1(y) = F_X^2(y + \tau) \Rightarrow \tau = F_X^{2(-1)}(F_X^1(y)) - y \quad (2)$$

where $F_X^{2(-1)}(\cdot)$ is the inverse function of $F_X^2(\cdot)$. Finally, a simple application of the law of total probability allows us to decondition $F_{X,Y}(x, y)$, obtaining

$$\begin{aligned} F_{X,Y}(t) &= \int_{-\infty}^{+\infty} \Pr(X \leq t | Y = y) f_Y(y) dy \\ &= \int_0^t \Pr(X \leq t | Y = y) f_Y(y) dy \\ &\quad + \int_t^{+\infty} \Pr(X \leq t | Y = y) f_Y(y) dy \\ &= \int_0^t (1 - \Pr(X > t | Y = y)) f_Y(y) dy \\ &\quad + F_X^1(t) (F_Y(y)|_t^\infty) \end{aligned} \quad (3)$$

This way, considering $y \leq t$, $\Pr(X > t | Y = y) = 1 - F_X^2(t + \tau) = 1 - F_X^2(t + F_X^{2(-1)}(F_X^1(y)) - y)$ and thus by Eq. (3)

$$\begin{aligned} F_{X,Y}(t) &= F_X^1(t) (1 - F_Y(t)) \\ &\quad + \int_0^t F_X^2(t + F_X^{2(-1)}(F_X^1(y)) - y) f_Y(y) dy \end{aligned} \quad (4)$$

Note that Eq. (4) has been deduced by simply applying probability theory. Therefore, it is valid whenever the above model and related assumptions (continuity and invertibility of marginal CDFs, and no probability mass at changing point for the joint CDF) are satisfied. We can associate several physical meanings with such an equation. All of them are related to *conservation laws* in different application areas, such as stochastic modeling (conservation of reliability-Sedyakin model [2], survival models, duration models), physics (e.g., conservation of energy, momentum), economics, sociology, history, biology, and so on.

2.2 Phase Type

Phase type distributions were first used by Erlang [7] in his work on congestion in telephone systems at the beginning of this century. His *method of stages*, that has been further generalized since then, represents a non-exponentially distributed state sojourn time by a combination of *stages*, each of which is exponentially distributed. In this way, once a proper stage combination has been found to represent or approximate a distribution, the whole process becomes Markovian and can be easily analyzed through well-known and established approaches. Note that the division into stages is an operational device. Thus, stages have no physical significance in general.

Later, Neuts [8, 9] formally defined the class of phase type distributions. A non-negative random variable T (or its distribution function) is said to be of phase type (PH) when T is the time until absorption of a finite-state Markov chain [9]. When continuous time Markov chains are considered, we talk about *continuous phase-type* (CPH) distributions. In particular, we say that T is distributed according to a CPH distribution with representation (α, \mathbf{G}) and of order n if α and \mathbf{G} are such that $\pi(0) = [\alpha, \alpha_{n+1}]$ is the $n + 1$ dimensional *initial probability vector* in which $\alpha_{n+1} = 1 - \sum_{i=1}^n \alpha_i$ and $\hat{\mathbf{G}} \in \mathbb{R}^{n+1} \times \mathbb{R}^{n+1}$ is the *infinitesimal generator matrix* of the underlying continuous time Markov chain in which

$$\hat{\mathbf{G}} = \left[\begin{array}{c|c} \mathbf{G} & \mathbf{U} \\ \hline \mathbf{0} & 0 \end{array} \right]$$

Matrix $\mathbf{G} \in \mathbb{R}^n \times \mathbb{R}^n$ describes the transient behavior of the CTMC and $\mathbf{U} \in \mathbb{R}^n$ is a vector grouping the transition rates to the absorbing state. Matrix $\hat{\mathbf{G}}$ must be stochastic so $\mathbf{U} = -\mathbf{G}\mathbf{1}$, where $\mathbf{1} \in \mathbb{R}^n$ is a vector of 1. Thus, the CDF of T , $F_T(t)$, i.e. the probability of reaching the absorbing state of the CPH, is given by:

$$F(t) = 1 - \alpha e^{\mathbf{G}t} \mathbf{1}, \quad t \geq 0 \quad (5)$$

One of the main drawbacks of PH distributions is the state-space explosion. In fact, expanding each non-exponential distribution with a set of phases greatly increases the number of model states. In order to provide a compact representation of the resulting CTMC, Kronecker algebra is usually exploited [9–11], providing several advantages: (i) it is not necessary to physically generate nor store the expanded state space as a whole; (ii) the memory cost of the representation of the expanded state space grows linearly with the dimension of the PH distributions instead of geometrically; (iii) specific algorithms developed for similar cases can be used for the model solution [12].

The method of stages allows one to code memory through Markov models. Each phase represents a particular condition reached by the stochastic process during its evolution. Usually, this condition is associated to the time. CPHs are specified with the aim of coding the time instant at which a specific event occurs, by keeping memory of the phase reached so far. This allows resumption of the evolution of an

activity from the saved point if the activity is interrupted. This mechanism works if the observed events are not changing their nature. On the other hand, if the nature of the observed event changes preserving memory of the time instant may not be enough.

In such cases, as described in Sect. 2.1, since the distribution characterizing the observed phenomenon changes according to some event, it becomes a bi-variate function of both the observed event and the external one. Moreover, it is generally difficult or even impossible to obtain the joint CDF by only knowing the marginal CDFs.

As an example, let us consider that the observed event is a component failure characterized by the component lifetime CDF in a changing environment. In such a context, a way to adequately address the problem may be to characterize the lifetime CDFs of the observed system both in isolation or in the initial working condition (without dependencies or in the baseline environment), and in the new environment or after the dependency application. By representing the two lifetime CDFs as CPHs in this way, it would be necessary to specify how the system jumps from one CPH to the other one and viceversa. Such a “jump” process is usually governed by a law regulating the underlying system. For example, the conservation of reliability principle states that reliability should be preserved in the transition between the two conditions. Of course, other possible examples are the conservation of momentum, of age, of battery charge, and so on. In general, such quantities can be expressed as a function of time. Thus, the quantity to preserve is not necessarily the time but a (usually monotonic, nonincreasing) function of the time.

In several previous works, [13–23], we presented a method based on CPHs and Kronecker algebra to manage such kind of memory, without incurring in the complexity of the problem described in Sect. 2.1. The proposed method is able to manage many changing points affecting the reliability, availability, or other performability indexes that can change many times during the system evolution. The technique requires some assumptions on the CPHs structure in order to be easily implemented. In particular, it is necessary that the CPHs coding the functions associated to the changing indexes in different operating states have the same number of phases and initial probability vector. In order to deal with this issue, we propose an ad hoc fitting algorithm. The basic idea to represent the conservation of the considered quantity in terms of CPHs is to discretize the probability codomain associating to each stage of the CPH a specific value of the function. Thus, in order to jump from a CPH representing the behavior of the system under specific environmental conditions to the one representing the system in different environmental conditions, it is only necessary to save the phase reached, thus implementing the conservation of the considered quantity. In other words, a specific phase encodes the value reached by the quantity. Then, if we represent the two functions in terms of CPHs with the same number of phases n , the i th phase of the two CPHs, with $i \leq n$, corresponds to the same function value.

In this chapter, we give an overview of the technique also providing a set of simplified examples taken from previous works. The aim is to show that our approach can deal with a wide class of phenomena and to give the reader an overall idea of our work on such a topic.

3 PH Codomain-Fitting Algorithm

The goal of the fitting algorithm we propose is to associate a specific meaning, a memory tag or value, with the phases of the CPH, while approximating the original distribution through a CPH. The idea is to characterize the CPH phases with some information encoding the probability codomain, in order for each phase to represent a value of probability or better, an interval of probability values in $\mathbf{K}_i = [a_i, b_i] \subseteq [0, 1]$. The \mathbf{K}_i intervals must be mutually exclusive ($\mathbf{K}_i \cap \mathbf{K}_j = \emptyset \forall i \neq j$) and cover the whole $[0, 1]$ probability codomain ($\bigcup_i \mathbf{K}_i = [0, 1]$).

The first step to perform the proposed fitting algorithm is to discretize the codomain of the CDF to be approximated through continuous phase type. For example, applying this algorithm to the CDF shown in Fig. 1a, $n = 10$ contiguous intervals $\mathbf{K}_i = [i/n, (i+1)/n] \in [0, 1] \subset \mathbb{R}$, with $i = 0, \dots, n-1 \subset \mathbb{N}$, have been identified, characterizing $n+1$ endpoints in total. The proposed technique associates with each of such endpoints a phase of the corresponding CPH as reported in Fig. 1b for the CDF given as example, where phase 0 corresponds to probability 0, 1 to 0.1, 2 to 0.2, and so on. In general, the j th phase corresponds to the j/n probability with $j = 0, \dots, n$.

The transition between the states i and j is strictly related to the event characterizing the underlying process. Transitions are allowed either to the next phase of the CPH model or to the absorbing state, as shown in Fig. 1b. If, while in the i th interval, the event occurs a transition between the i th and the last n th phase is performed, otherwise, if the event does not occur, the CPH transits from the i th phase to the $(i+1)$ st.

A generic CPH obtained by applying the proposed technique is reported in Fig. 2, where $n+1$ phases labeled $0, \dots, n$ are identified, subdividing the probability codomain into n contiguous intervals $[i/n, (i+1)/n] \in [0, 1] \subset \mathbb{R}$ with $i = 0, \dots, n-1 \subset \mathbb{N}$. As stated above, two are the possible transitions outgoing from the i th phase: one incoming to the next $i+1$ phase and the other to the absorbing n th phase. To evaluate the rates to associate with such transitions, we start from the $i \rightarrow i+1$ transition. Assuming the considered event is not occurred before t_i and does not occur in the interval $[t_i, t_{i+1})$, the sojourn time in phase i depends only on the time spent in the i th state due to the Markov property of CTMCs. This sojourn time can be characterized by the random variable τ_i with mean value $E[\tau_i] = t_{i+1} - t_i$. Since τ_i has to be approximated by a random sojourn time characterized by the negative exponential rate λ_i , we have that that:

$$\lambda_i = \frac{1}{E[\tau_i]} = \frac{1}{t_{i+1} - t_i} \quad (6)$$

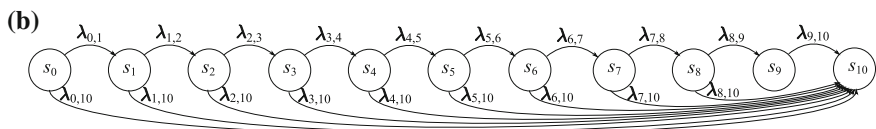
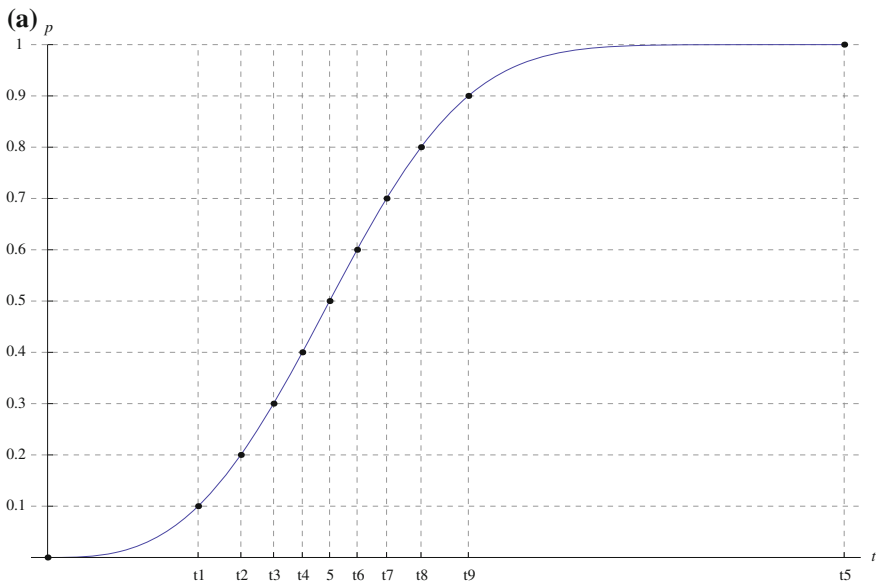


Fig. 1 A CDF Codomain Sampling (a) and corresponding CPH (b)

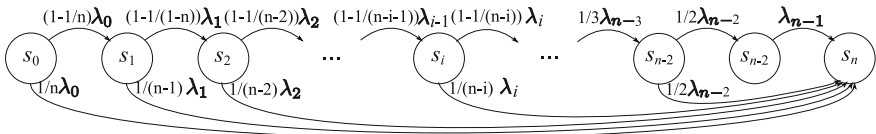


Fig. 2 A generic CPH of the proposed approach

It is worth noting that Eq. (6) is the rate of the CPH approximating the sojourn time in state i if the observed event has not occurred before t_i and does not occur in the interval $[t_i, t_{i+1})$. Thus, we have to identify the probability of transiting to state $i + 1$ and n from i . The $i \rightarrow i + 1$ probability $p_{i,i+1}$ is

$$\begin{aligned}
 p_{i,i+1} &= Pr\{X > t_{i+1} | X > t_i\} = 1 - Pr\{X \leq t_{i+1} | X > t_i\} \\
 &= 1 - \frac{Pr\{t_i < X \leq t_{i+1}\}}{Pr\{X > t_i\}} = 1 - \frac{F(t_{i+1}) - F(t_i)}{1 - F(t_i)} \\
 &= 1 - \frac{1/n}{1 - i/n} = 1 - \frac{1}{n - i}
 \end{aligned} \tag{7}$$

where X is the observed random variable and $F(t)$ is its corresponding CDF to be approximated by CPH. Since the approximation algorithm split the codomain into intervals of size $1/n$, we have that $F(t_{i+1}) - F(t_i) = 1/n$ and $F(t_i) = i/n$. On the other hand, the $i \rightarrow n$ probability $p_{i,n}$ is the complement of the Eq. (7):

$$\begin{aligned} p_{i,n} &= Pr\{X \leq t_{i+1} | X > t_i\} = 1 - Pr\{X > t_{i+1} | X > t_i\} \\ &= 1 - p_{i,i+1} = \frac{1}{n-i}. \end{aligned} \quad (8)$$

This way, the rate specified by Eq. (6) is split between the two possible transitions such that:

$$\lambda_{i,i+1} = p_{i,i+1} \lambda_i = \left(1 - \frac{1}{n-i}\right) \lambda_i \quad (9)$$

$$\lambda_{i,n} = p_{i,n} \lambda_i = \frac{\lambda_i}{n-i} \quad (10)$$

as shown in Fig. 2. The $n-1 \rightarrow n$ transition has only one rate λ_n that, since a CDF asymptotically reaches 1 at $t = \infty$, should be ideally 0. For the CTMC of Fig. 2 to be a CPH, there exists only one absorbing state. Therefore, in building the CPH, the last rate λ_n is characterized by a low value, approximating with only one phase the long asymptotic tail of the CDF considered, for example by a fraction of the CPH lowest rate λ_k . The proposed algorithm has some problems exhibits some inaccuracy when fitting long-tail distributions and in general of distributions with constant or quasi-constant segments. A possible solution or workaround could be to increase n , especially in those segments, adopting a kind of “resolution” trimming or tuning on them, to be taken into account in the retrieval of the codomain value associated with the phase.

4 Phase-Type Distributions to Model Variable Conditions

The fundamentals introduced in the previous section together with the fitting algorithm presented in Sect. 3 aim to more easily manage the phenomena of changing conditions. In fact, the method of stages is a way to code memory through a Markov model, since a phase of a CPH represents a particular condition reached by the underlying stochastic process during its evolution. In classical methods [24–26], the condition usually is associated with time in order to remember the elapsed time since the enabling of an event. In other words, such CPHs are specified to save the time instant in which a specific event occurs, by just saving the phase reached and to continue their evolution from the saved point. Such mechanism works if the observed events are not changing their nature. In the context where the component dynamics are subject to change, preserving time memory is useless due to the change in the distribution characterizing the firing time. In such cases, we need a specific represen-

tation of the change event in order to preserve the memory as expressed by Eq. (2) in Sect. 2.1. According to that relation the quantity to preserve is no longer the time but the probability to fire at the enabling time y . To directly apply Eq. (2) implies a search of the time shift τ and the computation of Eq. (4) that is numerically inapplicable in real cases where more than one changing event can happen at a state change.

Our proposal is to represent the event firing times in terms of CPHs obtained as result of the fitting algorithm of Sect. 3; in that case, the firing probability is discretized and a specific value of probability is associated to each stage of the CPH. Thus, in order to jump from a CPH T_1 representing the behavior of the system under specific environmental conditions to the one (T_2) representing the system in different environmental conditions, it is only necessary to save the phase reached, thus implementing the conservation of the firing probability. In other term, a specific phase must encode the value of reached firing probability, and, by representing the two CDF in terms of CPHs with the same number of phases n , the probability encoded by the i -th phase of a CPH, where $i \leq n$, is the same encoded in the i -th phase of the other CPH. Of course, T_1 and T_2 must have the same number of phases; their representation is different due to the different dynamic of the events into the two operating conditions.

Jumping from T_1 to T_2 maintaining the phase reached implements the behavior described by Eq. (2). In fact, the CDF value is maintained by construction and the time shift is a consequence of a different rate assigned in CPH T_2 . The fact that at a given time y the process switches from a phase k of T_1 to phase k of T_2 implies that the event in the new condition is as enabled by a different time with respect to the dynamic described through T_1 . In this way, we take into account of the time shift τ of Eq. (2). This behavior is depicted in Fig. 3.

When the events are coded through CPHs, expansion methods can be used to adequately study the stochastic process also in the case of non-exponentially distributed events. The memory conservation at the switching condition time is taken

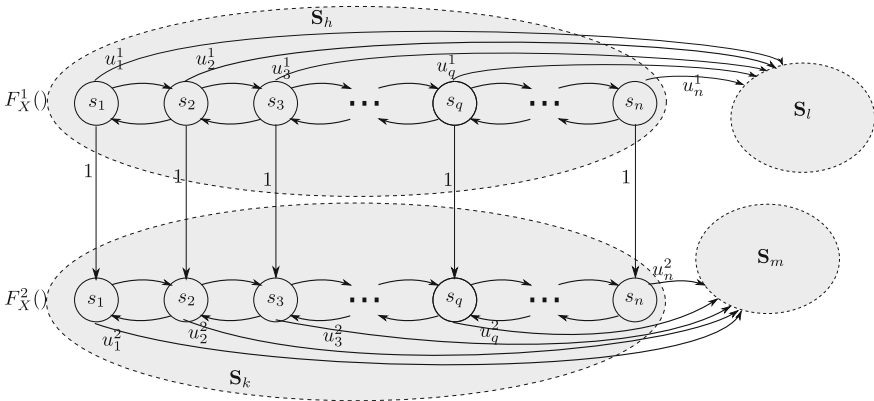


Fig. 3 CPH switching due to operating condition changing

into account using the CPH coding given above and associating to the changing event the classical memory policy managed by the expansion methods. These methods are very memory intensive. Thus, in order to provide a compact representation of CTMCs resulting from phase-type expansion, the Kronecker algebra is used as shown in [12, 25, 27–29].

4.1 Symbolic Representation

Let us consider a discrete-state discrete-event model and let S be the system state space and ε the set of CPH distributed system events. Following the state-space expansion approach [9], the stochastic process can be represented by an expanded CTMC. Such CTMC is composed of $\|S\|$ macro-states and it is characterized by a $\|S\| \times \|S\|$ block infinitesimal generator matrix \mathbf{Q} in which

- the generic diagonal block $\mathbf{Q}_{n,n}$ ($1 < n < \|S\|$) is a square matrix that describes the evolution of the CTMC inside the macro-state related to state n , which depends on the possible events that are enabled in such state;
- the generic off-diagonal block $\mathbf{Q}_{n,m}$ ($1 < n < \|S\|$, $1 < m < \|S\|$) describes the transition from the macro-state related to state n to the one related to state m , which depends on the events that occur in state n and on the possible events that are still able to occur in state m .

By exploiting Kronecker algebra, matrix \mathbf{Q} does not need to be generated and stored as a whole, but can be symbolically represented through Kronecker expressions and algorithmically evaluated on-the-fly when needed with consequent extreme memory saving [25, 28]. Moreover, this approach is particularly suitable in models where the memory conservation at the switching condition time has to be applied. Going into details of matrix \mathbf{Q} , its blocks have the following form:

$$\mathbf{Q}_{n,n} = \bigoplus_{1 < e < \|\varepsilon\|} \mathbf{Q}_e \quad (11)$$

$$\mathbf{Q}_{n,m} = \bigotimes_{1 < e < \|\varepsilon\|} \mathbf{Q}_e \quad (12)$$

In other words, the diagonal blocks are computed as Kronecker sums whereas off-diagonal blocks are computed as Kronecker products of a series of matrices \mathbf{Q}_e , each associated to a CPH representing system events. In the case of switching conditions incurred during a state change, the CPHs in the two states are different and this is reflected in Eq. (11) in the fact that matrix \mathbf{Q}_e related to the same event is different in the two states. Thanks to the fitting algorithm of Sect. 3, the structure of CPHs in different conditions can be modeled with the same number of phases n where each

phase codes a level of probability to fire. As a consequence, the conservation of the phase reached between the two CPHs the contribution of event e to the equation Eq. (12) is the rank n identity matrix \mathbf{I}_n .

5 Examples

In this section we aim at highlighting the generality of the proposed framework in modeling a wide range of systems. To this purpose, four examples are proposed dealing with both ICT and physics' systems and demonstrating the effectiveness of the approach in a variety of application fields.

5.1 Network Data Performance

In this example, we investigate both performance and reliability of data transferred through a connection-oriented virtual circuit network (VCN) at datalink or network layer such as X.25, ATM, Frame Relay, GPRS, or MPLS. To send data in a layer 2–3 VCN, a virtual circuit between the two endpoints must be established first. This means that data transmissions in such networks always follow the same path among nodes. Although this implies several benefits (bandwidth reservation, low overhead, and simple switching), the main drawback of such an approach is the virtual channel reliability, especially in wireless networks. Indeed, a channel failure usually breaks the virtual circuit forcing creation of a new connection. Several mechanisms and countermeasures, such as fault tolerance and redundancy policies, have been adopted by virtual circuit protocols to mitigate these failure effects. For example, bandwidth overprovisioning sends packet replicas to increase the delivery ratio. This often implies bandwidth overbooking to improve the channel utilization and the provider outcome. However, it impacts both the channel and the data transmission reliability, which become sensitive to network traffic fluctuations.

In this example, we focus on both reliability and performance of packet transmission in VCN, taking into account different traffic conditions in the evaluation of the packet time-to-failure and time-to-delivery r.v. Packet lifetime includes all the causes of delivery failure (channel failure, congestion, hardware and software faults, etc.). In other words, we aim at evaluating the probability the channel is reliable when and while a packet is sent, given that the channel is available at sending time.

As discussed above, we assume that both the packet time-to-failure X and time-to-delivery T also depend on the traffic, i.e. on the bandwidth available for transmission. To this end, we characterize two traffic conditions, *average* and *high*, assuming knowledge of the CDFs of the packet lifetime and time-to-delivery in such conditions, namely $F_X^A(t)$, $F_T^A(t)$, $F_X^H(t)$ and $F_T^H(t)$, respectively.

This way, the trigger events associated with traffic fluctuations are characterized by two CDFs $F^I(t)$ and $F^D(t)$ corresponding to the switching from average to high traffic

(traffic increase – I) and vice-versa (traffic decrease – D), respectively. Assuming periodic fluctuations between the two types of traffic, a bursty traffic pattern, a Markov modulated Poisson process can be adopted in the modeling if $F^I(t)$ and $F^D(t)$ are exponentially distributed.

The resulting state-space model is composed of four main states identifying the reliable channel both in the average (S_{T_A}) and in high (S_{T_H}) traffic conditions, the successful packet delivery (S_{P_D}), and the failed transmission (S_{P_F}), as shown in Fig. 4a. If the channel is available at the beginning, the system starts from either state S_{T_A} or S_{T_H} according to the traffic condition. The transitions between these two states model the traffic condition switching and are triggered by the e_{T_I} and e_{T_D} events representing traffic fluctuations characterized by the corresponding CDFs $F^I(t)$ and $F^D(t)$.

From such states, in the case of a failure during message sending, the state S_{P_F} is reached through transitions labeled e_{F_A} or e_{F_H} depending on the traffic condition, associated with $F_X^A(t)$ and $F_X^H(t)$ respectively. Otherwise, for successful delivery, state S_{P_D} is reached through events e_{D_A} or e_{D_H} , according to the traffic, characterized by $F_T^A(t)$ and $F_T^H(t)$ respectively. The goal is to evaluate the CDFs of the time to message delivery ($F_R(t)$), i.e., the probability of reaching state S_{P_D} , and the time to message sending failures ($F_F(t)$), i.e. the probability of reaching state S_{P_F} . Since states S_{P_D} and S_{P_F} are both absorbing, a steady state exists for the model and the values $F_R(\infty)$ and $F_F(\infty)$ represent the steady state probability the packet is delivered or fails, respectively, i.e. the packet/data delivery ratio, with $F_R(\infty) + F_F(\infty) = 1$. It follows that the that $F_R(t)$ and $F_F(t)$ are defective CDFs.

To evaluate the model, we based our analysis on parameters and values taken from the literature. With regard to the time to delivery, we used the data published in [30, 31], providing statistics of two workload conditions on the channel. Therefore, we stochastically characterized the time to delivery by Gaussian CDFs, the average traffic one ($F_T^A(t)$) with mean value 2.5 ms and variance 0.5 ms, the high traffic CDF ($F_T^H(t)$) has higher mean value (11 ms) and variance (3 ms).

The (sending) time to failure CDF has been obtained by the delivery ratio taken from [32], i.e. 0.9 in the average traffic case and 0.6 in the high one. This requires

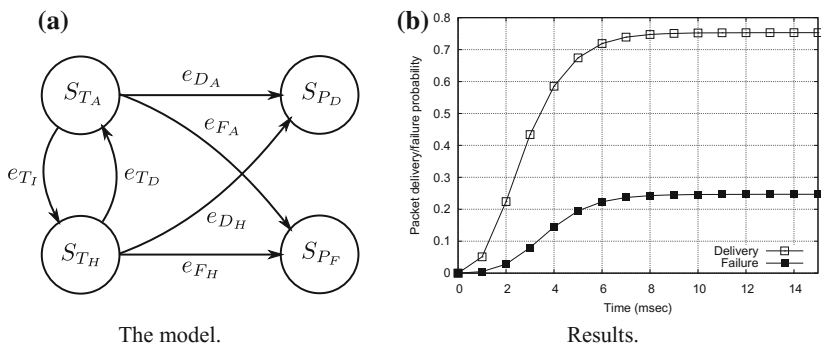


Fig. 4 Network data performability example

further manipulation. The delivery ratio is the steady state probability the packet is delivered in the above model. Knowing the delivery steady state probability in each traffic condition, we have to find a failure CDF such that the steady state probability of delivery in the model of Fig. 4a is equal to the delivery ratio. Assuming the message sending lifetime is represented by a Weibull CDF $(1 - e^{-(\frac{t}{\alpha})^\beta})$ commonly used in reliability, setting $\beta = 5$ through the model we identified $\alpha_A = 22$ ms for the average traffic and $\alpha_H = 4$ ms for the high one. Finally, the traffic switching rate is $\lambda_I = \lambda_D = 500 \text{ s}^{-1}$, i.e. a switch every 2 ms.

Applying the codomain fitting algorithm of Sect. 3 four 100 phase CPHs approximate these distributions. Then we evaluated the packet delivery and failure probability, obtaining the results shown in Fig. 4b. As discussed above, two defective distributions the time to delivery and the lifetime of the packet are identified, corresponding to states S_{P_D} and S_{P_F} , respectively. The steady state is reached at approximately 12 ms, with a delivery ratio of about 75 %. Similar transient trends can be observed. The first knee identifies a change in probability following the underlying CDFs, with initially a very low probability to deliver the packet, then suddenly increasing. The second knee represents the transition to the steady state, where it is very likely either the packet has been already delivered or sending failed.

5.2 IaaS Cloud Performance

Cloud computing systems offer virtualized environments where jobs, in terms of virtual machines (VMs), can be executed without any knowledge on the physical computing infrastructure. In such a scenario, multiple VMs can be allocated in the same physical machine (PM), e.g., a core in a multi-core architecture, in order to respond to specific provider goals such as energy consumption or load burst management. Multiple VMs sharing the same PM can incur in a reduction of the performance mainly due to I/O interference between VMs. In order to meet the stringent QoS level required by service-level agreements (SLAs), cloud providers must find trade-offs between the internal management strategies and the QoS offered. To this end, we propose an application of the proposed technique that could help data center managers to tune system parameters.

We model a system composed of a single PM that is able to concurrently execute up to m VMs. The actual number of VMs is a function of the traffic load as well as of the VM allocation strategy. We model such an aspect through a system composed of m states (see Fig. 5a), where transitions among states are triggered by events e_{C_I} (Increase) and e_{C_D} (Decrease) representing an increase or a decrease in the level of concurrency. Such events can be generally distributed with mean λ and μ , respectively and can be correlated through the *traffic intensity* factor ρ defined as $\rho = \frac{\lambda}{\mu}$. With low values of ρ (< 1), the system will experience a low concurrency level while with values of ρ near to 1 the system will experience a concurrency level near to m .

We are interested in measuring the time needed to execute a job that can be modeled through events $e_{T_1}, e_{T_2}, \dots, e_{T_m}$ that represent the execution times associated with

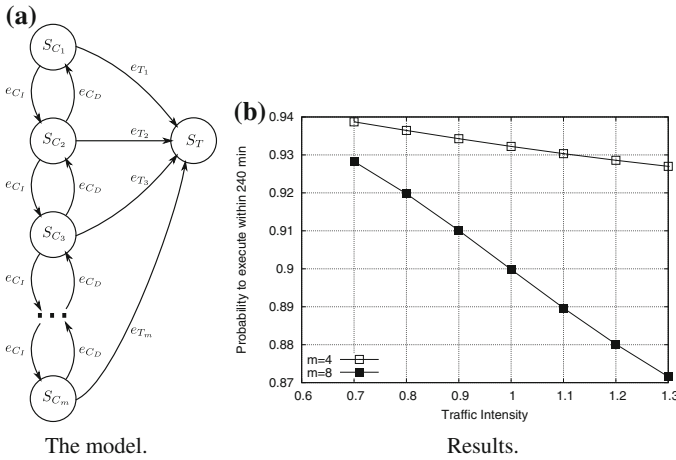


Fig. 5 The Cloud example

the different operating conditions. Events e_{T_i} are generally distributed and depend on the performance degradation due to the concurrent execution. They can be estimated by means of theoretical studies or statistical observations. Finally, the state S_T is the absorbing state, which corresponds to the task completion.

Measures of interest include the mean execution time or the probability to execute the task within a given time deadline τ . Such performance indexes can be investigated by varying the maximum concurrency level m under different load conditions characterized by different values of ρ .

In order to provide a quantitative example, we start defining a job whose duration T_1 can be modeled as a series of cycles composed of a CPU execution (modeled by an exponentially distributed execution time T_{CPU}^1 with mean λ_{CPU}) followed by an I/O access (modeled by an exponentially distributed execution time $T_{I/O}^1$ with mean $\lambda_{I/O}$). To reach the job completion, such events need to be repeated for a certain number of times. We model such an aspect by defining a probability p_f that is the probability of finishing the job at the end of a CPU-I/O cycle. The performance degradation [33] is modeled by varying the CPU and I/O execution times with respect to the concurrency level: $T_{CPU}^i = T_{CPU}^1 \cdot (1 + d_{CPU})^{(i-1)}$, $T_{I/O}^i = T_{I/O}^1 \cdot (1 + d_{I/O})^{(i-1)}$, where d_{CPU} and $d_{I/O}$ are the degradation factors related to CPU and I/O events, respectively. Notice that modeling job duration according to such a technique corresponds to a CPH distributed job completion time that, even without the specific form of Fig. 2, respects the characteristic discussed in Sect. 3, i.e., phases have a physical meaning and preserving memory of the phase is equal to preserving memory of the corresponding physical quantity (in this case the execution time).

Figure 5b shows the results obtained with the parameters $\lambda_{CPU} = 0.33 \text{ min}^{-1}$, $\lambda_{I/O} = 0.2 \text{ min}^{-1}$, $p_f = 0.9$, $d_{CPU} = 0.05$, $d_{I/O} = 0.1$. This figure gives an example of how the proposed technique can be exploited in order to obtain high level information on the cloud system. In particular, from the plotted curves, it is possible to quantify

the probability that a job executes within the threshold defined by an SLA (in this case $\tau = 240$ min). Such a probability can be evaluated under different load conditions and using different concurrency strategies, thus providing useful insights to the system administrator to identify a better set of strategies.

5.3 *Energy Consumption in IoT Mobile Devices*

While talking about mobile environments, the first thing that attracts attention is the comparison between resource-poor devices and fixed stations. Mobile devices are limited in computational power, available memory, and battery life. The presence of wireless connections with different available bandwidth and probability of disconnection and reconnection is another important characteristic of these environments. One of the more interesting topics in this research field is the Internet of Things (IoT), where concerns regarding mobility are even more constrained.

Mobile crowdsensing (MCS) [34] applications are a typical scenario in the IoT world. In such applications, hundreds or even thousands of mobile phones are employed to collect environmental information usually aggregated with user-generated patterns, in order to analyze and forecast crowd behavior and attitude measuring various individual as well as community trends. Individual trends are those pertaining to a single device(s) owner, while collective trends are those inherent to an aggregate of surroundings and not limited to any individual in particular. Examples of individual trends are movement patterns (e.g., running, walking, climbing stairs), commuting modes (e.g., biking, driving, taking a bus, riding the subway), as well as activities in general (e.g., using an ATM, visiting a store, having a conversation, listening to music, making coffee). Collective trends may be exemplified by (air/noise) pollution levels, e.g., in a neighborhood or, even more aptly, by realtime traffic patterns and transit timings, including significant deviations from recurrent trends and averages, like reroutings due to either scheduled (e.g., closures) or sudden (e.g., pot holes) road alterations.

Such large scale, collective trends monitoring is achievable only if a large community of individuals are willing to share their resources by accepting a certain amount of the processing duties. We distinguish between participatory and opportunistic approaches. In participatory scenarios, individuals are actively involved in contributing by mean of devices or (meta-)data (e.g., taking a picture or reporting a road bump). In opportunistic scenarios, sensing is more automatic and user involvement is minimal (e.g., continuous sampling of geo-localized data).

Given that mobile devices are usually equipped with low voltage batteries, energy saving is certainly one of the main issues in such a context. In fact, users usually have full control on their devices and they are allowed to choose when and how they want to contribute to the MCS application. Usually, some kind of policy is exploited, referring to the energy status in terms of battery charge. Experimental results show that the energy consumption of a mobile device depends on the load the device is subjected to. This is related to the specific tasks and operations the

device is performing at a particular time instant. There are tasks that request a higher energy consumption when compared to others, while low consumption statuses, e.g., suspend, idle, can be forced to extend lifetime.

Here, we consider the simple example of a user willing to contribute to an MCS application by allowing its smart phone to periodically share geo-localized data collected through on-board sensors. The user agrees to send such data only if the remaining battery energy is above 20%. Several works, such as [35], show that the main high-level tasks that affect smart phone energy consumption are: (i) making phone calls, and (ii) browsing the Web, while, in all the other situations, the smart phone is usually maintained in a minimum energy consumption (suspend) mode. For simplicity, let us consider three different usage patterns:

- *generic*—over a single day, about 20% of time is spent making phone calls while about 10% of time is spent in browsing the Web;
- *business*—over a single day, about 40% of time is spent making phone calls while about 10% of time is spent in browsing the Web;
- *student*—over a single day, about 10% of time is spent making phone calls while about 40% of time is spent in browsing the Web.

Finally, let us suppose that the battery duration T_d is approximately equal to 43 h in suspend mode, 10 h while making phone calls, and 5 h while browsing the Web. Our technique allows us to predict the smart phone battery duration and, accordingly, the amount of contribution that the mobile device can provide to the MCS application, according to the different usage patterns.

The state space of the model that can be used to represent our MCS application scenario is depicted in Fig. 6a. States S_S , S_C , and S_B represent the suspend, calling, and web browsing modes respectively. Events e_{S_C} and e_{C_S} represent the switching between suspend and calling modes while events e_{S_B} and e_{B_S} model the switching between suspend and browsing mode. They can be characterized with exponential distributions and their rates must be fixed so that the mean sojourn times in states S_S , S_C , and S_B during a day is set accordingly to the above reported usage patterns. State S_D represents the fully discharged battery state while events e_{D_C} , e_{D_S} , and e_{D_B} represent the battery discharge process.

As described in [17], the linear or nonlinear discharge process of a battery can be modeled, according to the technique described in Sect. 3, through an Erlang CPH with a sufficient number of phases. In our case, we used $n = 100$ phases. The phases of the CPH assume the physical meaning of the battery charge level. The rates of the Erlang distributions are set equal to n/T_d . The discharge time is different in the three states, as described above. However, the battery charge level should be kept while switching among them.

Thanks to the physical meaning of the CPH phases resulting from the codomain fitting algorithm, our technique allows us to compute the probability of having a 80% discharged battery and, as a consequence, the probability for the mobile device to stop contributing to the IoT mobile crowdsensing service. Figure 6b shows this probability in the three usage scenarios considered.

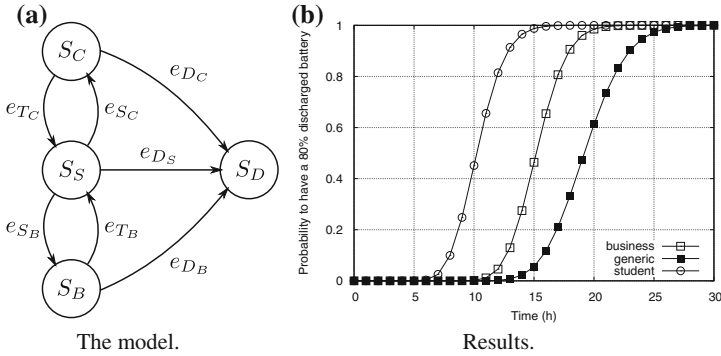


Fig. 6 The energy management in IoT mobile devices example

5.4 Car Travel Planning

As the last example, we consider a car with a fixed quantity of fuel in its tank that should arrive at a defined destination within a certain time. The car has to travel along a path growing at different altitudes. Thus, the road could posses a different slope in different sections. In particular, we consider a road with the elevation graph depicted in Fig. 7a.

The main requirement the driver must meet is to arrive to the stop point by the time of a scheduled appointment. Thus, he/she develops two different driving plans in order to meet this requirement. Two remarks must be considered: (1) even if the driver chooses to travel each path section at a given speed some degree of uncertainty must be considered due to the road condition (holes, traffic congestion, and other unpredictable events potentially occurring during the travel). Thus each section travel time is a r.v. characterized by a mean value (the time planned by the driver) and a nonzero coefficient of variation; (2) the power required to cover a path section depends on the section slope; this means the engine consumes petrol at different rates in each section.

Our method is able to evaluate such problems thanks to its ability to remember an arbitrary quantity following changing operational conditions. In this example, we preserve memory of the level of petrol in the tank, taking into account both the road slope and the changing engine usage.

Based on these considerations, a state model to investigate the car travel plan is depicted in Fig. 7b. Specifically, the model must evaluate the distance the car is able to run and the corresponding time, given an initial quantity of fuel and a travel plan. States S_i , $0 \leq i \leq 5$, represent the car running in section S_i of the path, whereas states S_{E_i} represent the car stopped somewhere in those sections due to empty tank. If the state S_5 is reached the mission is successfully accomplished.

Each plan fixes the expected speeds on each path section and, as a consequence, the required engine power per section is computed by referring to the mechanical

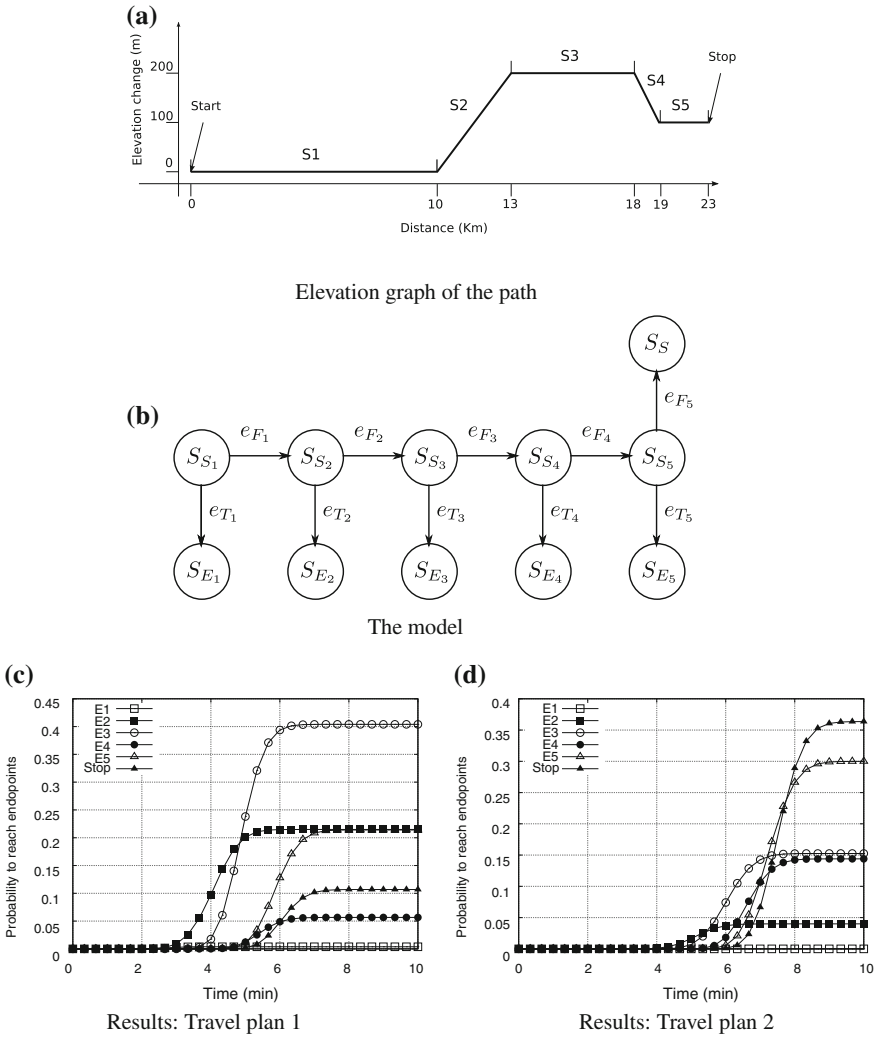


Fig. 7 Car travel planning example

specifications of the car (power vs. engine speed graph, final drive ratio, gear, and tires). In our model, we assume the expected power as the power used along a specific path section.¹ The amount of fuel $a(\tau)$ used during a time interval $[0, \tau]$ depends on both the *engine fuel consumption* c and the power P since $a = c \cdot P \cdot \tau$. Thus, the

¹This is an approximation we made because the speed is not constant but a r.v. Thus, the supplied power is a r.v. too. The approximation introduced is not required by the proposed method and we made it just to simplify the presentation because our purpose is to show the effectiveness of the technique and not to study the mechanical system with high accuracy.

Table 1 Travel plans

Plane n.	Speed (km/h)				
	S1	S2	S3	S4	S5
1	290	80	290	130	290
2	222	80	222	130	222

quantity of fuel in the tank after τ is $a(\tau) = a_0 - c \cdot P \cdot \tau$, where a_0 is the quantity of fuel in the tank when the car starts running.

We use $a(\tau)$ to define events e_{T_i} , as explained in Sect. 3, by taking into account the fact that P changes according to the path sections but the quantity of fuel at changing points must be preserved. The time to run along the path sections S_i (events e_{F_i}) are modeled by Erlang distributions with the mean time depending on the speed provided by the travel plan and a coefficient of variation estimated according to the environmental conditions. The travel plans we considered are reported in Table 1. Moreover, we assumed the car has just 15 l of fuel in its tank. Fuel consumption depends on the power and the speed of the engine. It is important to remark that the fuel consumption may also increase in case of lower speeds. The numerical values are taken from the fuel consumption graph of a BMW M3 E46 car and the engine installed as its standard equipment. For lack of space, we do not show all the mechanical feature graphs used to obtain such values.

Figure 7c, d show the results related to the two travel plans of Table 1. The lesson learned by observing the graphs is that plan 1 does not ensure the car arrives at the end of the path (the *Stop* curve), since it has 0.4 probability to stop at the middle of the overall path and a probability equal to 0.1 to arrive at destination. Plan 2 gives more confidence about the possibility to complete the travel with a probability equal to 0.36. On the other hand, the time required to travel along path 2 is longer than the path 1 one, due to the reduced speed in path sections S1, S3, and S5.

6 Conclusions

In this chapter, we considered a class of systems characterized by different working conditions that alternate and change a system's stochastic behavior. In such systems, the continuity of particular quantities needs to be preserved and this calls for adequate analytical techniques. The chapter summarizes our work on an analytical framework able to represent such phenomena which is based on phase-type distributions and on an ad hoc fitting algorithm relying on codomain discretization. The framework is general enough to model a wide range of systems and, to show its usefulness, four examples have been presented both for ICT and physical systems. Different quantitative analyses have been conducted in order to provide an overview of the available modeling tools.

References

1. Finkelstein MS (1999) Wearing-out of components in a variable environment. *Reliab Eng Syst Saf* 66(3):235–242
2. Sedyakin N (1966) On one physical principle in reliability theory. *Tekhn Kibernetika* (in Russian—Tech Cybern) 3:80–87
3. Ciardo G, Marie R, Sericola B, Trivedi K (1990) Performability analysis using semi-Markov reward processes. *IEEE Trans Comput* 39:1252–1264
4. Limnios N, Oprisan G (2001) Semi-Markov processes and reliability, ser. Statistics for industry and technology. Birkhäuser, Boston, MA, USA
5. Gribaudo M, Bobbio A, Sereno M (2001) Modeling physical quantities in industrial systems using fluid stochastic Petri nets. In: Proceeding of fifth international workshop on performability modeling of computer and communication systems (PMCCS 5), 2001, pp 81–85
6. Ciardo G, Nicol D, Trivedi KS (1999) Discrete-event simulation of fluid stochastic Petri nets. *IEEE Trans Softw Eng* 2:207–217
7. Brokemeyer F, Halstron HS, Jensen A (1948) The life and works of A. K. Erlang. *Trans Danish Acad Tech Sci* 2
8. Neuts MF, Meier KS (1981) On the use of phase type distributions in reliability modelling of systems with two components. *OR Spectr* 2(4):227–234
9. Neuts MF (1981) Matrix-geometric solutions in stochastic models: an algorithmic approach. Johns Hopkins University Press, Baltimore
10. Scarpa M (1999) Non Markovian stochastic Petri nets with concurrent generally distributed transitions. PhD dissertation, University of Turin
11. Pérez-Ocón R, Montoro-Cazorla D (2004) A multiple system governed by a quasi-birth-and-death process. *Reliab Eng Syst Saf* 84(2):187–196
12. Buchholz P, Ciardo G, Donatelli S, Kemper P (2000) Complexity of memory-efficient kronecker operations with applications to the solution of markov models. *Inf J Comput* 12(3):203–222
13. Distefano S, Longo F, Scarpa M (2010) Availability assessment of ha standby redundant clusters. In: SRDS, 2010, pp 265–274
14. Distefano S, Longo F, Scarpa M (2010) Symbolic representation techniques in dynamic reliability evaluation. In: HASE, 2010, pp 45–53
15. Bruneo D, Distefano S, Longo F, Puliafito A, Scarpa M (2010) Reliability assessment of wireless sensor nodes with non-linear battery discharge. In: Wireless Days (WD), 2010 IFIP, Oct 2010, pp 1–5
16. Bruneo D, Longo F, Puliafito A, Scarpa M, Distefano S (2010) Software rejuvenation in the cloud. In: Proceedings of the 5th international ICST conference on simulation tools and techniques, ser. SIMUTOOLS '12. ICST, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, Belgium, pp 8–16 [Online]. <http://dl.acm.org/citation.cfm?id=2263019.2263022>
17. Bruneo D, Distefano S, Longo F, Puliafito A, Scarpa M (2012) Evaluating wireless sensor node longevity through Markovian techniques. *Comput Netw* 56(2):521–532
18. Bruneo D, Distefano S, Longo F, Puliafito A, Scarpa M (2013) Workload-based software rejuvenation in cloud systems. *IEEE Trans Comput* 62(6):1072–1085
19. Distefano S, Bruneo D, Longo F, Scarpa M (2013) Quantitative dependability assessment of distributed systems subject to variable conditions. In: Pathan M, Wei G, Fortino G (eds) Internet and distributed computing systems, ser. Lecture notes in computer science, vol 8223. Springer, Berlin, pp 385–398 [Online]. http://dx.doi.org/10.1007/978-3-642-41428-2_31
20. Distefano S, Longo F, Scarpa M (2013) Investigating mobile crowdsensing application performance. In: Proceedings of the third ACM international symposium on design and analysis of intelligent vehicular networks and applications, ser. DIVANet '13. ACM, New York, NY, USA, pp. 77–84 [Online]. <http://doi.acm.org/10.1145/2512921.2512931>

21. Distefano S, Longo F, Scarpa M, Trivedi K (2014) Non-markovian modeling of a blade center chassis midplane. In: Horvath A, Wolter K (eds) *Computer performance engineering*, ser. *Lecture notes in computer science*, vol 8721. Springer International Publishing, pp. 255–269 [Online]. http://dx.doi.org/10.1007/978-3-319-10885-8_18
22. Longo F, Bruneo D, Distefano S, Scarpa M (2010) Variable operating conditions in distributed systems: modeling and evaluation. *Concurrency and computation: practice and experience*, pp n/a–n/a, 2014 [Online]. <http://dx.doi.org/10.1002/cpe.3419>
23. Longo F, Distefano S, Bruneo D, Scarpa M (2015) Dependability modeling of software defined networking. *Comput Netw* 83(0):280–296 [Online]. <http://www.sciencedirect.com/science/article/pii/S1389128615001139>
24. Cumani A (1985) ESP—a package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In: *PNPM*, pp 144–151
25. Bobbio A, Scarpa M (1998) Kronecker representation of stochastic petri nets with discrete ph distributions. In: *Proceedings of the third IEEE annual international computer performance and dependability symposium (IPDS '98)*
26. Horvath A, Puliafito A, Telek M, Scarpa M (2000) Analysis and evaluation of non-markovian stochastic Petri nets. In: *Tools 2000*. Springer, Schaumburg, IL, USA, March 2000, pp 171–187
27. Buchholz P, Ciardo G, Donatelli S, Kemper P (1997) Complexity of kronecker operations on sparse matrices with applications to the solution of markov models. Technical report
28. Longo F, Scarpa M (2009) Applying symbolic techniques to the representation of non-markovian models with continuous ph distributions. In: *EPEW '09: Proceedings of the 6th European performance engineering workshop on computer performance engineering*. Springer, Berlin, Heidelberg, pp 44–58
29. Longo F, Scarpa M (2015) Two-layer symbolic representation for stochastic models with phase-type distributed events. *Int J Syst Sci* 46(9):1540–1571
30. Franz R, Gradischnig K, Huber M, Stiefel R (1994) Atm-based signaling network topics on reliability and performance. *IEEE J Sel Areas Commun* 12(3):517–525
31. Herrmann C, Lott M, Du Y, Hettich A (1998) A wireless atm lan prototype: test bed implementation and first performance results. In: *Proceedings of the of MTT-S wireless '98 symposium*, pp 145–150
32. Liu H-L, Shooman M (2003) Reliability computation of an ip/atm network with congestion. In: *Annual reliability and maintainability symposium*, pp 581–586
33. Bruneo D (2014) A stochastic model to investigate data center performance and qos in iaas cloud computing systems. *IEEE Trans Parallel Distrib Syst* 25(3):560–569
34. Ganti R, Ye F, Lei H (2011) Mobile crowdsensing: current state and future challenges. *IEEE Commun Mag* 49(11):32–39
35. Carroll A, Heiser G (2010) An analysis of power consumption in a smartphone. In: *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, ser. *USENIX-ATC '10*. USENIX Association, Berkeley, CA, USA, pp 21–21

Fitting Phase-Type Distributions and Markovian Arrival Processes: Algorithms and Tools

Hiroyuki Okamura and Tadashi Dohi

Abstract This chapter provides a comprehensive survey of PH (phase-type) distribution and MAP (Markovian arrival process) fitting. The PH distribution and MAP are widely used in analytical model-based performance evaluation because they can approximate non-Markovian models with arbitrary accuracy as Markovian models. Among a number of past research results on PH/MAP fitting, we present the mathematical definition of the PH distribution and MAP, and summarize the most recent state-of-the-art results on the fitting methods. We also offer an overview of the software tools for PH/MAP fitting.

1 Introduction

Model-based performance evaluation is widely used to estimate system attributes such as dependability and security, as well as quantitative system performance. In model-based performance evaluation, the continuous-time Markov chain (CTMC) plays a central role and is useful to describe the probabilistic behavior of a target system as a state-based stochastic model. Though CTMCs are mathematically tractable, efficient numerical techniques are needed to compute stationary and transient measures for the performance of large-scale systems [12, 71].

When the underlying stochastic model is non-Markovian, i.e., the model includes nonexponential distributions such as the Weibull, Pareto, and lognormal distributions, it is often difficult to apply the analytical approach. Distefano and Trivedi [22] introduced analytical methods for non-Markovian models, phase-type (PH) expansions, Markov renewal programming, and the supplementary variable method. In this chapter, we focus on the PH expansion.

H. Okamura (✉) · T. Dohi

Department of Information Engineering, Graduate School of Engineering,
Hiroshima University, 1-4-1 Kagamiyama, Higashihiroshima 739-8527, Japan
e-mail: okamu@rel.hiroshima-u.ac.jp

T. Dohi

e-mail: dohi@rel.hiroshima-u.ac.jp

In brief, the main idea of the PH expansion is to replace general probability distributions and/or general stochastic point processes by approximate PH distributions [47] and/or Markovian arrival processes (MAPs), respectively. The PH distribution and MAP are one of the widest classes of probability distributions and stochastic point processes. It is well known that both PH distribution and MAP are dense for arbitrary probability distributions with non-negative support and point processes, so they can provide good approximations for general probability distributions and point processes with arbitrary accuracy [5]. Additionally, since the PH distribution and MAP are defined by CTMCs, the resulting model approximated by the PH expansion reduces to a relatively simple CTMC. Hence, analytical and numerical approaches for CTMCs are directly applied to analyze the approximation model. However, in practice, there is a serious problem of how to determine PH/MAP parameters to guarantee a good approximation. The aim of PH/MAP fitting is to estimate model parameters from empirical data or probability density functions. Thus, these techniques possess practical significance to the model-based performance evaluation of large-scale complex systems.

In this chapter, we provide a comprehensive survey of PH/MAP fitting. In general, although PH/MAP fitting can be classified as moment-based and likelihood-based approaches, we concern ourselves with several likelihood-based algorithms proposed by [54, 56] as joint works with Prof. Kishor S. Trivedi. These are useful to understand the fundamental ideas of PH/MAP fitting from the statistical point of view. In addition, we offer an overview of the existing PH/MAP fitting tools. This information is valuable for all researchers who study model-based performance evaluation.

2 PH Fitting

The PH distribution is defined as the distribution of required time to reach the absorbing state in a time-homogeneous CTMC with an absorbing state. Without loss of generality, we consider an absorbing CTMC on the finite state space $\{0, 1, 2, \dots, m\}$ with the following infinitesimal generator:

$$Q_{\text{PH}} = \begin{pmatrix} 0 & \mathbf{0} \\ \boldsymbol{\tau} & T \end{pmatrix}, \quad (1)$$

where T is an infinitesimal generator between transient states and $\boldsymbol{\tau}$ is a column vector of transition rates from transient states to the absorbing state. By using the column vector $\mathbf{1}$ whose all elements are 1, the vector $\boldsymbol{\tau}$ is given by $\boldsymbol{\tau} = -T\mathbf{1}$. The above absorbing CTMC is called a *phase process*, denoted by J_x , and the state of the phase process is referred to as a *phase* in this chapter. The PH random variable is defined by the first passage time to the absorbing state (state 0) on the phase process J_x , i.e.,

$$X = \inf\{x \geq 0; J_x = 0\}. \quad (2)$$

Suppose that the initial phase J_0 is determined by a probability (row) vector α over the transient states $\{1, \dots, m\}$. The probability density function (p.d.f.) and cumulative distribution function (c.d.f.) of X can be written by

$$g(x) = \alpha \exp(\mathbf{T}x)\boldsymbol{\tau}, \quad 0 \leq x < \infty, \quad (3)$$

$$G(x) = 1 - \alpha \exp(\mathbf{T}x)\mathbf{1}, \quad 0 \leq x < \infty, \quad (4)$$

respectively, where $(\alpha, \mathbf{T}, \boldsymbol{\tau})$ are the PH parameters.¹ Also, α_i , τ_i , and $\lambda_{i,j}$ denote the i -th entries of α and $\boldsymbol{\tau}$, and the (i, j) -entry of \mathbf{T} , respectively.

The PH distribution can approximate any kind of probability distribution defined on the non-negative support [5]. The purpose of PH fitting is to determine the PH parameters $(\alpha, \mathbf{T}, \boldsymbol{\tau})$ to approximate the original distribution with the fitted PH distribution. In general, the accuracy of approximation depends on the number of phases and the phase structure which determines the structure of state transitions of the underlying phase process. For example, if the number of phases is 1, the corresponding PH distribution reduces to an exponential distribution. Needless to say, the exponential distribution does not exhibit much flexibility when fitting distributions. Hence, we encounter the following three research questions on PH fitting:

- (i) What is the appropriate phase structure?
- (ii) How many phases are required?
- (iii) How should the parameters $(\alpha, \mathbf{T}, \boldsymbol{\tau})$ be estimated?

For research question (i), three kinds of phase structures were typically assumed in the past literature. The first is a general PH (GPH) distribution, which does not have any special phase structure, i.e., α and \mathbf{T} are allowed to be full vector and matrix, respectively. The second is an acyclic PH (APH) distribution whose underlying phase process is restricted to an acyclic CTMC. In this case, \mathbf{T} is expressed as an upper or lower triangular matrix. Cumani [19] proved that any APH distribution can be represented in three standard structures called canonical forms. Thus, PH fitting with the APH distribution essentially reduces to PH fitting with canonical forms. The third phase structure is a finite mixture of exponential/Erlang distributions. A finite mixture of probability distributions is represented by a weighted sum of the finite number of distributions. When the mixed components are PH distributions, the mixture also becomes a PH distribution. In other words, the parameter estimation of the mixture of exponential/Erlang distributions is equivalent to PH fitting with a special phase structure. One of the advantages of using the mixture of exponential/Erlang distributions avoids the computation of a matrix exponential.

Research question (ii) is still an open and unresolved problem of PH fitting. Several approaches have been proposed in the past literature. The most intuitive method is to determine the number of phases from the viewpoint of moments. That is, the number of phases is determined by the minimum number of phases so that

¹The PH parameters can be defined by (α, \mathbf{T}) . In this chapter, since the diagonals of \mathbf{T} are determined after estimating nondiagonals of \mathbf{T} and $\boldsymbol{\tau}$, the PH parameters include $\boldsymbol{\tau}$.

the resulting moments (mean, variance, skewness, etc.) of the PH expansion coincide with the theoretical (given) moments. A more sophisticated approach is based on the Laplace–Stieltjes (LS) transform. The LS transform is a variant of Laplace transform. O’Cinneide [48–51] discussed the characteristics of the PH distribution and provided the lower bounds on the order of the PH distribution by means of poles of the LS transform. He and Zhang [28] derived the phase reduction algorithm to find the minimal representation of Coxian distribution based on O’Cinneide’s works. The theoretically valid approach is to use information criteria such as AIC (Akaike information criterion) [1] and BIC (Bayesian information criterion) [66] from the statistical point of view. The information criterion represents a kind of distance between the true and estimated models, and is used in many applications of the statistical model selection problem. Briefly speaking, the information criterion imposes a penalty term on models that contain many parameters. Thus, information criteria can also identify the minimum number of phases.

For research question (iii), there are many variations proposed in the past literature, and these methods can be classified into three types. The first method is moment matching (MM). The fundamental concept of MM is to find PH parameters so that the moments can fit the given moments. In fact, many authors focused on only the first two or three moments to determine PH parameters. The second method is maximum likelihood estimation (MLE). The MLE is a commonly used technique to estimate model parameters by maximizing the likelihood so that the observed data are drawn from the model. When PH parameters are estimated from a theoretical p.d.f., MLE is essentially equivalent to minimizing cross entropy. The advantage of MLE is that it can estimate parameters from any type of data and possesses several statistically rich properties such as asymptotic normality² when fitting PH distributions. On the other hand, since the computational cost of MLE is relatively high, a challenging issue is to identify methods to reduce the computational cost. The EM (expectation-maximization) algorithm [20, 75] is frequently used for PH fitting.

The third method is Bayes estimation. In the Bayesian paradigm, it is assumed that unknown model parameters are random variables, i.e., all model parameters are randomly distributed. By observing field data, the corresponding probability distribution of model parameters is updated according to Bayes theorem. The updated probability distribution of model parameters is called the posterior distribution, and the probability distribution of model parameters before the update is called the prior distribution. The main feature of Bayes estimation is to compute the posterior distribution of parameters. The Bayes estimation enables us to evaluate not only interval estimation of parameters but also the predictive distribution in many cases. However, the computation of a posterior distribution is often more expensive than MM and MLE. Thus, approximation methods are sometimes used to compute the posterior distribution.

The main concern for the earliest PH fitting efforts was MM with special structure such as a mixture of Erlang distributions and a 2-phase Coxian distribution based

²When the number of samples increases, the log-likelihood function can be approximated by a multivariate normal distribution.

on the first two moments in application of Markov model [43]. Altioek [3] presented a method to approximate general distributions with PH distributions via the LS transform of the mixture of exponential distributions. van der Heijden [72] gave a simple formula for MM of a Coxian distribution with the first three moments. Aldous and Shepp [2] showed that the number of phases is easily determined by the coefficient variation for the Erlang distribution. Johnson and Taaffe [36–39] and Johnson [35] obtained several results on PH fitting for the mixture of Erlang distributions with nonlinear programming techniques in terms of the difference of p.d.f. as well as moments. Feldmann and Whitte [26] discussed an algorithm to estimate a mixture of exponentials to fit long-tailed data. In the context of MM for APH distribution, Telek and Heindl [67] presented a method for the 2-phase APH distribution. Osogami and Harchol-Balter [60] and Bobbio et al. [10] found the MM for APH distributions with the first three moments. Specifically, their methods addressed the problem of determining the number of phases, where the number of phases is given by a minimum number of phases so that the first three moments of the estimated PH distribution match the given first three moments. van de Liefvoort [73] and Telek and Horvath [68] extended the PH distribution to the matrix-exponential distribution, and discussed MM with high-order moments.

Here, we remark on the computational risk in the use of MM. There is no doubt that MM works well for PH approximation when the theoretical moments are known. However, if MM is applied to PH fitting with statistical data, we should consider the estimation errors for the moments. In general, an estimate of high-order moments is rather sensitive to errors contained in a random sample. Thus, it is difficult to obtain accurate estimates of high-order moments when the number of samples is small. In addition, there is the situation where we cannot even calculate the first moment when some data are missing. In such cases, MLE or other statistical approaches are more appropriate.

In the context of MLE, Bobbio and Cumani [9] presented the MLE for the APH distribution by applying linear programming iteratively to solve the maximization problem for the log-likelihood function which is a logarithm of probability mass or density function for observed data. Also, Bobbio and Telek [11] carried out experiments on MLE-based PH fitting for the APH distribution. Asmussen et al. [6] proposed an EM algorithm for the GPH distribution. Olsson [59] extended the EM algorithm to the case with censored data. Okamura et al. [55] improved Asmussen's EM algorithm with respect to time complexity of the number of phases. Also, they proposed an algorithm to estimate the APH distribution with EM-based PH fitting. Moreover, in [56], the EM-based PH fitting is extended to handle the cases of grouped, truncated, and missing data. On the other hand, for PH fitting with a mixture of Erlang distributions (hyper-Erlang distribution), Thummler et al. [69] and Panchenko and Thummler [61] developed EM algorithms which can determine the phase structure of a hyper-Erlang distribution. Reinecke et al. [63] combined data clustering with PH fitting.

For Bayes estimation, Bladt et al. [8] proposed an MCMC (Markov chain Monte Carlo) algorithm³ for the GPH distribution with a Markov jump process to approximate the posterior distributions. Watanabe et al. [74] considered an improved MCMC algorithm by using the uniformization technique. Yamaguchi et al. [76] and Okamura et al. [58] proposed the variational Bayes algorithms for a mixture of Erlang distributions and the GPH distribution.

3 PH Fitting Algorithms

3.1 Data for PH Fitting

In this section, we describe a detailed PH fitting algorithm proposed in [56]. The algorithm is based on MLE with the EM principle and can be applied to estimate GPH parameters with grouped, truncated, and missing data. The grouped data consists of several time intervals and the number of events that occur in the time intervals. The time points that generate time intervals are referred to as *break points* in this chapter. Table 1a presents an example of grouped, truncated, and missing data with break points 0, 10, 20, . . . , 100. In this table, the values in intervals [10, 20] and [40, 50] are missing and denoted as NA (not available). Furthermore, the bottom row indicates that 5 samples are truncated at 100. Similarly, Table 1b gives another example. In this case, several samples are truncated at 100, but we do not know the exact number of truncated samples. Since such grouped, truncated, and missing data frequently appear in practical situations, the PH fitting approach presented here is one of the most appropriate algorithms.

3.2 EM-Step Formulas

The fitting algorithm is based on the EM algorithm, which is an iterative method to compute MLE when observed data are incomplete [20, 75]. Define \mathcal{D} and \mathcal{U} as observable and unobservable data, respectively. The EM algorithm consists of two steps, namely the E-step and M-step. In the E-step, we compute the expected log-likelihood function (LLF) with respect to the unobserved data when observed data are given. The M-step finds the parameters that maximize the expected LLF. The LLF monotonically increases by updating the parameters in each EM-step. These EM-steps are executed until the LLF or parameters converge. Let $Q(\theta|\theta')$ denote the expected LLF of the parameter vector θ provided that the provisional (initial)

³The MCMC is a sample-based approximation method for posterior distributions.

Table 1 Examples of grouped, truncated, and missing data for PH fitting (break points: 0, 10, 20, . . . , 90, 100)

Data (a)	
Time interval	Number of events
[0, 10]	1
[10, 20]	NA
[20, 30]	5
[30, 40]	11
[40, 50]	NA
[50, 60]	29
[60, 70]	9
[70, 80]	12
[80, 90]	4
[90, 100]	0
[100, ∞]	5
Data (b)	
Time interval	Number of events
[0, 10]	1
[10, 20]	NA
[20, 30]	5
[30, 40]	11
[40, 50]	NA
[50, 60]	29
[60, 70]	9
[70, 80]	12
[80, 90]	4
[90, 100]	0
[100, ∞]	NA

parameters θ' are given. The E-step computes $Q(\theta|\theta')$ by using θ' . In the M-step, we update the parameter vector by solving

$$\theta \leftarrow \underset{\theta}{\operatorname{argmax}} Q(\theta|\theta'). \tag{5}$$

In the EM algorithm, the E- and M-steps are successively repeated until the parameter vector converges.

Here, we consider grouped data with break points $0 = x_0 < x_1 < \dots < x_K$, i.e., the data consist of the number of samples for $K + 1$ intervals $[0, x_1), [x_1, x_2), \dots, [x_K, \infty)$. For the sake of notational convenience, let $X^{(k,l)}$ be the l -th sample in the k -th interval. In addition, we assume two kinds of intervals: observable and unobservable intervals. Let \mathcal{I} and $\overline{\mathcal{I}}$ be the sets of indexes of observable and unobservable intervals. Note that \mathcal{I} and $\overline{\mathcal{I}}$ are disjoint, and that $\mathcal{I} \cup \overline{\mathcal{I}}$ gives all the indexes, i.e., $\mathcal{I} \cup \overline{\mathcal{I}} = \{1, 2, \dots, K + 1\}$. We can count the numbers of samples in only the

observable intervals, but the numbers of samples in the unobservable intervals are unknown, i.e., the data are missing in the unobservable intervals.

Let $\mathcal{D} = \{n_k\}_{k \in \mathcal{J}}$ be the numbers of samples in observable intervals. Define the following unobserved variables:

- U_k : the number of samples in the k -th unobserved interval $[x_{k-1}, x_k)$, $k \in \overline{\mathcal{J}}$.
- $B_i^{(k,l)}$: an indicator random variable for the event that the phase process of $X^{(k,l)}$ begins with phase i .
- $Z_i^{(k,l)}$: the total sojourn time of phase i on the phase process of $X^{(k,l)}$.
- $Y_i^{(k,l)}$: an indicator random variable for the event that the phase process of $X^{(k,l)}$ is i just before the absorption.
- $M_{i,j}^{(k,l)}$: the total number of phase transitions from i to j on the phase process of $X^{(k,l)}$.

Since the estimates of CTMC parameters are given by frequencies and sojourn time for each state when the CTMC process is completely observed, the M-step formulas of the PH parameters are obtained from Eq. (5):

$$\alpha_i \leftarrow \frac{\sum_{k \in \mathcal{J}} \mathbb{E} \left[\sum_{l=1}^{n_k} B_i^{(k,l)} \mid \mathcal{D} \right] + \sum_{k \in \overline{\mathcal{J}}} \mathbb{E} \left[\sum_{l=1}^{U_k} B_i^{(k,l)} \mid \mathcal{D} \right]}{\sum_{k \in \mathcal{J}} n_k + \sum_{k \in \overline{\mathcal{J}}} \mathbb{E}[U_k \mid \mathcal{D}]}, \quad (6)$$

$$\tau_i \leftarrow \frac{\sum_{k \in \mathcal{J}} \mathbb{E} \left[\sum_{l=1}^{n_k} Y_i^{(k,l)} \mid \mathcal{D} \right] + \sum_{k \in \overline{\mathcal{J}}} \mathbb{E} \left[\sum_{l=1}^{U_k} Y_i^{(k,l)} \mid \mathcal{D} \right]}{\sum_{k \in \mathcal{J}} \mathbb{E} \left[\sum_{l=1}^{n_k} Z_i^{(k,l)} \mid \mathcal{D} \right] + \sum_{k \in \overline{\mathcal{J}}} \mathbb{E} \left[\sum_{l=1}^{U_k} Z_i^{(k,l)} \mid \mathcal{D} \right]}, \quad (7)$$

$$\lambda_{i,j} \leftarrow \frac{\sum_{k \in \mathcal{J}} \mathbb{E} \left[\sum_{l=1}^{n_k} M_{i,j}^{(k,l)} \mid \mathcal{D} \right] + \sum_{k \in \overline{\mathcal{J}}} \mathbb{E} \left[\sum_{l=1}^{U_k} M_{i,j}^{(k,l)} \mid \mathcal{D} \right]}{\sum_{k \in \mathcal{J}} \mathbb{E} \left[\sum_{l=1}^{n_k} Z_i^{(k,l)} \mid \mathcal{D} \right] + \sum_{k \in \overline{\mathcal{J}}} \mathbb{E} \left[\sum_{l=1}^{U_k} Z_i^{(k,l)} \mid \mathcal{D} \right]}. \quad (8)$$

Note that all the expected values are computed from the provisional PH parameters $\theta = (\alpha, \mathbf{T}, \tau)$, which are given as a result of the previous M-step. At the first EM-step, they are provided as initial guesses of PH parameters.

Next, we derive the analytical forms of the expected values in Eqs. (6)–(8). Since it is known that the posterior distribution of unobserved samples $\mathbf{U} = \{U_k\}_{k \in \overline{\mathcal{J}}}$ becomes a negative multinomial distribution [44], the expected value of U_k , $k \in \overline{\mathcal{J}}$ is given by

$$\mathbb{E}[U_k \mid \mathcal{D}] = \frac{N \int_{x_{k-1}}^{x_k} g(x) dx}{\sum_{i \in \mathcal{J}} \int_{x_{i-1}}^{x_i} g(x) dx}, \quad k \in \overline{\mathcal{J}}, \quad (9)$$

where $N = \sum_{k \in \mathcal{J}} n_k$ and $g(x)$ is the p.d.f. of the PH distribution.

Define the following row and column vectors for a PH random variable X and its phase process J_x :

$$[\mathbf{f}(x)]_i = P(J_x = i), \quad [\mathbf{b}(x)]_i = P(X = x \mid J_0 = i), \quad (10)$$

where $i \neq 0$, $[\cdot]_i$ is the i -th element of the vector. For the sake of notational convenience, although X is a continuous random variable, $P(X = x)$ indicates the p.d.f. of X .

Since $X^{(k,l)}$ are IID (independent and identically distributed) samples, the expected value of $B_i^{(k,l)}$ in the observable interval is given by

$$\begin{aligned} \mathbb{E} \left[\sum_{l=1}^{n_k} B_i^{(k,l)} \middle| \mathcal{D} \right] &= n_k \mathbb{E}[B_i | x_{k-1} < X \leq x_k] = \frac{n_k P(J_0 = i, x_{k-1} < X \leq x_k)}{P(x_{k-1} < X \leq x_k)} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} P(J_0 = i) P(X = x | J_0 = i) dx}{\int_{x_{k-1}}^{x_k} P(X = x) dx} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} \alpha_i[\mathbf{b}(x)]_i dx}{\int_{x_{k-1}}^{x_k} g(x) dx}. \end{aligned} \quad (11)$$

In the case of an unobservable interval, by using the posterior distribution of U_k , the expected value becomes

$$\begin{aligned} \mathbb{E} \left[\sum_{l=1}^{U_k} B_i^{(k,l)} \middle| \mathcal{D} \right] &= \sum_{n=0}^{\infty} n \mathbb{E}[B_i | U_k = n, x_{k-1} < X \leq x_k] P(U_k = n | \mathcal{D}) \\ &= \sum_{n=0}^{\infty} \frac{n \int_{x_{k-1}}^{x_k} \alpha_i[\mathbf{b}(x)]_i dx}{\int_{x_{k-1}}^{x_k} g(x) dx} P(U_k = n | \mathcal{D}) \\ &= \frac{\mathbb{E}[U_k | \mathcal{D}] \int_{x_{k-1}}^{x_k} \alpha_i[\mathbf{b}(x)]_i dx}{\int_{x_{k-1}}^{x_k} g(x) dx} \\ &= \frac{N \int_{x_{k-1}}^{x_k} \alpha_i[\mathbf{b}(x)]_i dx}{\sum_{i \in \mathcal{I}} \int_{x_{i-1}}^{x_i} g(x) dx}. \end{aligned} \quad (12)$$

Similarly, we have

$$\begin{aligned} \mathbb{E} \left[\sum_{l=1}^{n_k} Y_i^{(k,l)} \middle| \mathcal{D} \right] &= \frac{n_k \int_{x_{k-1}}^{x_k} P(X = x, J_x^- = i) dx}{P(x_{k-1} < X \leq x_k)} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} P(J_x^- = i) P(J_x = 0 | J_x^- = i) dx}{P(x_{k-1} < X \leq x_k)} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} [\mathbf{f}(x)]_i \tau_i dx}{\int_{x_{k-1}}^{x_k} g(x) dx}, \end{aligned} \quad (13)$$

$$\mathbb{E} \left[\sum_{l=1}^{U_k} Y_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{N \int_{x_{k-1}}^{x_k} [\mathbf{f}(x)]_i \tau_i dx}{\sum_{i \in \mathcal{I}} \int_{x_{i-1}}^{x_i} g(x) dx}, \quad (14)$$

where $J_x^- = \lim_{x' \rightarrow 0^+} J_{x-x'}$.

For $Z_i^{(k,l)}$ and $M_{i,j}^{(k,l)}$, the expected values in the observable interval are

$$\begin{aligned} \mathbb{E} \left[\sum_{l=1}^{n_k} Z_i^{(k,l)} \middle| \mathcal{D} \right] &= \frac{n_k \int_{x_{k-1}}^{x_k} \int_0^x P(X = x, J_u = i) du dx}{P(x_{k-1} < X \leq x_k)} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} \int_0^x P(J_u = i) P(X = x, |J_u = i) du dx}{P(x_{k-1} < X \leq x_k)} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} \int_0^x [\mathbf{f}(u)]_i [\mathbf{b}(x - u)]_i du dx}{\int_{x_{k-1}}^{x_k} g(x) dx}, \end{aligned} \quad (15)$$

$$\begin{aligned} \mathbb{E} \left[\sum_{l=1}^{n_k} M_{i,j}^{(k,l)} \middle| \mathcal{D} \right] &= \frac{n_k \int_{x_{k-1}}^{x_k} \int_0^x P(X = x, J_u^- = i, J_u = j) du dx}{P(x_{k-1} < X \leq x_k)} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} \int_0^x P(J_u^- = i) P(J_u = j | J_u^- = i) P(X = x, |J_u = j) du dx}{P(x_{k-1} < X \leq x_k)} \\ &= \frac{n_k \int_{x_{k-1}}^{x_k} \int_0^x [\mathbf{f}(u)]_i \lambda_{i,j} [\mathbf{b}(x - u)]_j du dx}{\int_{x_{k-1}}^{x_k} g(x) dx}. \end{aligned} \quad (16)$$

Also, the expected values in the unobservable interval become

$$\mathbb{E} \left[\sum_{l=1}^{U_k} Z_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{N \int_{x_{k-1}}^{x_k} \int_0^x [\mathbf{f}(u)]_i [\mathbf{b}(x - u)]_i du dx}{\sum_{i \in \mathcal{I}} \int_{x_{i-1}}^{x_i} g(x) dx}, \quad (17)$$

$$\mathbb{E} \left[\sum_{l=1}^{U_k} M_{i,j}^{(k,l)} \middle| \mathcal{D} \right] = \frac{N \int_{x_{k-1}}^{x_k} \int_0^x [\mathbf{f}(u)]_i \lambda_{i,j} [\mathbf{b}(x - u)]_j du dx}{\sum_{i \in \mathcal{I}} \int_{x_{i-1}}^{x_i} g(x) dx}. \quad (18)$$

3.3 Computation Algorithms

To compute the expected values, we define the following vectors and matrices based on $\mathbf{f}(x) = \boldsymbol{\alpha} \exp(\mathbf{T}x)$ and $\mathbf{b}(x) = \exp(\mathbf{T}x)\boldsymbol{\tau}$:

$$\bar{f}_k = \int_{x_k}^{\infty} f(x)dx = \boldsymbol{\alpha}(-\mathbf{T})^{-1} \exp(\mathbf{T}x_k), \quad (19)$$

$$\tilde{f}_k = \int_{x_{k-1}}^{x_k} f(x)dx = \bar{f}_{k-1} - \bar{f}_k, \quad (20)$$

$$\bar{b}_k = \int_{x_k}^{\infty} \mathbf{b}(x)dx = \exp(\mathbf{T}x_k)\mathbf{1}, \quad (21)$$

$$\tilde{b}_k = \int_{x_{k-1}}^{x_k} \mathbf{b}(x)dx = \bar{b}_{k-1} - \bar{b}_k. \quad (22)$$

Moreover, for the sake of mathematical convenience, we introduce the following matrices to compute the convolution integral of $f(u)$ and $\mathbf{b}(u)$:

$$\begin{aligned} \bar{\mathbf{H}}_k &= \int_{x_k}^{\infty} \int_0^u \mathbf{b}(x-u) f(u) du dx = \int_0^{x_k} \exp(\mathbf{T}(x_k-u)) \mathbf{1} \boldsymbol{\alpha} \exp(\mathbf{T}u) du + \mathbf{1} \bar{f}_k \\ &= \bar{\mathbf{Y}} + \mathbf{1} \bar{f}_k, \end{aligned} \quad (23)$$

$$\tilde{\mathbf{H}}_k = \int_{x_{k-1}}^{x_k} \int_0^u \mathbf{b}(x-u) f(u) du dx = \bar{\mathbf{Y}}_{k-1} - \bar{\mathbf{Y}}_k + \mathbf{1} \tilde{f}_k, \quad (24)$$

where $\bar{\mathbf{Y}}_k = \int_0^{x_k} \exp(\mathbf{T}(x_k-u)) \mathbf{1} \boldsymbol{\alpha} \exp(\mathbf{T}u) du$. By using the above vectors and matrices, the expected values are rewritten as

$$\mathbb{E}[U_k | \mathcal{D}] = \frac{N \boldsymbol{\alpha} \tilde{\mathbf{b}}_k}{\sum_{i \in \mathcal{J}} \boldsymbol{\alpha} \tilde{\mathbf{b}}_i}, \quad (25)$$

$$\mathbb{E} \left[\sum_{l=1}^{n_k} B_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{n_k \alpha_i [\tilde{\mathbf{b}}_k]_i}{\boldsymbol{\alpha} \tilde{\mathbf{b}}_k}, \quad \mathbb{E} \left[\sum_{l=1}^{U_k} B_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{N \alpha_i [\tilde{\mathbf{b}}_k]_i}{\sum_{u \in \mathcal{J}} \boldsymbol{\alpha} \tilde{\mathbf{b}}_u}, \quad (26)$$

$$\mathbb{E} \left[\sum_{l=1}^{n_k} Y_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{n_k [\tilde{\mathbf{f}}_k]_i \tau_i}{\boldsymbol{\alpha} \tilde{\mathbf{b}}_k}, \quad \mathbb{E} \left[\sum_{l=1}^{U_k} Y_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{N [\tilde{\mathbf{f}}_k]_i \tau_i}{\sum_{u \in \mathcal{J}} \boldsymbol{\alpha} \tilde{\mathbf{b}}_u}, \quad (27)$$

$$\mathbb{E} \left[\sum_{l=1}^{n_k} Z_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{n_k [\tilde{\mathbf{H}}_k]_{i,i}}{\boldsymbol{\alpha} \tilde{\mathbf{b}}_k}, \quad \mathbb{E} \left[\sum_{l=1}^{U_k} Z_i^{(k,l)} \middle| \mathcal{D} \right] = \frac{N [\tilde{\mathbf{H}}_k]_{i,i}}{\sum_{u \in \mathcal{J}} \boldsymbol{\alpha} \tilde{\mathbf{b}}_u}, \quad (28)$$

$$\mathbb{E} \left[\sum_{l=1}^{n_k} M_{i,j}^{(k,l)} \middle| \mathcal{D} \right] = \frac{n_k \lambda_{i,j} [\tilde{\mathbf{H}}_k]_{j,i}}{\alpha \tilde{\mathbf{b}}_k}, \quad \mathbb{E} \left[\sum_{l=1}^{U_k} M_{i,j}^{(k,l)} \middle| \mathcal{D} \right] = \frac{N \lambda_{i,j} [\tilde{\mathbf{H}}_k]_{j,i}}{\sum_{u \in \mathcal{S}} \alpha \tilde{\mathbf{b}}_u}, \quad (29)$$

where $[\cdot]_{i,j}$ is the (i, j) -entry of matrix. Note that $\tilde{\mathbf{f}}_k, \tilde{\mathbf{b}}_k, \tilde{\mathbf{H}}_k$ are replaced by $\bar{\mathbf{f}}_k, \bar{\mathbf{b}}_k,$ and $\bar{\mathbf{H}}_k$ respectively at $k = K + 1$ because t_{K+1} goes to infinity.

Much computation time of the EM algorithm for the PH distribution is spent on the calculation of convolution integral of matrix exponential. In [52, 55], an effective computation algorithm for the convolution integral of the matrix exponential was proposed based on the uniformization. Let $\Upsilon(t; \mathbf{v}_1, \mathbf{v}_2)$ be the convolution integral of matrix exponential;

$$\Upsilon(x; \mathbf{v}_1, \mathbf{v}_2) = \int_0^x \exp(\mathbf{T}(x-u)) \mathbf{v}_1 \mathbf{v}_2 \exp(\mathbf{T}u) du, \quad (30)$$

where \mathbf{v}_1 and \mathbf{v}_2 are arbitrary column and row vectors, respectively. Algorithm 1 presents a pseudo code to compute $\Upsilon(x; \mathbf{v}_1, \mathbf{v}_2)$. In this algorithm, ε is an error tolerance for the computation of the Poisson probability. The function $\text{PoiBound}(\mu, \varepsilon)$ gives the first point where the c.d.f. of Poisson distribution with mean μ becomes more than $1 - \varepsilon$. The function $\text{PoiProb}(n, \mu)$ is the Poisson probability mass function with mean parameter μ , i.e., $e^{-\mu} \mu^n / n!$. Also, \mathbf{I} denotes the identity matrix. The time complexity of Algorithm 1 is proportional to a square of the number of phases. In particular, when \mathbf{T} is a sparse matrix, the time complexity is proportional to the number of nonzero elements of \mathbf{T} .

Algorithm 1 Uniformization-based convolution integral of matrix exponential [55, 56].

```

function convint( $x, \mathbf{v}_1, \mathbf{v}_2$ ) : return value  $\Upsilon$ 
 $r \leftarrow \max(\text{abs}(\text{diag}(\mathbf{T})))$ ;  $\mathbf{P} \leftarrow \mathbf{I} + \mathbf{T}/r$ ;  $R \leftarrow \text{PoiBound}(rx, \varepsilon)$ 
 $\mathbf{v}_1(0) \leftarrow \mathbf{v}_1$ 
for  $i = 1 : R$  do
     $\mathbf{v}_1(i) \leftarrow \mathbf{P} \mathbf{v}_1(i-1)$ 
end for
 $\mathbf{v}_2(R) \leftarrow \text{PoiProb}(R+1, rx) \mathbf{v}_2$ 
for  $i = R-1 : 0$  do
     $\mathbf{v}_2(i) \leftarrow \mathbf{v}_2(i+1) \mathbf{P} + \text{PoiProb}(i+1, rx) \mathbf{v}_2$ 
end for
 $\Upsilon \leftarrow \mathbf{0}$ 
for  $i = 0 : R$  do
     $\Upsilon \leftarrow \Upsilon + \mathbf{v}_1(i) \mathbf{v}_2(i) / r$ 
end for
end function

```

Applying the algorithm into the computation of expected values, we define

$$\mathbf{H} = \sum_{k=1}^{K+1} w_k \tilde{\mathbf{H}}_k, \quad (31)$$

where

$$w_k = \begin{cases} n_k / \boldsymbol{\pi} \tilde{\mathbf{b}}_k, & k \in \mathcal{I}, \\ N / \sum_{u \in \overline{\mathcal{I}}} \boldsymbol{\pi} \tilde{\mathbf{b}}_u, & k \in \overline{\mathcal{I}}. \end{cases} \quad (32)$$

Since $\tilde{\mathbf{H}} = \overline{\mathbf{H}}_{k-1} - \overline{\mathbf{H}}_k$ and $\overline{\mathbf{H}}_k = \boldsymbol{\Upsilon}(x_k; \mathbf{1}, \boldsymbol{\alpha}) + \mathbf{1} \tilde{\mathbf{f}}_k$, we get

$$\mathbf{H} = \sum_{k=1}^{K+1} w_k \mathbf{1} \tilde{\mathbf{f}}_k + \sum_{k=1}^K \boldsymbol{\Upsilon}(\Delta x_k; \overline{\mathbf{b}}_{k-1}, \mathbf{c}_k), \quad (33)$$

where $\Delta x_k = x_k - x_{k-1}$ and

$$\mathbf{c}_k = \sum_{l=k}^K (w_{l+1} - w_l) \boldsymbol{\alpha} \exp(\mathbf{T}(t_l - t_k)). \quad (34)$$

Algorithm 2 shows our E-step procedure for PH distributions under grouped, truncated, and missing data according to the above formulas. In this algorithm, the function `convint`($x, \mathbf{v}_1, \mathbf{v}_2$) denotes the convolution integral of the matrix exponential presented in Algorithm 1. By using the results of Algorithm 2, the parameters are updated with $\alpha_i \leftarrow \alpha_i [\mathbf{B}]_i / (N + U)$, $\tau_i \leftarrow \tau_i [\mathbf{Y}]_i / [\mathbf{H}]_{i,i}$ and $\lambda_{i,j} \leftarrow \lambda_{i,j} [\mathbf{H}]_{j,i} / [\mathbf{H}]_{i,i}$.

3.4 M-Step of Canonical Form

As mentioned before, any APH distribution can be transformed to three canonical forms [19]. Thus, PH fitting for canonical forms is of practical importance. For instance, one of the canonical forms (CF1: canonical form 1) is given by the following bidiagonal infinitesimal generator:

$$\boldsymbol{\alpha} = (\alpha_1 \dots \alpha_m), \quad \mathbf{T} = \begin{pmatrix} -\lambda_1 & \lambda_1 & & & \\ & -\lambda_2 & \lambda_2 & & \\ & & \ddots & \ddots & \\ & & & & -\lambda_m \end{pmatrix}, \quad \boldsymbol{\tau} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \lambda_m \end{pmatrix}, \quad (35)$$

where the restriction $\lambda_1 \leq \dots \leq \lambda_m$ holds.

Algorithm 2 E-step procedure for PH distributions with grouped and truncated data [56].

```

B  $\leftarrow$  0; Y  $\leftarrow$  0; H  $\leftarrow$  O
 $\bar{f}_0 \leftarrow \alpha(-T)^{-1}$ ;  $\bar{b}_0 \leftarrow \mathbf{1}$ ; N  $\leftarrow$  0; U  $\leftarrow$  0
for  $k = 1 : K$  do
   $\bar{f}_k \leftarrow \bar{f}_{k-1} \exp(T \Delta x_k)$ ;  $\tilde{f}_k \leftarrow \bar{f}_{k-1} - \bar{f}_k$ ;  $\bar{b}_k \leftarrow \exp(T \Delta x_k) \bar{b}_{k-1}$ ;  $\tilde{b}_k \leftarrow \bar{b}_{k-1} - \bar{b}_k$ 
  if  $k \in \mathcal{J}$  then
    N  $\leftarrow$  N +  $n_k$ ; U  $\leftarrow$  U +  $\alpha \tilde{b}_k$ 
  end if
end for
if  $K + 1 \in \mathcal{J}$  then
  N  $\leftarrow$  N +  $n_{K+1}$ ; U  $\leftarrow$  U +  $\alpha \bar{b}_K$ 
end if
for  $k = 1 : K$  do
  if  $k \in \mathcal{J}$  then
     $w(k) \leftarrow n_k / \alpha \tilde{b}_k$ ;
  else
     $w(k) \leftarrow N / U$ 
  end if
  B  $\leftarrow$  B +  $w(k) \tilde{b}_k$ ; Y  $\leftarrow$  Y +  $w(k) \tilde{f}_k$ 
end for
if  $K + 1 \in \mathcal{J}$  then
   $w(K + 1) \leftarrow n_{K+1} / \alpha \bar{b}_K$ 
else
   $w(K + 1) \leftarrow N / U$ 
end if
B  $\leftarrow$  B +  $w(K + 1) \bar{b}_K$ ; Y  $\leftarrow$  Y +  $w(K + 1) \bar{f}_K$ 
 $c_K \leftarrow (w(K + 1) - w(K)) \alpha$ 
for  $k = K - 1 : 1$  do
   $c_k \leftarrow c_{k+1} \exp(T \Delta x_{k+1}) + (w(k + 1) - w(k)) \alpha$ 
end for
for  $k = 1 : K$  do
  H  $\leftarrow$  H +  $w(k) \mathbf{1} \tilde{f}_k + \text{convint}(\Delta x_k, \bar{b}_{k-1}, c_k)$ 
end for
H  $\leftarrow$  H +  $w(K + 1) \mathbf{1} \bar{f}_K$ 

```

Algorithm 3 Resorting algorithm [55].

```

for  $i = 1 : (m - 1)$  do
  for  $j = 1 : (m - i)$  do
    if  $\lambda_j > \lambda_{j+1}$  then
       $\alpha_j \leftarrow \alpha_j + \alpha_{j+1} (1 - \lambda_{j+1} / \lambda_j)$ ;
       $\alpha_{j+1} \leftarrow \alpha_{j+1} \lambda_{j+1} / \lambda_j$ ;
       $\text{swap}(\lambda_j, \lambda_{j+1})$ ;
    end if
  end for
end for

```

In the M-step, the parameters are updated by using the results of E-step;

$$\alpha_i \leftarrow \frac{\alpha_i[\mathbf{B}]_i}{N + U}, \quad \lambda_i \leftarrow \frac{\lambda_i[\mathbf{H}]_{i+1,i}}{[\mathbf{H}]_{i,i}} \text{ for } i = 1, \dots, m - 1, \quad \lambda_m \leftarrow \frac{\lambda_m[\mathbf{Y}]_m}{[\mathbf{H}]_{m,m}}. \quad (36)$$

However, this procedure does not ensure that the restriction of CF1 holds. Okamura et al. [55] proposed Algorithm 3 as the resorting algorithm of $\lambda_1, \dots, \lambda_m$ for the bidiagonal PH distribution. After every M-step execution, Algorithm 3 is performed to ensure the restriction.

4 MAP Fitting

A MAP (Markovian arrival process) is defined by a time-homogeneous CTMC on a two-dimensional state space $\{(n, i); n = 0, 1, \dots, \text{ and } i = 1, \dots, m\}$ with the following infinitesimal generator:

$$\mathbf{Q}_{\text{MAP}} = \begin{pmatrix} \mathbf{D}_0 & \mathbf{D}_1 & & \\ & \mathbf{D}_0 & \mathbf{D}_1 & \\ & & \ddots & \ddots \end{pmatrix}. \quad (37)$$

The first coordinate of state space is called the *level* which represents the number of arrivals corresponding to the block structure of the infinitesimal generator in which \mathbf{D}_1 corresponds to the arrival rates. The second coordinate of state space is called the *phase* which dominates the arrival rate. In general, the MAP is denoted by a two-dimensional point process (N_t, J_t) where N_t and J_t are the level and phase processes of the MAP. From the infinitesimal generator, J_t becomes a CTMC on the finite state space $\{1, \dots, m\}$ with the infinitesimal generator $\mathbf{D}_0 + \mathbf{D}_1$.

By taking account of time instants when arrivals occur, we build an embedded Markov chain, i.e., the phase process is reduced to a discrete-time Markov chain (DTMC) with the transition probability $\mathbf{P} = (-\mathbf{D}_0)^{-1} \mathbf{D}_1$. Then the interarrival time follows a PH random variable. Let $\boldsymbol{\pi}$ be the probability vector for initial phase at time $t = 0$. Then, the time to the first arrival occurrence becomes a PH random variable with PH parameters $(\boldsymbol{\pi}, \mathbf{D}_0, \mathbf{D}_1 \mathbf{1})$. Generally, let X_0, X_1, \dots, X_k be the k -successive interarrival times of MAP with $(\mathbf{D}_0, \mathbf{D}_1)$. The examples of moment-based characteristics of the MAP are;

- the lag- l joint moments;

$$\mathbb{E}[X_0^i X_l^j] = i! j! \boldsymbol{\pi}_s (-\mathbf{D}_0)^{-i} \mathbf{P}^l (-\mathbf{D}_0)^{-j} \mathbf{1}, \quad (38)$$

- the lag- l autocorrelation;

$$\rho_l = \frac{E[X_0 X_l] - E[X]^2}{E[X^2] - E[X]^2}, \quad (39)$$

where $E[X^k] = k! \boldsymbol{\pi}_s (-\mathbf{D}_0)^{-k} \mathbf{1}$ is the k -th moment of the PH distribution and $\boldsymbol{\pi}_s$ is the stationary vector of \mathbf{P} satisfying $\boldsymbol{\pi}_s = \boldsymbol{\pi}_s \mathbf{P}$.

MAP fitting also encounters the following three research questions:

- (i) What is the appropriate phase structure?
- (ii) How many phases are required?
- (iii) How should the parameters \mathbf{D}_0 and \mathbf{D}_1 be estimated?

Research questions (i) and (ii) have not been overcome yet by the research community. For research question (i), the Markov-modulated Poisson process (MMPP) and interrupted Poisson process (IPP) are often assumed as subclasses of the MAP. Telek and Horvath [68] proposed a minimal representation for general MAP by allowing parameters to take complex values, i.e., a rational arrival process. Also, from the computational point of view, Horvath et al. [32] focused on the interarrival time distribution of the MAP, assuming that it is given by hyper-Erlang distributions. Yoshihara et al. [77] and Casale et al. [18] presented a superposition of the MAP and MMPP. For research question (ii), Telek and Horvath [68] suggested that the size of the underlying CTMC could be determined in the framework of MM for an interarrival time distribution. Also, Okamura et al. [52, 54] suggested using the AIC to determine the number of phases in the context of MLE. However, this is still an open problem in MAP fitting.

For research question (iii), MAP parameter estimation is categorized as MM, MLE, and Bayes estimation as well. Heffes and Lucantoni [30] provided an explicit formula for estimating the parameters of a two-state MMPP by using empirical moments of the number of arrivals. Anderson and Nielsen [4] proposed a fitting method for a superposition of 2-state MAPs using the Hurst parameter in addition to the moments. Yoshihara et al. [77] developed a moment-based estimation procedure for a superposition of MMPP. Mitchell and van de Liefvoort [45] proposed a two-step method that deals with interarrival time data and lag correlation separately. Buchholz and Kriege [15] developed a heuristic algorithm with joint moments.

In the context of MLE, Deng and Mark [21] proposed the MLE for MAPs by converting it to a Markov-modulated Bernoulli process. Rydén [65] derived an EM algorithm for MMPP. The EM algorithm by Rydén [65] is analogous to the forward-backward algorithm for the well-known hidden Markov model [7], and this method can also be applied to the MAP. Breuer [13] and Klemm et al. [41] discussed EM algorithms to estimate the parameters of batch MAP. Roberts et al. [64] proposed a scaling method and a computation method for the matrix exponential to speed up Rydén's EM algorithm. Buchholz [14] reduced the time complexity of the EM algorithm by applying the uniformization technique. Also, Buchholz and Panchenko [16] and Horváth et al. [32] developed two-step fitting methods by combining the EM algorithm for the PH distribution and the moment-based two-step method proposed

in [45]. Okamura et al. [57] provided a deterministic annealing of the EM algorithm for the MAP and GPH distribution to relax the local maximum convergence of the EM algorithm. Furthermore, Okamura and Dohi [52] presented a novel EM algorithm using a hyper-Erlang distribution, which does not need to compute the matrix exponential. This algorithm is one of the most effective algorithms for MAP fitting. The same idea is extended to estimate the marked MAP [33]. On the other hand, MLE strongly depends on the data type. Thus, it is important to consider MAP fitting algorithms for a variety of data types. However, there are only a few works on MAP fitting with different data types. Okamura et al. [54] first discussed the MAP fitting problem with grouped data. Works on Bayes estimation for MAP fitting include Fearnhead and Sherlock [25] who presented an MCMC algorithm based on Gibbs sampling for the MMPP.

5 MAP Fitting Algorithm

5.1 Data for MAP Fitting

In this section, we introduce a detailed MAP fitting algorithm with grouped data proposed in [54]. Similar to the grouped data for PH fitting, the grouped data for MAP fitting is also defined by time intervals and the number of arrivals in those intervals. Table 2 shows examples of time data and grouped data generated from the time data. In this table, the grouped data is created from the time data by counting the number of arrivals in successive 5 s periods.

5.2 EM-Step Formulas

Consider the grouped data $\mathcal{D} = \{n_1, n_2, \dots, n_K\}$ with break points $0=t_0 < t_1 < \dots < t_K$, where n_k is the number of arrivals in $[t_{k-1}, t_k)$. When the grouped data \mathcal{D} are observed, we estimate MAP parameters $(\boldsymbol{\pi}, \mathbf{D}_0, \mathbf{D}_1)$, where $\boldsymbol{\pi}$ is the probability vector to determine the initial phase at time $t = 0$. In this chapter, π_i , $\lambda_{i,j}$ and $\mu_{i,j}$ are the i -th entry of $\boldsymbol{\pi}$, (i, j) -entries of \mathbf{D}_0 and \mathbf{D}_1 , respectively.

Define the following unobserved variables:

- B_i : an indicator random variable for the event that the phase process of the MAP begins with phase i .
- $Z_i^{(k)}$: the total sojourn time of phase i in the interval $[t_{k-1}, t_k)$.
- $Y_{i,j}^{(k)}$: the total number of arrivals leading to phase transitions from phase i to phase j in the interval $[t_{k-1}, t_k)$.
- $M_{i,j}^{(k)}$: the total number of phase transitions from phase i to phase j in the interval $[t_{k-1}, t_k)$.

Table 2 Examples of time and grouped data for MAP fitting (break points: 0, 5, 10, 15, 20, 25, ...)

Time data	
Arrival no.	Time (s)
1	1.240
2	1.608
3	4.206
4	9.140
5	11.036
6	15.075
7	17.912
8	20.604
9	22.032
10	26.300
⋮	⋮
Grouped data	
Time interval	Counts
[0, 5]	3
[5, 10]	1
[10, 15]	1
[15, 20]	2
[20, 25]	2
⋮	⋮

By considering the expected values for the unobserved variables, we have the following M-step formulas of MAP parameters:

$$\pi_i \leftarrow E[B_i | \mathcal{D}], \quad \lambda_{i,j} \leftarrow \frac{\sum_{k=1}^K E[M_{i,j}^{(k)} | \mathcal{D}]}{\sum_{k=1}^K E[Z_i^{(k)} | \mathcal{D}]}, \quad \mu_{i,j} \leftarrow \frac{\sum_{k=1}^K E[Y_{i,j}^{(k)} | \mathcal{D}]}{\sum_{k=1}^K E[Z_i^{(k)} | \mathcal{D}]} \quad (40)$$

Next, we consider the expected values of the unobserved variables, i.e., the E-step formulas for MAP fitting. Define the following row and column vectors:

$$[f_k(n, u)]_i = P(\Delta N_1 = n_1, \dots, \Delta N_{k-1} = n_{k-1}, N_{u+t_{k-1}} - N_{t_{k-1}} = n, J_{u+t_{k-1}} = i), \quad 0 \leq u \leq \Delta t_k, \quad (41)$$

$$[\mathbf{b}_k(n, u)]_i = P(N_{t_k} - N_{t_k-u} = n, \Delta N_{k+1} = n_{k+1}, \dots, \Delta N_K = n_K | J_{t_k-u} = i), \quad 0 \leq u \leq \Delta t_k, \quad (42)$$

where $\Delta N_k = N(t_k) - N(t_{k-1})$ and $\Delta t_k = t_k - t_{k-1}$. The sum of row vector $\mathbf{f}_k(n, u)$ represents the likelihood of observed data in $[0, t_{k-1} + u]$, and the i -th entry of $\mathbf{b}_k(n, u)$ denotes the conditional likelihood of observed data in $[t_k - u, t_k]$ provided that $J_{t_k-u} = i$.

By using $\mathbf{f}_k(n, u)$ and $\mathbf{b}_k(n, u)$, the expected value $E[B_i | \mathcal{D}]$ is given by

$$\begin{aligned} E[B_i | \mathcal{D}] &= \frac{P(J_0 = i, \Delta N_1 = n_1, \dots, \Delta N_K = n_K)}{P(\Delta N_1 = n_1, \dots, \Delta N_K = n_K)} \\ &= \frac{P(J_0 = i)P(\Delta N_1 = n_1, \dots, \Delta N_K = n_K | J_0 = i)}{P(\Delta N_1 = n_1, \dots, \Delta N_K = n_K)} \\ &= \frac{\pi_i [\mathbf{b}_1(n_1, \Delta t_1)]_i}{\boldsymbol{\pi} \mathbf{b}_1(n_1, \Delta t_1)}, \end{aligned} \quad (43)$$

where $\boldsymbol{\pi} \mathbf{b}_1(n_1, \Delta t_1)$ becomes the likelihood of \mathcal{D} . Also, the expected value $E[Z_i^{(k)} | \mathcal{D}]$ becomes

$$E[Z_i^{(k)} | \mathcal{D}] = \frac{\int_0^{\Delta t_k} P(\Delta N_1 = n_1, \dots, \Delta N_k = n_k, J_{t_{k-1}+u} = i, \dots, \Delta N_K = n_K) du}{P(\Delta N_1 = n_1, \dots, \Delta N_K = n_K)}. \quad (44)$$

For the numerator of Eq. (44), we have

$$\begin{aligned} &P(\Delta N_1 = n_1, \dots, \Delta N_k = n_k, J_{t_{k-1}+u} = i, \dots, \Delta N_K = n_K) \\ &= \sum_{l=0}^{n_k} P(\Delta N_1 = n_1, \dots, N_{t_{k-1}+u} - N_{t_{k-1}} = l, J_{t_{k-1}+u} = i) \\ &\quad \times P(N_{t_k} - N_{t_k-u} = n_k - l, \dots, \Delta N_K = n_K | J_{t_{k-1}+u} = i) \\ &= \sum_{l=0}^{n_k} [\mathbf{f}_k(l, u)]_i [\mathbf{b}_k(n_k - l, \Delta t_k - u)]_i. \end{aligned} \quad (45)$$

From Eqs. (44) and (45), the expected value can be written in the form:

$$E[Z_i^{(k)} | \mathcal{D}] = \frac{\int_0^{\Delta t_k} \sum_{l=0}^{n_k} [\mathbf{f}_k(l, u)]_i [\mathbf{b}_k(n_k - l, \Delta t_k - u)]_i du}{\boldsymbol{\pi} \mathbf{b}_1(n_1, \Delta t_1)}. \quad (46)$$

Similarly, the expected value $E[M_{i,j}^{(k)}|\mathcal{D}]$ is given by

$$\begin{aligned}
E[M_{i,j}^{(k)}|\mathcal{D}] &= \frac{1}{P(\Delta N_1 = n_1, \dots, \Delta N_K = n_K)} \\
&\quad \times \int_0^{\Delta t_k} P(\Delta N_1 = n_1, \dots, \Delta N_k = n_k, \\
&\quad J_{t_{k-1}+u}^- = i, J_{t_{k-1}+u} = j, \dots, \Delta N_K = n_K) du \\
&= \frac{1}{\boldsymbol{\pi} \mathbf{b}_1(n_1, \Delta t_1)} \int_0^{\Delta t_k} \sum_{l=0}^{n_k} P(\Delta N_1 = n_1, \dots, N_{t_{k-1}+u} - N_{t_{k-1}} = l, \\
&\quad J_{t_{k-1}+u}^- = i) \times P(J_{t_{k-1}+u} = j | J_{t_{k-1}+u}^- = i) \\
&\quad P(N_{t_k} - N_{t_k-u} = n_k - l, \dots, \Delta N_K = n_K | J_{t_{k-1}+u} = j) du \\
&= \frac{\int_0^{\Delta t_k} \sum_{l=0}^{n_k} [\mathbf{f}_k(l, u)]_i \lambda_{i,j} [\mathbf{b}_k(n_k - l, \Delta t_k - u)]_j du}{\boldsymbol{\pi} \mathbf{b}_1(n_1, \Delta t_1)}. \tag{47}
\end{aligned}$$

Also, the expected value $E[Y_{i,j}^{(k)}|\mathcal{D}]$ is derived as

$$\begin{aligned}
E[Y_{i,j}^{(k)}|\mathcal{D}] &= \frac{1}{P(\Delta N_1 = n_1, \dots, \Delta N_K = n_K)} \\
&\quad \times \int_0^{\Delta t_k} P(\Delta N_1 = n_1, \dots, \Delta N_k = n_k, \\
&\quad N_{t_{k-1}+u} - N_{t_{k-1}+u}^- = 1, J_{t_{k-1}+u}^- = i, J_{t_{k-1}+u} = j, \dots, \Delta N_K = n_K) du \\
&= \frac{1}{\boldsymbol{\pi} \mathbf{b}_1(n_1, \Delta t_1)} \int_0^{\Delta t_k} \sum_{l=0}^{n_k-1} P(\Delta N_1 = n_1, \dots, N_{t_{k-1}+u}^- - N_{t_{k-1}} = l, J_{t_{k-1}+u}^- = i) \\
&\quad \times P(N_{t_{k-1}+u} - N_{t_{k-1}+u}^- = 1, J_{t_{k-1}+u} = j | J_{t_{k-1}+u}^- = i) \\
&\quad \times P(N_{t_k} - N_{t_k-u} = n_k - l - 1, \dots, \Delta N_K = n_K | J_{t_{k-1}+u} = j) du \\
&= \frac{\int_0^{\Delta t_k} \sum_{l=0}^{n_k-1} [\mathbf{f}_k(l, u)]_i \mu_{i,j} [\mathbf{b}_k(n_k - l - 1, \Delta t_k - u)]_j du}{\boldsymbol{\pi} \mathbf{b}_1(n_1, \Delta t_1)}, \tag{48}
\end{aligned}$$

where $N_t^- = \lim_{t' \rightarrow 0^+} N_{t-t'}$.

5.3 Computation of E-Step Formulas

The E-step of MAP fitting with grouped data requires the computation of $\mathbf{f}_k(n, u)$, $\mathbf{b}_k(n, u)$, and their convolutions. Here, we define the following row and column

vectors:

$$\hat{\mathbf{f}}_k(u) = (f_k(0, u) \dots f_k(n_k, u)), \quad \hat{\mathbf{b}}_k(u) = \begin{pmatrix} \mathbf{b}_k(n_k, u) \\ \vdots \\ \mathbf{b}_k(0, u) \end{pmatrix}. \quad (49)$$

Hence, $\hat{\mathbf{f}}_k(u)$ and $\hat{\mathbf{b}}_k(u)$ can be expressed as

$$\begin{aligned} \hat{\mathbf{f}}_k(u) &= \hat{\boldsymbol{\pi}} \exp(\hat{\mathbf{D}}_0(1)\Delta t_1)\hat{\mathbf{I}}(1) \\ &\quad \times \dots \times \exp(\hat{\mathbf{D}}_0(k-1)\Delta t_{k-1})\hat{\mathbf{I}}(k-1) \exp(\hat{\mathbf{D}}_0(k)u), \end{aligned} \quad (50)$$

$$\begin{aligned} \hat{\mathbf{b}}_k(u) &= \exp(\hat{\mathbf{D}}_0(k)u)\hat{\mathbf{I}}(k) \exp(\hat{\mathbf{D}}_0(k+1)\Delta t_{k+1})\hat{\mathbf{I}}(k+1) \\ &\quad \times \dots \times \exp(\hat{\mathbf{D}}_0(K)\Delta t_K)\hat{\mathbf{I}}(K)\hat{\mathbf{1}}, \end{aligned} \quad (51)$$

where $\hat{\mathbf{D}}_0(k)$ and $\hat{\mathbf{I}}(k)$ are the following block matrices:

$$\hat{\mathbf{D}}_0(k) = \underbrace{\begin{pmatrix} \mathbf{D}_0 & \mathbf{D}_1 & & \\ & \mathbf{D}_0 & \mathbf{D}_1 & \\ & & \ddots & \ddots \\ & & & \mathbf{D}_0 \end{pmatrix}}_{m(n_k+1)}, \quad \hat{\mathbf{I}}(k) = \underbrace{\begin{pmatrix} \mathbf{O} & \dots & \mathbf{O} \\ \vdots & \dots & \vdots \\ \mathbf{O} & \dots & \mathbf{O} \\ \mathbf{I} & \mathbf{O} & \dots & \mathbf{O} \end{pmatrix}}_{m(n_{k+1}+1)}. \quad (52)$$

The row and column vectors $\hat{\boldsymbol{\pi}}$ and $\hat{\mathbf{1}}$ are defined by

$$\hat{\boldsymbol{\pi}} = \underbrace{(\boldsymbol{\pi} \ \mathbf{0} \ \dots \ \mathbf{0})}_{m(n_1+1)}, \quad \hat{\mathbf{1}} = \begin{pmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{1} \end{pmatrix} \Bigg\} m(n_K+1). \quad (53)$$

Next, we define the following convolution integral of $\hat{\mathbf{b}}_k(u)$ and $\hat{\mathbf{f}}_k(u)$:

$$\hat{\mathbf{H}}_k = \int_0^{\Delta t_k} \hat{\mathbf{b}}_k(\Delta t_k - s)\hat{\mathbf{f}}_k(s)ds. \quad (54)$$

From the definition of $\hat{\mathbf{b}}_k(u)$ and $\hat{\mathbf{f}}_k(u)$, the sum of diagonal matrices yields an m -by- m matrix representing the convolution of $f_k(n, u)$ and $b_k(n, u)$, i.e.,

$$\sum_{v=0}^{n_k} \hat{\mathbf{H}}_k(u)[v, v] = \int_0^u \sum_{l=0}^{n_k} \mathbf{b}_k(n_k - l, u - s) \mathbf{f}_k(l, s) ds, \quad (55)$$

where $\hat{\mathbf{H}}_k[v, v]$ is the $(v + 1)$ -st diagonal submatrix of $\hat{\mathbf{H}}_k$. Also, the sum of subdiagonal matrices becomes

$$\sum_{v=1}^{n_k} \hat{\mathbf{H}}_k(u)[v, v - 1] = \int_0^u \sum_{l=0}^{n_k-1} \mathbf{b}_k(n_k - l - 1, u - s) \mathbf{f}_k(l, s) ds. \quad (56)$$

For the sake of notational convenience, we define $\hat{\mathbf{f}}_k = \hat{\mathbf{f}}_k(\Delta t_k)$ and $\hat{\mathbf{b}}_k = \hat{\mathbf{b}}_k(\Delta t_k)$. Also, let $\hat{\mathbf{f}}_k[v]$ and $\hat{\mathbf{b}}_k[v]$, $v = 0, 1, \dots, n_k$ be the v -th subvectors of $\hat{\mathbf{f}}_k$ and $\hat{\mathbf{b}}_k$, respectively. Then, the expected values can be rewritten as

$$\begin{aligned} \mathbb{E}[B_i | \mathcal{O}] &= \frac{\pi_i [\hat{\mathbf{b}}_1[0]]_i}{\boldsymbol{\pi} \hat{\mathbf{b}}_1[0]}, & \mathbb{E}[Z_i^{(k)} | \mathcal{O}] &= \frac{[\sum_{v=0}^{n_k} \hat{\mathbf{H}}_k[v, v]]_{i,i}}{\boldsymbol{\pi} \hat{\mathbf{b}}_1[0]}, \\ \mathbb{E}[M_{i,j}^{(k)} | \mathcal{O}] &= \frac{\lambda_{i,j} [\sum_{v=0}^{n_k} \hat{\mathbf{H}}_k[v, v]]_{j,i}}{\boldsymbol{\pi} \hat{\mathbf{b}}_1[0]}, & \mathbb{E}[Y_{i,j}^{(k)} | \mathcal{O}] &= \frac{\mu_{i,j} [\sum_{v=1}^{n_k} \hat{\mathbf{H}}_k[v, v - 1]]_{j,i}}{\boldsymbol{\pi} \hat{\mathbf{b}}_1[0]}. \end{aligned} \quad (57)$$

$$(58)$$

From Eqs. (50) and (51), $\hat{\mathbf{f}}_k$ and $\tilde{\mathbf{b}}_k$ can be obtained by computing the matrix exponential in the forward-backward manner. On the other hand, Eq. (54) can be reduced to

$$\begin{aligned} \hat{\mathbf{H}}_k &= \int_0^{\Delta t_k} \exp(\hat{\mathbf{D}}_0(k)(\Delta t_k - s)) \hat{\mathbf{I}}(k) \hat{\mathbf{b}}_{k+1} \hat{\mathbf{f}}_{k-1} \hat{\mathbf{I}}(k - 1) \exp(\hat{\mathbf{D}}_0(k)s) ds \\ &= \boldsymbol{\Upsilon}(\Delta t_k; \hat{\mathbf{I}}(k) \hat{\mathbf{b}}_{k+1}, \hat{\mathbf{f}}_{k-1} \hat{\mathbf{I}}(k - 1)). \end{aligned} \quad (59)$$

Therefore, Algorithm 1 can be applied to compute $\hat{\mathbf{H}}_k$. Since the computation time is proportional to the number of nonzero elements of $\mathbf{D}_0(k)$, the time complexity is proportional to $(n_k + 1)m^2$ even if \mathbf{D}_0 and \mathbf{D}_1 are m -by- m full matrices.

Finally, we present the E-step computation of MAP fitting with grouped data in Algorithm 4.⁴ By using the algorithm, the parameters are updated with $\pi_i \leftarrow \pi_i[\mathbf{B}]_i$, $\lambda_{i,j} \leftarrow \lambda_{i,j}[\mathbf{H}]_{j,i} / [\mathbf{H}]_{i,i}$, and $\mu_{i,j} \leftarrow \mu_{i,j}[\mathbf{Y}]_{j,i} / [\mathbf{H}]_{i,i}$.

⁴In the implementation, to avoid underflow in $\hat{\mathbf{f}}_k$ and $\hat{\mathbf{b}}_k$, the scaling technique should be applied.

Algorithm 4 E-step procedure for MAP fitting with grouped data [54].

```

H ← O; Y ← O;  $\hat{f}_0 \leftarrow \hat{\pi}$ ;  $\hat{b}_{K+1} \leftarrow \hat{\mathbf{1}}$ 
for  $k = 1 : K$  do
   $\hat{f}_k \leftarrow \hat{f}_{k-1} \exp(\hat{D}_0(k) \Delta t_k) \hat{\mathbf{I}}(k)$ 
end for
for  $k = K : 1$  do
   $\hat{b}_k \leftarrow \exp(\hat{D}_0(k) \Delta t_k) \hat{\mathbf{I}}(k) \hat{b}_{k+1}$ 
end for
B ←  $\hat{b}_1[0] / \pi \hat{b}_1[0]$ 
for  $k = 1 : K$  do
   $\hat{H}_k \leftarrow \text{convint}(\Delta t_k, \hat{\mathbf{I}}(k) \hat{b}_{k+1}, \hat{f}_{k-1} \hat{\mathbf{I}}(k-1)) / \pi \hat{b}_1[0]$ 
   $H \leftarrow H + \sum_{v=0}^{n_k} \hat{H}_k[v, v]$ 
  if  $n_k \geq 1$  then
     $Y \leftarrow Y + \sum_{v=1}^{n_k} \hat{H}_k[v, v-1]$ 
  end if
end for

```

6 Tools

There have been several PH/MAP fitting software tools available on the Internet. *Empht* [23] is a C program language-based tool for PH fitting with EM algorithms by Asmussen et al. [6] and Olsson [59]. *PhFit* [31, 62] consists of a Java-based interface and computational engine written in the C language. This tool implements PH fitting for continuous and discrete PH distributions. Furthermore, it can be applied to PH fitting from a p.d.f. *momentmatching* [46] is a set of MATLAB files to execute three moment matching algorithms [60]. *BuTools* [17] are program packages for Mathematica and MATLAB/Octave. The tool provides PH/MAP fitting with the MM method. *G-FIT* [27] is a command line tool to provide EM algorithms for the hyper-Erlang distribution [61, 69]. *jPhaseFit* [40] is a library for Java to handle PHs. In this library, both of PH fitting with MM and EM algorithm are implemented. In particular, the EM algorithm for the hyper-Erlang distribution [69] was implemented in this tool. *HyperStar* [34] is a Java-based GUI tool to estimate the hyper-Erlang distribution and to plot graphs. It implements the cluster-based algorithm [63]. KPC Toolbox [70] is a library for MATLAB for PH/MAP fitting based on MM presented in [18]. *mapfit* [42, 53] is an R package for PH/MAP fitting, which is distributed by CRAN (The Comprehensive R Archive Network). This package provides the fast EM algorithms for PH/MAP [52, 55] and PH/MAP fitting with grouped data [54, 56].

7 Conclusion

In this chapter, we have introduced the PH/MAP fitting, specifically, statistical estimation algorithms and tools. Since the PH/MAP fitting has been studied since the early of 1980s, there exist several interesting results that could not be covered in this chapter. In the last five years, the computation speed of PH/MAP fitting has been drastically improved to study large-scale systems. For instance, the latest tool developed by the authors, *mapfit*, can handle the PH fitting for CF1 with over 200 phases. These methods and their associated PH/MAP fitting tools are expected to be used in performance evaluation of computer, communication and production systems in the real world. On the other hand, from the theoretical point of view, the fitting techniques for matrix-exponential distribution [24], rational arrival process [68], and the marked MAP [29], which are generalized from PH and MAP, are quite challenging issues. The similar techniques of PH/MAP fitting introduced in this chapter will be useful to attain the new achievements.

Acknowledgments The authors (H.O. and T.D.) are grateful to Prof. Kishor S. Trivedi who stimulated their research interests in model-based performance evaluation through his many papers.

References

1. Akaike H (1973) Information theory and an extension of the maximum likelihood principle. In: Petrov BN, Csaki F (eds) Proceedings of the 2nd international symposium on information theory. Akademiai Kiado, pp 267–281
2. Aldous D, Shepp L (1987) The least variable phase-type distribution is Erlang. *Stoch Models* 3:467–473
3. Altioik T (1985) On the phase-type approximations of general distributions. *IEEE Trans* 17:110–116
4. Andersen AT, Nielsen BF (1998) A Markovian approach for modeling packet traffic with long-range dependence. *IEEE J Sel Areas Commun* 16(5):719–732
5. Asmussen S, Koole G (1993) Marked point processes as limits of Markovian arrival streams. *J Appl Probab* 30:365–372
6. Asmussen S, Nerman O, Olsson M (1996) Fitting phase-type distributions via the EM algorithm. *Scand J Stat* 23(4):419–441
7. Baum LE, Petrie T, Soules G, Weiss N (1970) A maximization technique occurring in the statistical analysis of probabilistic function of Markov chains. *Ann Math Stat* 41(1):164–171
8. Bladt M, Gonzalez A, Lauritzen SL (2003) The estimation of phase-type related functionals using Markov chain Monte Carlo methods. *Scand Act J* 4:280–300
9. Bobbio A, Cumani A (1992) ML estimation of the parameters of a PH distribution in triangular canonical form. In: Balbo G, Serazzi G (eds) Computer performance evaluation. Elsevier Science Publishers, pp 33–46
10. Bobbio A, Horváth A, Telek M (2005) Matching three moments with minimal acyclic phase type distributions. *Stoch Models* 21(2/3):303–326
11. Bobbio A, Telek M (1994) A benchmark for PH estimation algorithms: result for acyclic-PH. *Stoch Models* 10:661–677
12. Bolch G, Greiner S, de Meer H, Trivedi KS (2006) Queueing networks and Markov chains: modeling and performance evaluation with computer science applications, 2nd edn. Wiley, New York

13. Breuer L (2002) An EM algorithm for batch Markovian arrival processes and its comparison to a simpler estimation procedure. *Ann Oper Res* 112:123–138
14. Buchholz P (2003) An EM-algorithm for MAP fitting from real traffic data. In: Kemper P, Sanders WH (eds) *Computer performance evaluation modelling techniques and tools*, vol 2794. Springer, LNCS, pp 218–236
15. Buchholz P, Kriege J (2009) A heuristic approach for fitting MAPs to moments and joint moments. In: *Proceedings of the 6th international conference on the quantitative evaluation of systems*, pp 53–62
16. Buchholz P, Panchenko A (2004) A two-step EM-algorithm for MAP fitting. In: *Proceedings of 19th international symposium on computer and information sciences*, LNCS, vol 3280. Springer, pp 217–227
17. BuTools: <http://webspn.hit.bme.hu/telek/tools/butools/butools.html>
18. Casale G, Zhang E, Smirni E (2008) KPC-toolbox: simple yet effective trace fitting using Markovian arrival processes. In: *Proceedings of 5th international conference on quantitative evaluation of systems (QEST2008)*, pp 83–92
19. Cumani A (1982) On the canonical representation of homogeneous Markov processes modelling failure-time distributions. *Microelectron Reliab* 22:583–602
20. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. *J R Stat Soc Series B* 39:1–38
21. Deng L, Mark J (1993) Parameter estimation for Markov modulated Poisson processes via the EM algorithm with time discretization. *Telecommun Syst* 1:321–338
22. Distefano S, Trivedi KS (2013) Non-Markovian state-space models in dependability evaluation. *Qual Reliab Eng Int* 29(2):225–239
23. EMpht: <http://home.math.au.dk/asmus/pspapers.html>
24. Fackrell M (2005) Fitting with matrix-exponential distributions. *Stoch Models* 21:377–400
25. Fearnhead P, Sherlock C (2006) An exact Gibbs sampler for the Markov modulated Poisson process. *J R Stat Soc Series B* 66(5):767–784
26. Feldmann A, Whitt W (1998) Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Perform Eval* 31:245–258
27. G-FIT: <http://ls4-www.cs.tu-dortmund.de/home/thummler/mainE.html>
28. He QH, Zhang H (2008) An algorithm for computing minimal Coxian representations. *INFORMS J Comput* 20(2):179–190
29. He QM, Neuts MF (1998) Markov chains with marked transitions. *Stoch Proces Appl* 74:37–52
30. Heffes H, Lucantoni DM (1986) A Markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE J Sel Areas Commun* 4(6):856–868
31. Horváth A, Telek M (2002) PhFit: a general phase-type fitting tool. In: *Proceedings of computer performance evaluation/tools*, lecture notes in computer science, vol. 2324. Springer, pp 82–91
32. Horváth G, Buchholz P, Telek M (2005) A MAP fitting approach with independent approximation of the inter-arrival time distribution and the lag correlation. In: *Proceedings of the 2nd international conference on the quantitative evaluation of systems*. IEEE CS Press, pp 124–133
33. Horváth G, Okamura H (2013) A fast EM algorithm for fitting marked Markovian arrival processes with a new special structure. In: *Proceedings of European performance engineering workshop*, lecture notes in computer science (LNCS), vol 8168, pp 119–133
34. HyperStar: <http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/HyperStar/index.html>
35. Johnson MA (1993) Selecting parameters of phase distributions: combining nonlinear programming, heuristics, and Erlang distributions. *ORSA J Comput* 5:69–83
36. Johnson MA, Taaffe MR (1989) Matching moments to phase distributions: mixtures of erlang distribution of common order. *Stoch Models* 5:711–743
37. Johnson MA, Taaffe MR (1990) Matching moments to phase distributions: density function shapes. *Stoch Models* 6:283–306
38. Johnson MA, Taaffe MR (1990) Matching moments to phase distributions: nonlinear programming approaches. *Stoch Models* 6:259–281

39. Johnson MA, Taaffe MR (1991) An investigation of phase-distribution moment-matching algorithms for use in queueing models. *Queueing Syst* 8:129–148
40. jPhaseFit: <http://copa.uniandes.edu.co/?p=141>
41. Klemm A, Lindemann C, Lohmann M (2003) Traffic modeling of IP networks using the batch Markovian arrival process. *Perform Eval* 54:149–173
42. mapfit: <http://cran.r-project.org/web/packages/mapfit/index.html>
43. Marie R (1980) Calculating equilibrium probabilities for $\lambda(n)/C_k/1/N$ queues. In: *Proceedings of the 1980 international symposium on computer performance modelling, measurement and evaluation (SIGMETRICS'80)*. ACM Press, pp 117–125
44. McLachlan GJ, Jones PN (1988) Fitting mixture models to grouped and truncated data via the EM algorithm. *Biometrics* 44(2):571–578
45. Mitchell K, van de Liefvoort A (2003) Approximation models of feed-forward G/G/1/N queueing networks with correlated arrivals. *Perform Eval* 52(2–4):137–152
46. Momentmatching: <http://www.cs.cmu.edu/osogami/code/momentmatching/index.html>
47. Neuts MF (1981) *Matrix-geometric solutions in stochastic models*. Johns Hopkins University Press, Baltimore
48. O' Cinneide CA (1989) On nonuniqueness of representations of phase-type distributions. *Stoch Models* 5:322–330
49. O' Cinneide CA (1990) Characterization of phase-type distributions. *Stoch Models* 6:1–57
50. O' Cinneide CA (1991) Phase-type distributions and invariant polytopes. *Adv Appl Probab* 23:515–535
51. O' Cinneide CA (1993) Triangular order of triangular phase-type distributions. *Stoch Models* 9:507–529
52. Okamura H, Dohi T (2009) Faster maximum likelihood estimation algorithms for Markovian arrival processes. In: *Proceedings of 6th international conference on quantitative evaluation of systems (QEST2009)*, pp 73–82
53. Okamura H, Dohi T (2015) Mapfit: an R-based tool for PH/MAP parameter estimation. In: *Proceedings of The 12th international conference on quantitative evaluation of systems (QEST2015)*, pp 105–112
54. Okamura H, Dohi T, Trivedi KS (2009) Markovian arrival process parameter estimation with group data. *IEEE/ACM Trans Netw* 17(4):1326–1339
55. Okamura H, Dohi T, Trivedi KS (2011) A refined EM algorithm for PH distributions. *Perform Eval* 68(10):938–954
56. Okamura H, Dohi T, Trivedi KS (2013) Improvement of EM algorithm for phase-type distributions with grouped and truncated data. *Appl Stoch Models Bus Ind* 29(2):141–156
57. Okamura H, Kishikawa H, Dohi T (2013) Application of deterministic annealing EM algorithm to MAP/PH parameter estimation. *Telecommun Syst* 54:79–90
58. Okamura H, Watanabe R, Dohi T (2014) Variational Bayes for phase-type distribution. *Commun Stat Theory Methods* 43(8):2013–2044
59. Olsson M (1996) Estimation of phase-type distributions from censored data. *Scand J Stat* 23:443–460
60. Osogami T, Harchol-Balter M (2006) Closed form solutions for mapping general distributions to minimal PH distributions. *Perform Eval* 63(6):524–552
61. Panchenko A, Thümmler A (2007) Efficient phase-type fitting with aggregated traffic traces. *Perform Eval* 64:629–645
62. PhFit: <http://webspn.hit.bme.hu/telek/tools.htm>
63. Reinecke P, Kraub T, Wolter K (2012) Cluster-based fitting of phase-type distributions to empirical data. *Comput Math Appl* 64:3840–3851
64. Roberts WJJ, Ephraim Y, Dieguez E (2006) On Rydén's EM algorithm for estimating MMPPs. *IEEE Signal Process Lett* 13(6):373–376
65. Rydén T (1996) An EM algorithm for estimation in Markov-modulated Poisson processes. *Comput Stat Data Anal* 21(4):431–447
66. Schwarz G (1978) Estimating the dimension of a model. *Ann Stat* 6:461–464

67. Telek M, Heindl A (2002) Matching moments for acyclic discrete and continuous phase-type distributions of second order. *Int J Simul Syst Sci Technol* 3(3–4):47–57
68. Telek M, Horváth G (2007) A minimal representation of Markov arrival processes and a moments matching method. *Perform Eval* 64(9–12):1153–1168
69. Thümmler A, Buchholz P, Telek M (2006) A novel approach for phase-type fitting with the EM algorithm. *IEEE Trans Dependable Secure Comput* 3(3):245–258
70. Toolbox K, <http://www.cs.wm.edu/MAPQN/kpctoolbox.html>
71. Trivedi KS (2001) *Probability and statistics with reliability, queueing, and computer sciences applications*, 2nd edn. Wiley, New York
72. van der Heijden MC (1988) On the three-moment approximation of a general distribution by a Coxian distribution. *Probab Eng Inf Sci* 2:257–261
73. van de Liefvoort A (1990) The moment problem for continuous distributions. Technical report, University of Missouri, WP-CM-1990-02, Kansas City
74. Watanabe R, Okamura H, Dohi T (2012) An efficient MCMC algorithm for continuous PH distributions. In: Laroque C, Himmelspach J, Pasupathy R, Rose O, Uhrmacher AM (eds) *Proceedings of The 2012 winter simulation conference*, p 12
75. Wu CFJ (1983) On the convergence properties of the EM algorithm. *Ann Stat* 11:95–103
76. Yamaguchi Y, Okamura H, Dohi T (2010) Variational Bayesian approach for estimating parameters of a mixture of Erlang distributions. *Commun Stat Theory Methods* 39(3):2333–2350
77. Yoshihara T, Kasahara S, Takahashi Y (2001) Practical time-scale fitting of self-similar traffic with Markov modulated Poisson process. *Telecommun Syst* 17(1–2):185–211

Constant-Stress Accelerated Life-Test Models and Data Analysis for One-Shot Devices

Narayanaswamy Balakrishnan, Man Ho Ling and Hon Yiu So

Abstract In reliability analysis, accelerated life-tests are commonly used for inducing rapid failures, thus producing more lifetime information in a relatively short period of time. A link function relating stress levels and lifetimes is then utilized to extrapolate lifetimes of units from accelerated conditions to normal operating conditions. In the context of one-shot device testing, encountered commonly in testing devices such as munitions, rockets, and automobile air bags, either left- or right-censored data are collected instead of actual lifetimes of the devices under test. In this chapter, we study binary response data of one-shot devices collected from constant-stress accelerated life-tests, and discuss the analysis of such one-shot device testing data under accelerated life-tests based on parametric and semi-parametric models. In addition, a competing risks model is introduced into the one-shot device testing analysis under constant-stress accelerated life-test setting. Finally, some numerical examples are presented to illustrate the models and inferential results discussed here.

1 Introduction

A one-shot device is a unit that performs its function only once and cannot be used for testing more than once; examples include electric explosive devices, fire extinguishers, airbags in cars, and missiles (see [10, 26, 29]). While testing one-shot devices, only the condition of the device at a specific inspection time can be recorded, and

N. Balakrishnan (✉) · H.Y. So
Department of Mathematics and Statistics, McMaster University,
1280 Main Street West, Hamilton, ON L8S 4K1, Canada
e-mail: bala@mcmaster.ca

H.Y. So
e-mail: honyius@gmail.com

M.H. Ling
Department of Mathematics and Information Technology,
The Hong Kong Institute of Education, Hong Kong Sar, China
e-mail: amhling@ied.edu.hk

exact failure times cannot be obtained from the test. As a result, the lifetimes of devices are either left- or right-censored, with the lifetime being less than the inspection time if the test outcome is a failure (resulting in left censoring) and the lifetime being more than the inspection time if the test outcome is a success (resulting in right censoring). One-shot device testing data also arise in destructive life-tests, wherein tested devices cannot be used again since the test results in their destruction.

Due to customers' needs and demands, new products are designed to be highly reliable with substantially long lifetimes. So, accelerated life-tests, wherein test units are subject to higher-than-normal stress levels, are usually performed to collect sufficient failure time information about the products. In this regard, reliability analysis for one-shot device testing data from constant-stress accelerated life-tests has recently received much attention. Estimation and inferential issues under different parametric models have been comprehensively discussed in the literature. Morris [25] analyzed battery data from destructive life-tests in which the batteries were stored under a mildly accelerated aging temperature and a relatively light load. Fan et al. [10] compared three different priors in the Bayesian approach for making predictions on the reliability and the mean lifetime of electro-explosive devices at typical operating conditions under the exponential lifetime distribution. Balakrishnan and Ling [1] developed an EM algorithm for the estimation of parameters of a one-shot device model under exponential distribution with a single stress factor and compared their results with those of Fan et al. [10] based on the Bayesian approach. Balakrishnan and Ling [2] further extended this approach to multiple stress factors, and developed an EM algorithm for the same problem under Weibull [3] and gamma [4] lifetime distributions, respectively. Recently, Balakrishnan et al. [5, 6] extended this line of work by introducing a competing risks model into a one-shot device testing analysis under an accelerated life test setting, and Ling et al. [21] considered semi-parametric models for analyzing one-shot device testing data collected from constant-stress accelerated life-tests.

2 Likelihood Inference

In constant-stress accelerated life-tests for one-shot devices, the devices are placed in I test groups. Suppose, for $i = 1, 2, \dots, I$, K_i devices are subjected to J types of stress factors at stress levels $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iJ})$ and under inspection at time IT_i . In the i -th test group, the number of failures, n_i , is collected. The data thus observed can be summarized as in Table 1. The purpose of the test is to determine the life characteristic of the device under normal operating conditions, $\mathbf{x}_0 = (x_{01}, x_{02}, \dots, x_{0J})$. For notational convenience, we denote $\mathbf{z} = \{IT_i, K_i, n_i, \mathbf{x}_i, i = 1, 2, \dots, I\}$ for the observed data, and θ for the model parameters. Suppose the lifetime distribution has

Table 1 Data on one-shot device testing at various stress levels collected at different inspection times

Test group	Inspection time	# of tested devices	# of failures	Stress levels
1	IT_1	K_1	n_1	$(x_{11}, x_{12}, \dots, x_{1J})$
2	IT_2	K_2	n_2	$(x_{21}, x_{22}, \dots, x_{2J})$
\vdots	\vdots	\vdots	\vdots	\vdots
I	IT_I	K_I	n_I	$(x_{I1}, x_{I2}, \dots, x_{IJ})$

a cumulative distribution function (cdf) $F(t; \theta)$. The observed likelihood function is then given by

$$L(\theta; \mathbf{z}) = C \prod_{i=1}^I [F(IT_i; \theta)]^{n_i} [1 - F(IT_i; \theta)]^{K_i - n_i},$$

where C is the normalizing constant.

2.1 EM Algorithm

The EM algorithm is an efficient and powerful technique for finding the maximum likelihood estimates (MLEs) of the model parameters in the presence of missing data; see McLachlan and Krishnan [23]. It is suitable for the determination of the MLEs of the model parameters when all the failure times of the devices are censored like in the case of one-shot device testing data. The EM algorithm simply involves approximating the missing data (the expectation step/E-step) and maximizing the corresponding likelihood function (the maximization step/M-step) in each iteration. These are developed here in this section for the problem under consideration.

Suppose the lifetime distribution has a probability density function (pdf) $f(t; \theta)$. In the EM algorithm, we first consider the log-likelihood function based on the complete data given by

$$\ell_c(\theta) = \sum_{i=1}^I \sum_{k=1}^{K_i} \ln(f(t_{ik}; \theta)).$$

The objective then is to maximize the function

$$Q(\theta, \theta^{(m)}) = E_{\theta^{(m)}}[\ell_c(\theta) | \mathbf{z}]$$

based on the current estimate $\boldsymbol{\theta}^{(m)}$ in the M-step, and then to obtain the conditional expectation $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m+1)})$ based on the updated estimate $\boldsymbol{\theta}^{(m+1)}$ in the E-step. These two steps are repeated until convergence is achieved to a desired level of accuracy. It can be seen that we actually solve the incomplete data problem by first approximating the missing data and then using the approximated values to find the estimate of the parameter vector as the solution for the complete data problem.

However, for the maximization problem, a closed-form solution cannot be found. The EM algorithm based on the one-step Newton-Raphson method needs to be employed for this purpose. It requires the second-order derivatives of the log-likelihood function with respect to the model parameters. For the present situation, let us define

$$\mathbf{I} = \left[\frac{\partial Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial \boldsymbol{\theta}} \right]_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(m)}}$$

and

$$\mathbf{J} = - \left[\frac{\partial^2 Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \right]_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(m)}}.$$

Then, we have the updated parameters as

$$\boldsymbol{\theta}^{(m+1)} = \mathbf{J}^{-1} \mathbf{I} + \boldsymbol{\theta}^{(m)}.$$

2.2 Asymptotic Confidence Intervals

To construct confidence intervals for a parameter of interest, asymptotic confidence intervals are commonly used when sufficiently large sample sizes are available. This would require the asymptotic variance-covariance matrix of the MLEs of the model parameters, $\hat{\boldsymbol{\theta}}$. Under the EM framework, the missing information principle, introduced by Louis [22], is often employed to obtain the observed information matrix of the MLEs of the model parameters. This method requires the complete information matrix, as well as the missing information matrix. These matrices are, respectively, given by

$$I_c = - \left[\frac{\partial^2 (\ell_c(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \quad \text{and} \quad I_m = - \left[\sum_{i=1}^I \sum_{k=1}^{K_i} \frac{\partial^2 (\ln(f(t_{ik} | \mathbf{z}; \boldsymbol{\theta})))}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}.$$

Using these matrices, we will then obtain the observed information matrix as

$$I_{\text{obs}} = I_c - I_m. \tag{1}$$

Theorem 1 *Suppose the lifetime distribution has a pdf $f(t; \theta)$ and a cdf $F(t; \theta)$. For the case when all lifetimes are censored, such as in one-shot device testing data, the observed Fisher information matrix based on the observed likelihood function is identical to the observed information matrix obtained by the Missing Information Principle in (1).*

Proof Given a data with a sequence of inspection times $0 < IT_1 < IT_2 < \dots < IT_{I-1}$ and the corresponding numbers of failures in each time slot, $\{n_1, n_2, \dots, n_I\}$, and K observed failure times, $t_k, k = 1, 2, \dots, K$, the log-likelihood function is then given by

$$\begin{aligned} \ell(\theta) = & n_1 \ln F(IT_1; \theta) + \sum_{i=2}^{I-1} n_i \ln(F(IT_i; \theta) - F(IT_{i-1}; \theta)) \\ & + n_I \ln(1 - F(IT_{I-1}; \theta)) + \sum_{k=1}^K \ln(f(t_k; \theta)) + \ln(C), \end{aligned}$$

where n_1, n_2, \dots, n_I and t_1, t_2, \dots, t_K are random variables.

On the other hand, the log-likelihood function for complete data is given by

$$\ell_c = \sum_{i=1}^I \sum_{j=1}^{n_i} \ln(f(t_j; \theta)) + \sum_{k=1}^K \ln(f(t_k; \theta)) + \ln(C),$$

while the log-likelihood function of the conditional distribution for missing data is given by

$$\begin{aligned} \ell_m = & \sum_{i=1}^I \sum_{j=1}^{n_i} \ln(f(t_j; \theta)) - \sum_{j=1}^{n_1} \ln(F(IT_1; \theta)) - \sum_{j=1}^{n_I} \ln(1 - F(IT_{I-1}; \theta)) \\ & - \sum_{i=2}^{I-1} \sum_{j=1}^{n_i} \ln(F(IT_i; \theta) - F(IT_{i-1}; \theta)) + \ln(C). \end{aligned}$$

Note that t_j in ℓ_c and ℓ_m are also random variables, but the terms of $\ln(f(t_j; \theta))$ are canceled out by the Missing Information Principle. Hence, the observed log-likelihood function is obtained from the Missing Information Principle as

$$\begin{aligned} \ell_{\text{obs}} = & n_1 \ln F(IT_1; \theta) + \sum_{i=2}^{I-1} n_i \ln(F(IT_i; \theta) - F(IT_{i-1}; \theta)) \\ & + n_I \ln(1 - F(IT_{I-1}; \theta)) + \sum_{k=1}^K \ln(f(t_k; \theta)) + \ln(C), \end{aligned}$$

where t_1, t_2, \dots, t_K are random variables. Therefore, for the case when all lifetimes are censored, the terms of $\sum_{k=1}^K \ln(f(t_k; \theta))$ in both $\ell(\theta)$ and ℓ_c disappear. Moreover, it is easy to show that the observed information matrix from the Missing Information Principle is identical to the observed Fisher information matrix based on the observed likelihood function. □

For the case of one-shot device testing data, due to the fact that all failure times are censored, the observed information matrix obtained from the Missing Information Principle is identical to the observed Fisher information matrix obtained from the log-likelihood function conditional on \mathbf{z} . The observed log-likelihood function is given by

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^I [n_i \ln(F(IT_i; \boldsymbol{\theta})) + (K_i - n_i) \ln(1 - F(IT_i; \boldsymbol{\theta}))] + \ln(C).$$

Then, the observed Fisher information matrix and the variance-covariance matrix of the MLEs of the model parameters can be obtained as

$$\begin{aligned} I_{\text{obs}} &= - \left[\frac{\partial^2(\ell(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \\ &= \sum_{i=1}^I \frac{\partial^2 F(IT_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \left(\frac{n_i}{F(IT_i; \boldsymbol{\theta})} - \frac{K_i - n_i}{1 - F(IT_i; \boldsymbol{\theta})} \right) \\ &\quad - \sum_{i=1}^I \frac{\partial F(IT_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial F(IT_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} \left(\frac{n_i}{(F(IT_i; \boldsymbol{\theta}))^2} + \frac{K_i - n_i}{(1 - F(IT_i; \boldsymbol{\theta}))^2} \right) \end{aligned}$$

and

$$\hat{\mathbf{V}}(\hat{\boldsymbol{\theta}}) = I_{\text{obs}}^{-1},$$

respectively.

In addition, the variance of the MLE of any parameter of interest ϕ can also be computed using the delta method as

$$\hat{\mathbf{V}}(\hat{\phi}) = \mathbf{P}^T \hat{\mathbf{V}}(\hat{\boldsymbol{\theta}}) \mathbf{P},$$

where $\mathbf{P} = [\partial \phi / \partial \boldsymbol{\theta}]$ is a column vector. Then, the estimated standard error of $\hat{\phi}$ is immediately obtained as

$$\hat{s}e(\hat{\phi}) = \sqrt{\hat{\mathbf{V}}(\hat{\phi})}. \quad (2)$$

Consequently, the $100(1 - \alpha) \%$ asymptotic confidence interval for the parameter ϕ can be constructed as

$$\left(\hat{\phi} - z_{1-\alpha/2} \hat{s}e(\hat{\phi}), \hat{\phi} + z_{1-\alpha/2} \hat{s}e(\hat{\phi}) \right),$$

where $\hat{\phi}$ is the MLE of ϕ , and $z_{1-\alpha/2}$ is the upper $(\alpha/2)$ -th quantile of the standard normal distribution.

2.3 Approximate Confidence Intervals

When the sample size is small, the distributions of the MLEs of the reliability and the mean-time-to-failure (MTTF) of the devices may not be close to the normal distribution, and consequently confidence intervals constructed directly by the asymptotic method may not maintain the nominal level of confidence. Hoyle [12] discussed various transformations for developing suitable confidence intervals for parameters from a skewed distribution.

Viveros and Balakrishnan [31] used a transformation approach to construct confidence intervals for reliability in the context of start-up demonstration testing. Even when the distribution of the estimate of reliability is skewed in the case of small samples, the bounds for the reliability always fall between 0 and 1 under this approach. Now, by employing a logit transformation, we assume

$$\hat{g} = \ln \left(\frac{\hat{R}(t)}{1 - \hat{R}(t)} \right)$$

to be asymptotically normally distributed with the corresponding standard error, determined by the delta method, as

$$\widehat{se}(\hat{g}) = \frac{\widehat{se}(\hat{R}(t))}{\hat{R}(t)(1 - \hat{R}(t))},$$

where $\widehat{se}(\hat{R}(t))$ is the estimated standard error of $\hat{R}(t)$. The inverse logit transformation gives an approximate $100(1 - \alpha) \%$ confidence interval for the reliability, $R(t)$, to be

$$\left(\frac{\hat{R}(t)}{\hat{R}(t) + (1 - \hat{R}(t))S}, \frac{\hat{R}(t)}{\hat{R}(t) + (1 - \hat{R}(t))/S} \right),$$

where $S = \exp(z_{1-\frac{\alpha}{2}} \widehat{se}(\hat{g}))$.

In a similar manner, Bishop et al. [7] suggested a log-approach for constructing confidence intervals for the MTTF of devices as it ensures positive lower bound for the mean lifetime. In the log-approach, we assume

$$\hat{g} = \ln(\hat{\mu})$$

to be asymptotically normally distributed. The corresponding standard error, by the delta method, is

$$\widehat{se}(\hat{g}) = \frac{\widehat{se}(\hat{\mu})}{\hat{\mu}},$$

where $\widehat{se}(\hat{\mu})$ is the estimated standard error of $\hat{\mu}$. The inverse log transformation gives an approximate $100(1 - \alpha) \%$ confidence interval for the MTTF, μ , as

$$\left(\hat{\mu} \exp\left(\frac{-z_{1-\frac{\alpha}{2}} \widehat{se}(\hat{\mu})}{\hat{\mu}}\right), \hat{\mu} \exp\left(\frac{z_{1-\frac{\alpha}{2}} \widehat{se}(\hat{\mu})}{\hat{\mu}}\right) \right).$$

It should be noted that the computation of the estimated standard errors of the reliability and the MTTF are in (2).

2.4 Parametric Models

Johnson et al. [14, 15] have presented booklength accounts on many lifetime distributions that are useful for reliability analysis. Here, we consider two popular lifetime models in reliability engineering, namely, the Weibull and gamma distributions.

2.4.1 Weibull Distribution

In practice, Weibull distribution is widely used as a lifetime model in engineering and physical sciences. It is also used extensively in biomedical sciences as a proportional hazards model for evaluating the effects of covariates on lifetimes; and in this setting, usually the scale parameter varies with covariates, but the shape parameter remains unchanged over all covariates. Of course, under these assumptions, the Weibull distribution can be parametrized as a proportional hazards model, meaning that the hazard rates of any two products maintain a constant ratio over time. However, the shape parameter of the Weibull distribution may be different for different conditions in general, and so the assumption that the shape parameter does not depend on stress factors or other covariates may often be violated. Many examples show that both the scale and shape parameters of the Weibull distribution vary with covariates ([16, 28, 30]). Meeter and Meeker [24] have presented additional examples of the Weibull distribution with unequal shape parameters while modeling lifetimes of devices.

We assume that the lifetimes of the devices in the i -th group that experience at the same stress level \mathbf{x}_i , $\{t_{ik}, i = 1, 2, \dots, I, k = 1, 2, \dots, K_i\}$, follow a Weibull distribution with corresponding pdf and cdf

$$f_T(t) = \frac{\eta_i t^{\eta_i - 1}}{\alpha_i^{\eta_i}} \exp\left(-\left(\frac{t}{\alpha_i}\right)^{\eta_i}\right), \quad t > 0,$$

and

$$F_T(t) = 1 - \exp\left(-\left(\frac{t}{\alpha_i}\right)^{\eta_i}\right), \quad t > 0,$$

where $\alpha_i > 0$ and $\eta_i > 0$ are the scale and shape parameters, respectively. We assume that both of these parameters relate to the stress levels in log-linear forms as

$$\alpha_i = \exp \left(\sum_{j=0}^J a_j x_{ij} \right) \quad \text{and} \quad \eta_i = \exp \left(\sum_{j=0}^J b_j x_{ij} \right).$$

Note that $x_{i0} = 1$, for all i , corresponding to constant effects on the scale and shape parameters in the model. We denote the model parameters that are to be estimated by $\theta = (\mathbf{a}, \mathbf{b}) = (a_0, a_1, \dots, a_J, b_0, b_1, \dots, b_J)$.

Instead of working with Weibull lifetimes, it is often more convenient to work with the extreme value distribution for the log-lifetimes $w_{ik} = \ln(t_{ik})$; see, for example, Meeter and Meeker [24], and Ng et al. [27]. Therefore, we consider here the extreme value distribution with the corresponding pdf and cdf as

$$f_W(w) = \frac{1}{\sigma_i} \exp(\xi_i) \exp(-\exp(\xi_i)), \quad -\infty < w < \infty,$$

and

$$F_W(w) = 1 - \exp(-\exp(\xi_i)), \quad -\infty < w < \infty,$$

where $\xi_i = (w - \delta_i)/\sigma_i$, $\delta_i = \ln(\alpha_i) = \sum_{j=0}^J a_j x_{ij}$, and $\sigma_i = \eta_i^{-1} = \exp\left(-\sum_{j=0}^J b_j x_{ij}\right)$.

Then, the corresponding reliability at mission time t and MTTF under normal operating conditions, \mathbf{x}_0 , are given by

$$R(t; \mathbf{x}_0) = \exp(-\exp(\xi)) \tag{3}$$

and

$$\mu(\mathbf{x}_0) = \exp(\delta) \Gamma(1 + \sigma), \tag{4}$$

respectively, where $\xi = (\ln(t) - \delta)/\sigma$, $\delta = \sum_{j=0}^J a_j x_{0j}$, $\sigma = \exp\left(-\sum_{j=0}^J b_j x_{0j}\right)$, and $\Gamma(\cdot)$ denotes the complete gamma function.

In the EM algorithm, the log-likelihood function based on the complete data can be expressed in this case as

$$\ell_c(\theta) = \sum_{i=1}^I \sum_{k=1}^{K_i} (-\ln(\sigma_i) + \xi_{ik} - \exp(\xi_{ik})) + \ln(C). \tag{5}$$

In the M-step, for $j = 0, 1, \dots, J$, taking the first-order derivatives of the log-likelihood function in (5) with respect to the model parameters a_j and b_j , we obtain the likelihood equations

$$\frac{\partial Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial a_j} = \sum_{i=1}^I \left(\frac{K_i x_{ij}}{\sigma_i} \right) (-1 + E_{\boldsymbol{\theta}^{(m)}}[\exp(\xi_i) | \mathbf{z}]),$$

$$\frac{\partial Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial b_j} = \sum_{i=1}^I K_i x_{ij} (1 + E_{\boldsymbol{\theta}^{(m)}}[\xi_i | \mathbf{z}] - E_{\boldsymbol{\theta}^{(m)}}[\xi_i \exp(\xi_i) | \mathbf{z}]).$$

Then, the required second-order derivatives of the log-likelihood function with respect to the model parameters are

$$\frac{\partial^2 Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial a_p \partial a_q} = - \sum_{i=1}^I \left(\frac{K_i x_{ip} x_{iq}}{\sigma_i^2} \right) E_{\boldsymbol{\theta}^{(m)}}[\exp(\xi_i) | \mathbf{z}],$$

$$\frac{\partial^2 Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial b_p \partial b_q} = - \sum_{i=1}^I K_i x_{ip} x_{iq} E_{\boldsymbol{\theta}^{(m)}}[\xi_i \exp(\xi_i) + \xi_i^2 \exp(\xi_i) - \xi_i | \mathbf{z}],$$

$$\frac{\partial^2 Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial a_p \partial b_q} = - \sum_{i=1}^I \left(\frac{K_i x_{ip} x_{iq}}{\sigma_i} \right) (1 - E_{\boldsymbol{\theta}^{(m)}}[\exp(\xi_i) + \xi_i \exp(\xi_i) | \mathbf{z}]),$$

for $p = 0, 1, \dots, J$, and $q = 0, 1, \dots, J$.

This calculation would require four conditional expectations to be considered in the following E-step. Given the updated model parameters $\boldsymbol{\theta}' = \boldsymbol{\theta}^{(m+1)}$, let $v_i = \ln(IT_i)$, $\zeta_i = \exp((v_i - \delta_i)/\sigma_i)$ and $S_W(v_i; \boldsymbol{\theta}') = 1 - F_W(v_i; \boldsymbol{\theta}')$. Then, the four conditional expectations of interest can be derived as

$$\begin{aligned} E_{\boldsymbol{\theta}'}[\xi_i | \mathbf{z}] &= \frac{n_i}{K_i F_W(v_i; \boldsymbol{\theta}')} \int_0^{\zeta_i} \ln(t) \exp(-t) dt \\ &\quad + \frac{K_i - n_i}{K_i S_W(v_i; \boldsymbol{\theta}')} \int_{\zeta_i}^{\infty} \ln(t) \exp(-t) dt \\ &= \frac{n_i}{K_i F_W(v_i; \boldsymbol{\theta}')} \left(-\gamma - \ln(\zeta_i) \exp(-\zeta_i) - \int_{\zeta_i}^{\infty} \frac{\exp(-t)}{t} dt \right) \\ &\quad + \frac{K_i - n_i}{K_i S_W(v_i; \boldsymbol{\theta}')} \left(\ln(\zeta_i) \exp(-\zeta_i) + \int_{\zeta_i}^{\infty} \frac{\exp(-t)}{t} dt \right), \end{aligned}$$

$$\begin{aligned} E_{\boldsymbol{\theta}'}[\exp(\xi_i) | \mathbf{z}] &= \frac{n_i}{K_i F_W(v_i; \boldsymbol{\theta}')} \int_0^{\zeta_i} t \exp(-t) dt \\ &\quad + \frac{K_i - n_i}{K_i S_W(v_i; \boldsymbol{\theta}')} \int_{\zeta_i}^{\infty} t \exp(-t) dt \\ &= \frac{n_i}{K_i F_W(v_i; \boldsymbol{\theta}')} (1 - \exp(-\zeta_i) - \zeta_i \exp(-\zeta_i)) \end{aligned}$$

$$\begin{aligned}
 & + \frac{K_i - n_i}{K_i S_W(v_i; \theta')} (\exp(-\zeta_i) + \zeta_i \exp(-\zeta_i)), \\
 E_{\theta'}[\xi_i \exp(\xi_i) | \mathbf{z}] &= \frac{n_i}{K_i F_W(v_i; \theta')} \int_0^{\zeta_i} t \ln(t) d(-\exp(-t)) \\
 & + \frac{K_i - n_i}{K_i S_W(v_i; \theta')} \int_{\zeta_i}^{\infty} t \ln(t) d(-\exp(-t)) \\
 &= \frac{n_i}{K_i F_W(v_i; \theta')} \left(1 - \gamma - A - \int_{\zeta_i}^{\infty} \frac{\exp(-t)}{t} dt \right) \\
 & + \frac{K_i - n_i}{K_i S_W(v_i; \theta')} \left(A + \int_{\zeta_i}^{\infty} \frac{\exp(-t)}{t} dt \right), \\
 E_{\theta'}[\xi_i^2 \exp(\xi_i) | \mathbf{z}] &= \frac{n_i}{K_i F_W(v_i; \theta')} \int_0^{\zeta_i} t (\ln(t))^2 \exp(-t) dt \\
 & + \frac{K_i - n_i}{K_i S_W(v_i; \theta')} \int_{\zeta_i}^{\infty} t (\ln(t))^2 \exp(-t) dt \\
 &= \frac{n_i B}{K_i F_W(v_i; \theta')} + \frac{K_i - n_i}{K_i S_W(v_i; \theta')} \left(\gamma^2 + \frac{\pi^2}{6} - 2\gamma - B \right),
 \end{aligned}$$

where

$$\begin{aligned}
 A &= \exp(-\zeta_i) + \ln(\zeta_i) \exp(-\zeta_i) + \zeta_i \ln(\zeta_i) \exp(-\zeta_i), \\
 B &= \sum_{m=0}^{\infty} \frac{(-\zeta_i)^{m+2}}{m!(m+2)} \left((\ln(\zeta_i))^2 - \frac{2 \ln(\zeta_i)}{m+2} + \frac{2}{(m+2)^2} \right),
 \end{aligned}$$

$\int_x^{\infty} \exp(-t)/t dt$ is the exponential integral that can be readily computed by mathematical programs such as MATLAB and Maple, and $\gamma \approx 0.577215665$ is Euler's constant.

Given the information on normal operating conditions, \mathbf{x} , we can also estimate the reliability at mission time t as well as the MTTF of devices by substituting the MLEs of the model parameters $\hat{\theta}$ into (3) and (4), respectively. In addition, to construct confidence intervals for $R(t; \mathbf{x})$ and $\mu(\mathbf{x})$, the required first-order derivatives of corresponding functions with respect to a_j and b_j are given by

$$\begin{aligned}
 \frac{\partial R(t; \mathbf{x})}{\partial a_j} &= -\frac{x_j}{\sigma} R(t; \mathbf{x}) \ln(R(t; \mathbf{x})), \\
 \frac{\partial R(t; \mathbf{x})}{\partial b_j} &= x_j R(t; \mathbf{x}) \ln(R(t; \mathbf{x})) \ln(-\ln(R(t; \mathbf{x}))), \\
 \frac{\partial \mu(\mathbf{x})}{\partial a_j} &= x_j \mu(\mathbf{x}), \quad \frac{\partial \mu(\mathbf{x})}{\partial b_j} = -x_j \sigma \Psi(1 + \sigma) \mu(\mathbf{x}),
 \end{aligned}$$

where $\Psi(z) = d \ln \Gamma(z)/dz$ is the digamma function.

2.4.2 Gamma Distribution

Gamma distribution is commonly used for fitting lifetime data in reliability and survival studies due to its flexibility. Its hazard function can be increasing, decreasing, or constant. When the hazard function of gamma distribution is a constant, it corresponds to the exponential distribution. In addition to the exponential distribution, the gamma distribution also includes the chi-square distribution as a special case. The gamma distribution has found a number of applications in different fields. To quote a few, Husak et al. [13] used the gamma distribution to describe monthly rainfall in Africa for the management of water and agricultural resources, as well as food reserves. Kwon and Frangopol [17] assessed and predicted bridge fatigue reliabilities of two existing bridges, the Neville Island Bridge and the Birmingham Bridge, based on long-term monitoring data. They applied log-normal, Weibull, and gamma distributions to estimate the mean and standard deviation of the stress range.

We now assume that the lifetimes of the devices in the i -th group that experience at the same stress level, \mathbf{x}_i , $\{t_{ik}, i = 1, 2, \dots, I, k = 1, 2, \dots, K_i\}$, follow gamma distribution with shape parameter $\alpha_i > 0$ and scale parameter $\beta_i > 0$. The corresponding pdf and cdf are given by

$$f(t) = \frac{t^{\alpha_i-1}}{\Gamma(\alpha_i)\beta_i^{\alpha_i}} \exp\left(-\frac{t}{\beta_i}\right), \quad t > 0,$$

and

$$F(t) = \int_0^t \frac{y^{\alpha_i-1}}{\Gamma(\alpha_i)\beta_i^{\alpha_i}} \exp\left(-\frac{y}{\beta_i}\right) dy, \quad t > 0,$$

respectively. The cdf $F(t)$ can be readily expressed in terms of the lower incomplete gamma ratio as

$$F(t) = \int_0^{\frac{t}{\beta_i}} \frac{y^{\alpha_i-1}}{\Gamma(\alpha_i)} \exp(-y) dy = \gamma\left(\alpha_i, \frac{t}{\beta_i}\right).$$

We assume that both shape and scale parameters are related to the stress factors in log-linear forms as

$$\alpha_i = \exp\left(\sum_{j=0}^J a_j x_{ij}\right) \quad \text{and} \quad \beta_i = \exp\left(\sum_{j=0}^J b_j x_{ij}\right),$$

where $x_{i0} = 1$, for all i .

Let $\boldsymbol{\theta} = (\mathbf{a}, \mathbf{b}) = (a_0, a_1, \dots, a_J, b_0, b_1, \dots, b_J)$, denote the model parameters to be estimated and the corresponding reliability at mission time t and the MTTF under normal operating conditions, \mathbf{x}_0 , are given by

$$R(t; \mathbf{x}_0) = \Gamma \left(\alpha, \frac{t}{\beta} \right) \quad (6)$$

and

$$\mu(\mathbf{x}_0) = \alpha\beta, \quad (7)$$

respectively, where $\alpha = \exp \left(\sum_{j=0}^J a_j x_{0j} \right)$, $\beta = \exp \left(\sum_{j=0}^J b_j x_{0j} \right)$, and $\Gamma(\alpha, t/\beta) = 1 - \gamma(\alpha, t/\beta)$ is the upper incomplete gamma ratio.

In the EM algorithm, the log-likelihood function based on the complete data can be expressed as

$$\ell_c(\boldsymbol{\theta}) = \sum_{i=1}^I \sum_{k=1}^{K_i} \left(-\ln \Gamma(\alpha_i) + \alpha_i \ln \left(\frac{t_{ik}}{\beta_i} \right) - \ln(t_{ik}) - \frac{t_{ik}}{\beta_i} \right) + \ln(C). \quad (8)$$

In the M-step, for $j = 0, 1, \dots, J$, taking the first-order derivatives of the log-likelihood function in (8) with respect to the model parameters a_j and b_j , we obtain the likelihood equations as

$$\begin{aligned} \frac{\partial Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial a_j} &= - \sum_{i=1}^I K_i x_{ij} \alpha_i \left(\psi(\alpha_i) - E_{\theta^{(m)}} \left[\ln \left(\frac{T_i}{\beta_i} \right) \middle| \mathbf{z} \right] \right), \\ \frac{\partial Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial b_j} &= - \sum_{i=1}^I K_i x_{ij} \left(\alpha_i - E_{\theta^{(m)}} \left[\frac{T_i}{\beta_i} \middle| \mathbf{z} \right] \right). \end{aligned}$$

Furthermore, the required second-order derivatives of the log-likelihood function with respect to the parameters a_j and b_j are

$$\begin{aligned} \frac{\partial^2 Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial a_p \partial a_q} &= - \sum_{i=1}^I K_i x_{ip} x_{iq} \alpha_i \left(\alpha_i \psi'(\alpha_i) + \psi(\alpha_i) - E_{\theta^{(m)}} \left[\ln \left(\frac{T_i}{\beta_i} \right) \middle| \mathbf{z} \right] \right), \\ \frac{\partial^2 Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial b_p \partial b_q} &= - \sum_{i=1}^I K_i x_{ip} x_{iq} E_{\theta^{(m)}} \left[\frac{T_i}{\beta_i} \middle| \mathbf{z} \right], \\ \frac{\partial^2 Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})}{\partial a_p \partial b_q} &= - \sum_{i=1}^I K_i x_{ip} x_{iq} \alpha_i, \end{aligned}$$

for $p = 0, 1, \dots, J$ and $q = 0, 1, \dots, J$, where $\psi'(z) = d\psi(z)/dz$ is the trigamma function.

This calculation would require two conditional expectations to be considered in the following E-step. Given the updated model parameters $\boldsymbol{\theta}' = \boldsymbol{\theta}^{(m+1)}$, the two conditional expectations of interest can be derived as follows:

$$\begin{aligned}
 E_{\theta'} \left[\frac{T_i}{\beta_i} \mid \mathbf{z} \right] &= \left(\frac{n_i}{K_i} \right) E_{\theta'} \left[\frac{T}{\beta_i} \mid T < IT_i \right] + \left(1 - \frac{n_i}{K_i} \right) E_{\theta'} \left[\frac{T}{\beta_i} \mid T > IT_i \right] \\
 &= \alpha'_i \left(1 - \frac{\left(\frac{n_i}{K_i} \right) \left(\frac{IT_i}{\beta'_i} \right)^{\alpha'_i} \exp \left(-\frac{IT_i}{\beta'_i} \right)}{\Gamma \left(\alpha'_i + 1 \right) \gamma \left(\alpha'_i, \frac{IT_i}{\beta'_i} \right)} \right. \\
 &\quad \left. + \frac{\left(1 - \frac{n_i}{K_i} \right) \left(\frac{IT_i}{\beta'_i} \right)^{\alpha'_i} \exp \left(-\frac{IT_i}{\beta'_i} \right)}{\Gamma \left(\alpha'_i + 1 \right) \Gamma \left(\alpha'_i, \frac{IT_i}{\beta'_i} \right)} \right), \\
 E_{\theta'} \left[\ln \left(\frac{T_i}{\beta_i} \right) \mid \mathbf{z} \right] &= \left(\frac{n_i}{K_i} \right) E_{\theta'} \left[\ln \left(\frac{T}{\beta_i} \right) \mid T < IT_i \right] \\
 &\quad + \left(1 - \frac{n_i}{K_i} \right) E_{\theta'} \left[\ln \left(\frac{T}{\beta_i} \right) \mid T > IT_i \right] \\
 &= \left(\frac{n_i}{K_i} \right) \frac{H_1 \left(\alpha'_i, \frac{IT_i}{\beta'_i} \right)}{\gamma \left(\alpha'_i, \frac{IT_i}{\beta'_i} \right)} + \left(1 - \frac{n_i}{K_i} \right) \frac{\psi \left(\alpha'_i \right) - H_1 \left(\alpha'_i, \frac{IT_i}{\beta'_i} \right)}{\Gamma \left(\alpha'_i, \frac{IT_i}{\beta'_i} \right)},
 \end{aligned}$$

where

$$\begin{aligned}
 H_1(\alpha, \beta) &= \frac{1}{\Gamma(\alpha)} \int_0^\beta \ln(x) x^{\alpha-1} \exp(-x) dx \\
 &= \ln(\beta) \gamma(\alpha, \beta) - \frac{\beta^\alpha {}_2F_2(\alpha, \alpha; 1 + \alpha, 1 + \alpha; -\beta)}{\alpha^2 \Gamma(\alpha)},
 \end{aligned}$$

and ${}_nF_m(a_1, a_2, \dots, a_n; b_1, b_2, \dots, b_m; z)$ is the Gaussian hypergeometric function that can be computed using MATLAB.

Given the information on normal operating conditions, \mathbf{x} , we can also estimate the reliability at mission time t as well as the MTTF of devices by substituting the MLEs of the model parameters $\hat{\theta}$ into (6) and (7), respectively. In addition, to construct confidence intervals for $R(t; \mathbf{x})$ and $\mu(\mathbf{x})$, the required first-order derivatives of corresponding functions with respect to a_j and b_j are

$$\begin{aligned}
 \frac{\partial R(t; \mathbf{x})}{\partial a_j} &= x_j \alpha \left\{ \psi(\alpha) \gamma \left(\alpha, \frac{t}{\beta} \right) - H_1 \left(\alpha, \frac{t}{\beta} \right) \right\}, \\
 \frac{\partial R(t; \mathbf{x})}{\partial b_j} &= \frac{x_j}{\Gamma(\alpha)} \left(\frac{t}{\beta} \right)^\alpha \exp \left\{ -\left(\frac{t}{\beta} \right) \right\}, \\
 \frac{\partial \mu(\mathbf{x})}{\partial a_j} &= x_j \alpha \beta, \quad \frac{\partial \mu(\mathbf{x})}{\partial b_j} = x_j \alpha \beta.
 \end{aligned}$$

It is noted that the estimates by the EM algorithm are not attainable in the situation when $n_i = 0$ for all i . In such a situation, we obtain no information on the reliability.

2.5 Goodness of Fit Test

To test the suitability of an assumed model for the observed data, a distance-based test statistic of the form

$$M = \max_i |n_i - K_i \hat{F}(IT_i)| \tag{9}$$

can be used as a discrepancy measure. The distance-based test statistic M in (9) simply quantifies the distance between the observed and the expected number of failures at each testing condition, and so when the assumed model does not fit the data, we would expect to observe a large value of the test statistic M . Note that in each testing condition, the number of failures, n_i , has a binomial distribution with $(K_i, \hat{F}(IT_i))$, and so we can calculate the exact p -value of this test as follows:

$$\begin{aligned} p_v &= Pr \left(\max_i |n_i - \hat{n}_i| > M \right) \\ &= 1 - Pr \left(\max_i |n_i - \hat{n}_i| \leq M \right) \\ &= 1 - Pr (|n_i - \hat{n}_i| \leq M \text{ for all } i) \\ &= 1 - \prod_{i=1}^I Pr (|n_i - \hat{n}_i| \leq M) \\ &= 1 - \prod_{i=1}^I \left(\sum_{n=\max(0, \lfloor \hat{n}_i - M + 1 \rfloor}^{\min(K_i, \lceil \hat{n}_i + M - 1 \rceil)} \binom{K_i}{n} \hat{F}(IT_i)^n (1 - \hat{F}(IT_i))^{K_i - n} \right), \tag{10} \end{aligned}$$

where $\hat{n}_i = K_i \hat{F}(IT_i)$. If the p -value determined is small, i.e., p -value < 0.05 or 0.1 , then we can conclude that the data do not provide enough evidence for the assumed model.

3 Competing Risk Models

One-shot devices often have multiple components that can cause failure. For example, a fire extinguisher contains a cylinder, a valve and chemicals inside; an automobile air bag contains a crash sensor, an inflator and an air bag; and for any packed food (which is also a kind of one-shot device), there are different causes for food expiry such as the growth of microorganisms in the package, the moisture level and the food deterioration due to oxidation. A failure of any of the components will result in the failure of the product. For those failed units, one will normally determine the cause responsible for the failure. Thus, the information collected from a life-test on one-shot devices in this case will include the status of the unit at inspection time as well as the cause of failure in case the unit did fail. With competing risks, the model

becomes more complicated than those considered earlier and so the corresponding estimation problem becomes quite complex. However, this competing risks model is more realistic since many one-shot devices do contain multiple components that could cause the failure of the device. This motivates us to consider the one-shot device model with competing risks here.

3.1 Likelihood Function

An accelerated life test for such one-shot devices is set up as follows:

1. the tests are only checked at inspection times IT_i , for $i = 1, \dots, I$,
2. the devices are tested under different stress levels w_j , for $j = 1, \dots, J$,
3. there are K_{ij} devices tested at IT_i and w_j ,
4. the number of devices failed due to the r -th cause at IT_i and w_j is denoted by D_{rij} , for $r = 1, \dots, R$,
5. the number of devices that function at IT_i and w_j is denoted by $S_{ij} = K_{ij} - \sum_{r=1}^R D_{rij}$.

Let us now assume that there are only two causes responsible for the failure, cause 1 and cause 2, and denote the random variable for the failure time due to causes 1 and 2 by T_{rijk} , for $r = 1, 2, i = 1, \dots, I, j = 1, \dots, J$ and $k = 1, \dots, K_{ij}$, respectively. In this work, we assume that T_{rijk} are independent of each other and follow an exponential distribution with rate parameter λ_{rj} with pdf

$$f_{rj}(t) = \lambda_{rj} \exp(-\lambda_{rj}t), \quad r = 1, 2, \quad j = 1, \dots, J,$$

and with cdf

$$F_r(IT_i|w_j) = 1 - \exp(-\lambda_{rj}IT_i), \quad r = 1, 2, \quad i = 1, \dots, I, \quad j = 1, \dots, J,$$

where λ_{rj} is the failure rate of the r -th component in the device under stress level w_j . Of course, t_{rijk} will be used to denote the realization of the variable T_{rijk} . The relationship between λ_{rj} and w_j is assumed to be a log-link function of the form

$$\lambda_{rj} = \alpha_{r0} \exp(\alpha_{r1}w_j), \quad \alpha_{r0}, \alpha_{r1}, w_j \geq 0. \tag{11}$$

We define Δ_{ijk} to be the indicator for the k -th device under stress level w_j and inspection time IT_i . When the device functions we will set $\Delta_{ijk} = 0$. However, if the device does not function, we will identify (by careful inspection) the specific cause responsible for the failure. If risk r is the cause for the failure, we will denote this event by $\Delta_{ijk} = r$, for $r = 1, 2$. Mathematically, the indicator Δ_{ijk} is then defined as

$$\Delta_{ijk} = \begin{cases} 0 & \text{if } \min(T_{1ijk}, T_{2ijk}) > IT_i \\ 1 & \text{if } T_{1ijk} < \min(T_{2ijk}, IT_i) \\ 2 & \text{if } T_{2ijk} < \min(T_{1ijk}, IT_i) \end{cases}$$

and then δ_{ijk} will be used for the realization of Δ_{ijk} .

We also denote p_{0ij} , p_{1ij} and p_{2ij} for the survival probability, failure probability due to cause 1 and failure probability due to cause 2, respectively, which are

$$p_{0ij} = (1 - F_1(IT_i|w_j))(1 - F_2(IT_i|w_j)) = \exp(-(\lambda_{1j} + \lambda_{2j})IT_i), \quad (12)$$

$$p_{1ij} = \left(\frac{\lambda_{1j}}{\lambda_{1j} + \lambda_{2j}} \right) (1 - \exp(-(\lambda_{1j} + \lambda_{2j})IT_i)), \quad (13)$$

$$p_{2ij} = \left(\frac{\lambda_{2j}}{\lambda_{1j} + \lambda_{2j}} \right) (1 - \exp(-(\lambda_{1j} + \lambda_{2j})IT_i)). \quad (14)$$

Now the data collected at stress levels $\mathbf{w} = \{w_j, j = 1, 2, \dots, J\}$ and inspection times $\mathbf{IT} = \{IT_i, i = 1, \dots, I\}$ are the numbers of devices with the indicator values $\delta_{ijk} = 0$, $\delta_{ijk} = 1$ and $\delta_{ijk} = 2$, which are denoted by S_{ij} , D_{1ij} and D_{2ij} , respectively. Then, the likelihood function of $\boldsymbol{\alpha} = \{\alpha_{10}, \alpha_{11}, \alpha_{20}, \alpha_{21}\}$ is given by

$$L(\boldsymbol{\alpha}|\boldsymbol{\delta}, \mathbf{IT}, \mathbf{w}) = \prod_{i=1}^I \prod_{j=1}^J p_{0ij}^{S_{ij}} p_{1ij}^{D_{1ij}} p_{2ij}^{D_{2ij}}.$$

3.2 EM Algorithm

In the example of one-shot devices, the parameters of interest are $\alpha_{r0}, \alpha_{r1}, r = 1, 2$, and the data that are not observable are the true lifetimes of the devices. Let us denote that lifetime by $T_{rijk}^{(\Delta_{ijk})}$ defined as

$$T_{rijk}^{(\Delta_{ijk})} = \begin{cases} T_{rijk} & \text{if } \min(T_{1ijk}, T_{2ijk}) > IT_i, \text{ when } \Delta_{ijk} = 0 \\ T_{rijk} & \text{if } T_{1ijk} < \min(T_{2ijk}, IT_i), \text{ when } \Delta_{ijk} = 1 \\ T_{rijk} & \text{if } T_{2ijk} < \min(T_{1ijk}, IT_i), \text{ when } \Delta_{ijk} = 2 \end{cases}.$$

Let us denote $\boldsymbol{\alpha} = (\alpha_{10}, \alpha_{11}, \alpha_{20}, \alpha_{21})$ and $\boldsymbol{\alpha}'$ for the current estimate of $\boldsymbol{\alpha}$. Then, the complete data likelihood is given by

$$\begin{aligned} l^c(\boldsymbol{\alpha}) &= \sum_{i=1}^I \sum_{j=1}^J \sum_{k=1}^{K_{ij}} \ln \left(f_{1j}(T_{1ijk}^{(\delta_{ijk})}) \right) + \ln \left(f_{2j}(T_{2ijk}^{(\delta_{ijk})}) \right) \\ &= \sum_{i=1}^I \sum_{j=1}^J l_{ij}^c(\boldsymbol{\alpha}), \end{aligned}$$

where

$$\begin{aligned}
 l_{ij}^c(\boldsymbol{\alpha}) &= \sum_{k=1}^{K_{ij}} \ln(\lambda_{1j}) - \lambda_{1j} T_{1ijk}^{(\delta_{ik})} + \ln(\lambda_{2j}) - \lambda_{2j} T_{2ijk}^{(\delta_{ik})} \\
 &= K_{ij} (\ln(\lambda_{1j}) + \ln(\lambda_{2j})) - \lambda_{1j} \sum_{k=1}^{K_{ij}} T_{1ijk}^{(\delta_{ik})} - \lambda_{2j} \sum_{k=1}^{K_{ij}} T_{2ijk}^{(\delta_{ik})}.
 \end{aligned}$$

In the E-step of the EM algorithm, we shall take the expected value of the missing data, given the observed data and the current parameter estimates, to approximate the missing data. It is given by

$$\begin{aligned}
 E(l_{ij}^c(\boldsymbol{\alpha})|\boldsymbol{\alpha}') &= K_{ij} (\ln(\lambda_{1j}) + \ln(\lambda_{2j})) \\
 &\quad - \lambda_{1j} \left(S_{ij} E(T_{1ijk}^{(0)}|\boldsymbol{\alpha}') + D_{1ij} E(T_{1ijk}^{(1)}|\boldsymbol{\alpha}') + D_{2ij} E(T_{1ijk}^{(2)}|\boldsymbol{\alpha}') \right) \\
 &\quad - \lambda_{2j} \left(S_{ij} E(T_{2ijk}^{(0)}|\boldsymbol{\alpha}') + D_{1ij} E(T_{2ijk}^{(1)}|\boldsymbol{\alpha}') + D_{2ij} E(T_{2ijk}^{(2)}|\boldsymbol{\alpha}') \right),
 \end{aligned}$$

where the conditional expectations, $E\left(T_{rijk}^{(\delta_{jk})} \mid \boldsymbol{\alpha}'\right)$, are as presented in Table 2. It is important to note here that the expectations are in simple closed-form since the lifetimes here are assumed to follow an exponential distribution.

Then, the partial objective function $Q_{ij}(\boldsymbol{\alpha}, \boldsymbol{\alpha}')$ for $l_{ij}^c(\boldsymbol{\alpha})$ is

$$Q_{ij}(\boldsymbol{\alpha}, \boldsymbol{\alpha}') = K_{ij} (\ln(\lambda_{1j}) + \ln(\lambda_{2j})) - \lambda_{1j} g_{1ij}(\boldsymbol{\alpha}') - \lambda_{2j} g_{2ij}(\boldsymbol{\alpha}'), \quad (15)$$

Table 2 Conditional expected values of missing data, with $\lambda'_{rj} = \alpha'_{r0} \exp(\alpha'_{r1})$, $r = 1, 2$, for different cases

Case	No. of cases	$E\left(T_{rijk}^{(\delta_{ijk})} \mid \boldsymbol{\alpha}'\right)$
$\delta_{ijk} = 0$	S_{ij}	$IT_i + \frac{1}{\lambda'_{1j}}$
$\delta_{ijk} = 1$	D_{1ij}	$\frac{1}{\lambda'_{1j} + \lambda'_{2j}} - \frac{IT_i \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}{1 - \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}$
$\delta_{ijk} = 2$	D_{2ij}	$\frac{1}{\lambda'_{1j}} + \frac{1}{\lambda'_{1j} + \lambda'_{2j}} - \frac{IT_i \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}{1 - \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}$
Case	No. of cases	$E\left(T_{2ijk}^{(\delta_{ijk})} \mid \boldsymbol{\alpha}'\right)$
$\delta_{ijk} = 0$	S_{ij}	$IT_i + \frac{1}{\lambda'_{2j}}$
$\delta_{ijk} = 1$	D_{1ij}	$\frac{1}{\lambda'_{2j}} + \frac{1}{\lambda'_{1j} + \lambda'_{2j}} - \frac{IT_i \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}{1 - \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}$
$\delta_{ijk} = 2$	D_{2ij}	$\frac{1}{\lambda'_{1j} + \lambda'_{2j}} - \frac{IT_i \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}{1 - \exp(-(\lambda'_{1j} + \lambda'_{2j})IT_i)}$

where

$$g_{rij}(\boldsymbol{\alpha}') = \left(S_{ij} E(T_{rijk}^{(0)} | \boldsymbol{\alpha}') + D_{1ij} E(T_{rijk}^{(1)} | \boldsymbol{\alpha}') + D_{2ij} E(T_{rijk}^{(2)} | \boldsymbol{\alpha}') \right),$$

$$g_{2ij}(\boldsymbol{\alpha}') = \left(S_{ij} E(T_{2ijk}^{(0)} | \boldsymbol{\alpha}') + D_{1ij} E(T_{2ijk}^{(1)} | \boldsymbol{\alpha}') + D_{2ij} E(T_{2ijk}^{(2)} | \boldsymbol{\alpha}') \right).$$

From (15), the objective function to maximize the overall likelihood function $l^c(\boldsymbol{\alpha})$ will be the summation of the partial objective functions, $Q_{ij}(\boldsymbol{\alpha}, \boldsymbol{\alpha}')$, $i = 1, \dots, I, j = 1, \dots, J$, given by

$$Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}') = \sum_{i=1}^I \sum_{j=1}^J Q_{ij}(\boldsymbol{\alpha}, \boldsymbol{\alpha}').$$

Upon substituting the link function for the failure rates specified in (11), and then differentiating the objective function with respect to $\alpha_{r0}, \alpha_{r1}, r = 1, 2$, we obtain the first-order derivatives of $Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}')$

$$\frac{\partial Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}')}{\partial \alpha_{r0}} = \frac{\sum_{i=1}^I \sum_{j=1}^J K_{ij}}{\alpha_{r0}} - \sum_{i=1}^I \sum_{j=1}^J \exp(\alpha_{r1} w_j) g_{rij}(\boldsymbol{\alpha}'),$$

$$\frac{\partial Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}')}{\partial \alpha_{r1}} = \sum_{i=1}^I \sum_{j=1}^J K_{ij} w_j - \sum_{i=1}^I \sum_{j=1}^J \alpha_{r0} w_j \exp(\alpha_{r1} w_j) g_{rij}(\boldsymbol{\alpha}'),$$

for $j = 0, 1, \dots, J$.

It is evident that the first-order derivatives involve nonlinear terms, and so we have to find the estimates by numerical methods. To solve the likelihood equations, we consider the updating equations

$$\alpha_{r1}^{(h+1)} = \alpha_{r1}^{(h)} - \frac{\sum_{i=1}^I \sum_{j=1}^J c_j \exp(\alpha_{r1}^{(h)} w_j) g_{rij}(\boldsymbol{\alpha}')}{\sum_{i=1}^I \sum_{j=1}^J c_j w_j \exp(\alpha_{r1}^{(h)} w_j) g_{rij}(\boldsymbol{\alpha}')},$$

$$\hat{\alpha}_{r0} = \frac{\sum_{i=1}^I \sum_{j=1}^J K_{ij}}{\sum_{i=1}^I \sum_{j=1}^J \exp(\hat{\alpha}_{r1} w_j) g_{rij}(\boldsymbol{\alpha}')},$$

where $r = 1, 2$ and $c_j = \left(w_j - \frac{\sum_{i=1}^I \sum_{j=1}^J K_{ij} w_j}{\sum_{i=1}^I \sum_{j=1}^J K_{ij}} \right)$.

3.2.1 Goodness of Fit Test

We can measure the goodness of fit of the proposed model using a distance-based test statistic of the form

$$M = \max_{ij} (|S_{ij} - \hat{S}_{ij}|, |D_{1ij} - \hat{D}_{1ij}|, |D_{2ij} - \hat{D}_{2ij}|),$$

where $i = 1, \dots, I, j = 1, \dots, J$, and $(\hat{S}_{ij}, \hat{D}_{rij})$ are the expected numbers of successes and failures based on the proposed model. The statistic M quantifies the distance between the fitted model and the observed data. If the model does not fit the data, the distance M will be large. If the model is true, $(S_{ij}, D_{1ij}, D_{2ij})$ should follow the multinomial distribution with probabilities close to $(\hat{p}_{0ij}, \hat{p}_{1ij}, \hat{p}_{2ij})$, which are obtained from (12)–(14) with parameters replaced by the corresponding estimates. Then, the exact p -value of the test can be found as follows:

$$\begin{aligned} pv &= Pr(\max_{ij} (|S_{ij} - \hat{S}_{ij}|, |D_{1ij} - \hat{D}_{1ij}|, |D_{2ij} - \hat{D}_{2ij}|) > M) \\ &= 1 - Pr(\max_{ij} (|S_{ij} - \hat{S}_{ij}|, |D_{1ij} - \hat{D}_{1ij}|, |D_{2ij} - \hat{D}_{2ij}|) \leq M) \\ &= 1 - \prod_{i,j} Pr(\{|S_{ij} - \hat{S}_{ij}| \leq M\} \cap \{|D_{1ij} - \hat{D}_{1ij}| \leq M\} \cap \{|D_{2ij} - \hat{D}_{2ij}| \leq M\}) \\ &= 1 - \prod_{i,j} \left(\sum_{D_{2ij}=b_{2ij}^l}^{b_{2ij}^u} \sum_{D_{1ij}=b_{1ij}^l(D_{2ij})}^{b_{1ij}^u(D_{2ij})} \frac{K_{ij}!}{S_{ij}!D_{1ij}!D_{2ij}!} \hat{p}_{0ij}^{S_{ij}} \hat{p}_{1ij}^{D_{1ij}} \hat{p}_{2ij}^{D_{2ij}} \right), \end{aligned}$$

where $S_{ij} = K_{ij} - D_{1ij} - D_{2ij}$, $b_{ij}^l = \max(0, \lceil \hat{D}_{1ij} - M \rceil, \lceil \hat{D}_{1ij} + \hat{D}_{2ij} - M \rceil - D_{2ij})$, $b_{1ij}^u(D_{2ij}) = \min(K_{ij} - D_{2ij}, \lfloor \hat{D}_{1ij} + M \rfloor, \lfloor \hat{D}_{1ij} + \hat{D}_{2ij} + M \rfloor - D_{2ij})$, $b_{2ij}^l(D_{2ij}) = \max(0, \lceil \hat{D}_{2ij} - M \rceil)$ and $b_{2ij}^u = \min(K_{ij}, \lfloor \hat{D}_{2ij} - M \rfloor)$ for $i = 1, \dots, I, j = 1, \dots, J$. If the exact p -value is smaller than the desired level, say, 0.05 or 0.1, then we may conclude that the proposed model does not fit the data well.

3.3 Bayesian Approach

Fan et al. [10] applied the Bayesian approach to analyze highly reliable one-shot devices. They suggested three priors for the Bayesian estimation: exponential, normal and beta. Their simulation results show that all the priors perform similarly when the data possess enough information. However, if the data possess zero-failure cases, the normal prior is recommended by these authors. Here, we extend their work by incorporating a competing risk model into a one-shot device testing analysis under an accelerated life test setting and develop a Bayesian estimation framework. We adopt the prior distributions of the parameters stated in [10], namely, the exponential distribution, normal distribution with non-informative prior for the variance, and Dirichlet distribution which is an extension of the beta distribution.

Let $\pi(\boldsymbol{\alpha})$ be the joint prior. Then, the joint posterior density of $\boldsymbol{\alpha}$, given $\boldsymbol{\delta}$, \mathbf{IT} and \mathbf{w} , is

$$\pi(\boldsymbol{\alpha}|\boldsymbol{\delta}, \mathbf{IT}, \mathbf{w}) = \frac{L(\boldsymbol{\alpha}|\boldsymbol{\delta}, \mathbf{IT}, \mathbf{w})\pi(\boldsymbol{\alpha})}{\int L(\boldsymbol{\alpha}|\boldsymbol{\delta}, \mathbf{IT}, \mathbf{w})\pi(\boldsymbol{\alpha})d\boldsymbol{\alpha}}. \tag{16}$$

The denominator in (16) is usually not in closed-form. Following the method of [10], the Bayesian point estimate of $\hat{\boldsymbol{\alpha}} = N^{-1} \sum_{n=1}^N \boldsymbol{\alpha}^{(n)}$, where $\boldsymbol{\alpha}^{(n)}$ is the n -th value out of N samples from the posterior distribution. We may be interested in estimates of some parameters of interest at the normal operating condition, such as the failure rate λ_r , the survival probability p_{0i} at IT_i , and the expected life time $E(T)$. They can be estimated as follows:

Quantity	Bayesian Point Estimate
λ_r	$N^{-1} \sum_{n=1}^N \alpha_{r0}^{(n)} \exp(\alpha_{r1}^{(n)} w_j) = N^{-1} \sum_{n=1}^N \lambda_r^{(n)}$
p_{0i}	$N^{-1} \sum_{n=1}^N \exp(-(\lambda_1^{(n)} + \lambda_2^{(n)})IT_i)$
$P(T_1 < T_2 \min(T_1, T_2) < IT)$	$N^{-1} \sum_{n=1}^N \lambda_1^{(n)} / (\lambda_1^{(n)} + \lambda_2^{(n)})$
$E(T_r)$	$N^{-1} \sum_{n=1}^N 1/\lambda_r^{(n)}$
$E(T)$	$N^{-1} \sum_{n=1}^N (\lambda_1^{(n)} + \lambda_2^{(n)})^{-1}$

In the above, $E(T_r)$ and $E(T)$ are the mean lifetimes for cause r and that of the device, respectively. We also denote the probability $P(T_1 < T_2 | \min(T_1, T_2) < IT)$ by P.d1, for simplicity. The mean lifetime is a useful quantity for finding the 100 q_0 % quantile t_0 of the devices with exponential lifetime, and the relationship is simply

$$P(T > t_0) = \exp(-\lambda t_0) \Rightarrow t_0 = -\ln(q_0)/\lambda.$$

Consequently, the Bayesian estimate of the quantile will be $-\ln(q_0)n^{-1} \sum_{n=1}^N 1/\lambda^{(n)}$ which is a constant multiple of the Bayesian estimate of the mean lifetime. So, the MSE of the estimate of the quantile will simply be the multiplication of the MSE of the mean lifetime by $(\ln(q_0))^2$.

Several prior distributions are considered in this study, as described below.

3.3.1 Exponential Prior

Since the parameters $(\alpha_{10}, \alpha_{11}, \alpha_{20}, \alpha_{21})$ are all positive, a simple prior distribution for them is an exponential one of the form

$$\pi_1(\boldsymbol{\alpha}) = \prod_{r=1}^R \theta_{r0}^{-1} \exp(-\alpha_{r0}/\theta_{r0}) \theta_{r1}^{-1} \exp(-\alpha_{r1}/\theta_{r1}),$$

where $\alpha_{rm}, \theta_{rm} > 0$ for $r = 1, 2, m = 0, 1$.

The θ_{rm} are the unknown hyperparameters and $E(\alpha_{rm}) = \theta_{rm}$. Fan et al. [10] assumed that the reliability of the items under w_j and IT_i are around $\hat{p}_{0ij}, \hat{p}_{1ij}, \hat{p}_{2ij}$, for $i = 1, \dots, I$ and $j = 1, \dots, J$, and so $\hat{p}_{0ij}, \hat{p}_{1ij}, \hat{p}_{2ij}$ can be empirically estimated as $S_{ij}/K_{ij}, D_{1ij}/K_{ij}, D_{2ij}/K_{ij}$. If one of these estimates is zero, then it will be hard to determine the initial value. Zero-frequency problem has often been discussed in the literature and Lee and Cohen [18] suggested using

$$(\tilde{p}_{0ij}, \tilde{p}_{1ij}, \tilde{p}_{2ij}) = \left(\frac{S_{ij} + 1}{K_{ij} + 3}, \frac{D_{1ij} + 1}{K_{ij} + 3}, \frac{D_{2ij} + 1}{K_{ij} + 3} \right). \tag{17}$$

Recall that we can estimate $\lambda_{1ij} + \lambda_{2ij} = -\ln(p_{0ij})/IT_i$ by (12). Then, by (13) and (14), we can rewrite

$$\ln(\alpha_{r0}) + \alpha_{r1}w_j = \ln(p_{rij}) - \ln(1 - p_{0ij}) + \ln(-\ln(p_{0ij})) - \ln(IT_i)$$

for $l = 1, 2$. By replacing p_{rij} by the estimates in (17), we can obtain the least-square estimate of α by minimizing

$$S(\alpha) = \sum_{i=1}^I \sum_{j=1}^J \sum_{r=1}^2 (\hat{y}_{rij} - \ln(\alpha_{r0}) - \alpha_{r1}w_j - \ln(IT_i))^2,$$

where

$$\hat{y}_{rij} = \ln(\tilde{p}_{rij}) - \ln(1 - \tilde{p}_{0ij}) + \ln(-\ln(\tilde{p}_{0ij})). \tag{18}$$

By performing the necessary algebra, we derive the least-square estimates as

$$\hat{\alpha}_{r1}^{\text{LSE}} = \left\{ \begin{array}{l} (\sum K_{ij}) (\sum K_{ij}w_j(\hat{y}_{rij} - \ln(IT_i))) \\ - (\sum w_j K_{ij}) (\sum K_{ij}(\hat{y}_{rij} - \ln(IT_i))) \end{array} \right\} \\ \times \left\{ (\sum w_j^2 K_{ij}) (\sum K_{ij}) - (\sum w_j K_{ij})^2 \right\}^{-1},$$

$$\hat{\alpha}_{r0}^{\text{LSE}} = \exp \left\{ \left[\begin{array}{l} (\sum w_j^2 K_{ij}) (\sum K_{ij}(\hat{y}_{rij} - \ln(IT_i))) \\ - (\sum w_j K_{ij}) (\sum K_{ij}w_j(\hat{y}_{rij} - \ln(IT_i))) \end{array} \right] \right. \\ \left. \times \left[(\sum w_j^2 K_{ij}) (\sum K_{ij}) - (\sum w_j K_{ij})^2 \right]^{-1} \right\},$$

where $\sum = \sum_{i=1}^I \sum_{j=1}^J$.

However, the least-square estimates are not guaranteed to be positive, which will violate the assumption that both α_{11} and α_{21} are positive. To have a least-square estimate with nonnegative constraints, Liew [19] suggested using Inequality Constraints Least-Square (ICLS) method: if $\hat{\alpha}_{r1}^{\text{LSE}}$ is negative, then

$$\hat{\alpha}_{r1}^{\text{ICLS}} = 0,$$

$$\ln(\hat{\alpha}_{r0}^{\text{ICLS}}) = \ln(\hat{\alpha}_{r0}^{\text{LSE}}) + \hat{\alpha}_{r1}^{\text{LSE}} \times \frac{\sum_{i=1}^I \sum_{j=1}^J K_{ij}^2 w_j}{\sum_{i=1}^I \sum_{j=1}^J K_{ij}^2};$$

otherwise,

$$\hat{\alpha}_{r1}^{\text{ICLS}} = \hat{\alpha}_{r1}^{\text{LSE}},$$

$$\hat{\alpha}_{r0}^{\text{ICLS}} = \hat{\alpha}_{r0}^{\text{LSE}}.$$

If $\hat{\alpha}_{r1}^{\text{ICLS}} = 0$, we will add a very small number to it and make it positive: $\hat{\alpha}_{r1}^{\text{ICLS}} = 10^{-14}$, say. By presuming that the ICLS estimates are close to the prior means, we can use them as hyperparameter values for α_{r0} and α_{r1} , i.e., $\theta_{rm} = \hat{\alpha}_{rm}^{\text{ICLS}}$ for $r = 1, 2$ and $m = 0, 1$. Upon combining with (16), the posterior density then becomes

$$\pi_1(\boldsymbol{\alpha} | \boldsymbol{\delta}, \mathbf{IT}, \mathbf{w}) \propto L(\boldsymbol{\alpha} | \boldsymbol{\delta}, \mathbf{IT}, \mathbf{w}) \pi_1(\boldsymbol{\alpha} | \boldsymbol{\theta} = \boldsymbol{\alpha}^{\text{ICLS}})$$

$$\propto \prod_{i=1}^I \prod_{j=1}^J p_{0ij}^{S_{ij}} p_{1ij}^{D_{1ij}} p_{2ij}^{D_{2ij}} \prod_{r=1}^R \exp\left(-\left(\frac{\alpha_{r0}}{\hat{\alpha}_{r0}^{\text{ICLS}}} + \frac{\alpha_{r1}}{\hat{\alpha}_{r1}^{\text{ICLS}}}\right)\right).$$

3.3.2 Normal Prior

Let ε_{rij} be the error such that

$$\tilde{p}_{rij} = p_{rij} + \varepsilon_{rij},$$

and let us now assume that the error ε_{rij} are i.i.d. $N(0, \sigma^2)$ variables. Then, the conditional likelihood function of $\boldsymbol{\alpha}$, given σ^2 , is

$$L(\boldsymbol{\alpha} | IT_i, w_j, \tilde{p}_{rij}, \sigma^2) \propto \prod_{r=1}^2 \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2} (p_{rij} - \tilde{p}_{rij})^2\right\},$$

where p_{rij} and \tilde{p}_{rij} are as specified in (13)–(17), respectively. We will now adopt the likelihood function as the prior distribution of $\boldsymbol{\alpha}$:

$$\pi_2(\boldsymbol{\alpha} | \mathbf{IT}, \mathbf{w}, \sigma^2) \propto \prod_{i=1}^I \prod_{j=1}^J \prod_{r=1}^2 \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2} (p_{rij} - \tilde{p}_{rij})^2\right\}.$$

Since σ^2 is unknown, we adopt the noninformative prior

$$\pi(\sigma^2) \propto \frac{1}{\sigma^2}, \quad \sigma^2 > 0,$$

which yields the joint prior density of $\boldsymbol{\alpha}$

$$\begin{aligned} \pi_2(\boldsymbol{\alpha}|\mathbf{IT}, \mathbf{w}) &\propto \int_0^\infty \pi_2(\boldsymbol{\alpha}|\mathbf{IT}, \mathbf{w}, \sigma^2)\pi(\sigma^2)d\sigma^2 \\ &\propto \int_0^\infty (\sigma^2)^{-\frac{2IJ+2}{2}} \exp\left\{-\frac{1}{2\sigma^2} \sum_{i=1}^I \sum_{j=1}^J \sum_{r=1}^2 (p_{rij} - \tilde{p}_{rij})^2\right\} d\sigma^2 \\ &\propto \left\{ \sum_{i=1}^I \sum_{j=1}^J \sum_{r=1}^2 (p_{rij} - \tilde{p}_{rij})^2 \right\}^{-IJ}. \end{aligned}$$

Then, by (16), the joint posterior density of $\boldsymbol{\alpha}$ becomes

$$\pi_2(\boldsymbol{\alpha}|\boldsymbol{\delta}, \mathbf{IT}, \mathbf{w}) \propto \prod_{i=1}^I \prod_{j=1}^J p_{0ij}^{S_{ij}} p_{1ij}^{D_{1ij}} p_{2ij}^{D_{2ij}} \left\{ \sum_{i=1}^I \sum_{j=1}^J \sum_{r=1}^2 (p_{rij} - \tilde{p}_{rij})^2 \right\}^{-IJ}. \quad (19)$$

3.3.3 Dirichlet Prior

The natural extension of the beta prior discussed in [10] to the competing risks scenario is the Dirichlet prior. The prior density corresponding to p_{rij} is

$$f_{ij}(p_{0ij}, p_{1ij}, p_{2ij}) = \frac{p_{0ij}^{\beta_{0ij}-1} p_{1ij}^{\beta_{1ij}-1} p_{2ij}^{\beta_{2ij}-1}}{B(\boldsymbol{\beta}_{ij})},$$

where $p_{0ij} + p_{1ij} + p_{2ij} = 1$, $p_{0ij}, p_{1ij}, p_{2ij} > 0$, and

$$B(\boldsymbol{\beta}_{ij}) = \frac{\Gamma(\beta_{0ij})\Gamma(\beta_{1ij})\Gamma(\beta_{2ij})}{\Gamma(\beta_{0ij} + \beta_{1ij} + \beta_{2ij})}.$$

The hyperparameters $\boldsymbol{\beta}_{ij}$ are then chosen to match

$$E(p_{rij}) = \frac{\beta_{rij}}{\beta_{0ij} + \beta_{1ij} + \beta_{2ij}} = \tilde{p}_{rij}, \quad r = 1, 2. \quad (20)$$

Clearly, one more equation is needed for the determination of the hyperparameter $\boldsymbol{\beta}_{ij}$. For this, we may focus on the variance of p_{0ij} , which corresponds to the accuracy of the survival probability of one-shot devices. With the prior belief that $\text{Var}(p_{0ij}) = c^2$, the last equation to be used to determine the hyperparameter $\boldsymbol{\beta}_{ij}$ is given by

$$\text{Var}(p_{0ij}) = \frac{\beta_{0ij}(\beta_{1ij} + \beta_{2ij})}{(\sum_{r=0}^2 \beta_{rij})^2 (\sum_{r=0}^2 \beta_{rij} + 1)} = c^2. \quad (21)$$

Using (20) and (21), we then obtain the hyperparameters as

$$\beta_{1ij} = \tilde{p}_{1ij} \sum_{r=0}^2 \beta_{rij}, \beta_{2ij} = \tilde{p}_{2ij} \sum_{r=0}^2 \beta_{rij}, \beta_{0ij} = \sum_{r=0}^2 \beta_{rij} - \beta_{1ij} - \beta_{2ij}, \quad (22)$$

where

$$\sum_{r=0}^2 \beta_{rij} = \left(\frac{\tilde{p}_{0ij}(1 - \tilde{p}_{0ij})}{c^2} - 1 \right),$$

which yields the posterior distribution

$$\pi_3(\boldsymbol{\alpha}|\boldsymbol{\delta}, \mathbf{IT}, \mathbf{w}) \propto \prod_{i=1}^I \prod_{j=1}^J p_{0ij}^{S_{ij}+\beta_{0ij}-1} p_{1ij}^{D_{1ij}+\beta_{1ij}-1} p_{2ij}^{D_{2ij}+\beta_{2ij}-1};$$

here, p_{0ij} , p_{1ij} and p_{2ij} are as specified in (12)–(14), respectively, and \tilde{p}_{0ij} , \tilde{p}_{1ij} and \tilde{p}_{2ij} are as specified in (17).

3.3.4 Prior Belief on p_{rij}

In [10], the authors assumed that the prior belief of p_{ij} , denoted by \hat{p}_{ij} , is very reliable with regard to the true unknown parameter p_{ij} . Thus, they generated p_{0ij} from a beta distribution with specific choice of parameters. Now, by incorporating competing risks into the one-shot device testing, we assume that the prior belief of p_{rij} , denoted by \hat{p}_{rij} , is also very reliable in the sense that the variance of prior belief on the survival probability, $\text{Var}(\hat{p}_{0ij}) = c^2$, is small, with c^2 being a small constant. We also assume that $E(\hat{p}_{rij}) = p_{rij}$. Then, with the choice of parameters similar to the one in (22), we have

$$f(\hat{p}_{0ij}, \hat{p}_{1ij}, \hat{p}_{2ij}) \propto \hat{p}_{0ij}^{\beta_{0ij}^*-1} \hat{p}_{1ij}^{\beta_{1ij}^*-1} \hat{p}_{2ij}^{\beta_{2ij}^*-1},$$

where $\hat{p}_{0ij} + \hat{p}_{1ij} + \hat{p}_{2ij} = 1$, $\hat{p}_{0ij}, \hat{p}_{1ij}, \hat{p}_{2ij} > 0$. The parameters β_{rij}^* are chosen to be

$$\beta_{1ij}^* = \hat{p}_{1ij} \sum_{r=0}^2 \beta_{rij}^*, \beta_{2ij}^* = \hat{p}_{2ij} \sum_{r=0}^2 \beta_{rij}^*, \beta_{0ij}^* = \sum_{r=0}^2 \beta_{rij}^* - \beta_{1ij}^* - \beta_{2ij}^*,$$

where

$$\sum_{r=0}^2 \beta_{rij}^* = \left(\frac{\hat{p}_{0ij}(1 - \hat{p}_{0ij})}{c^2} - 1 \right). \quad (23)$$

The prior belief on the parameter can be used to replace \tilde{p}_{rij} in (18), (19) and (22) and the corresponding posterior distribution will then result. Note that $\sum_{r=0}^2 \beta_{rij}^*$ must be larger than zero. From (23), this means that $c^2 < p_{0ij}(1 - p_{0ij})$.

It is observed that Dirichlet prior is generally good for estimation when the devices with high reliability and normal prior provides better estimates for devices with moderate and low reliability.

4 Semi-parametric Models

Under the parametric setup, product lifetimes are assumed to be fully described by a probability distribution consisting of a number of model parameters. In this case, inference developed may be quite sensitive to departures from the assumed parametric distribution. For this reason, semi-parametric models that consist of parametric and nonparametric parts become good alternatives for model fitting propose when no suitable parametric models are known. Moreover, semi-parametric models can help in the choice of an appropriate model. Finkelstein and Wolfe [11] developed a semi-parametric regression model to examine the effect of dose on the hazard for tumor onset in an animal tumorigenicity study. Cheng and Wei [8] subsequently considered another semi-parametric model for analyzing panel data from an AIDS clinical trial. It is therefore of great interest to develop a proportional hazards model for one-shot device testing data analysis.

4.1 Proportional Hazards Models

Consider I constant-stress accelerated life-tests and K inspection times. For the i -th life-test, N_i devices are placed under stress level combinations with J stress factors, $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iJ})$, of which N_{ik} are tested at the k -th inspection time IT_k , where $N_i = \sum_{k=1}^K N_{ik}$ and $0 < IT_1 < \dots < IT_K$. Then, the numbers of devices that have failed by the time, n_{ik} , are recorded. One-shot device testing data obtained from such a life-test can then be represented as $(n_{ik}, N_{ik}, \mathbf{x}_i, IT_k)$, for $i = 1, 2, \dots, I, k = 1, 2, \dots, K$. Under the proportional hazards assumption [9], the cumulative hazard function is given by

$$H(t; \boldsymbol{\eta}, \mathbf{a}, \mathbf{x}) = H_0(t; \boldsymbol{\eta})\lambda(\mathbf{x}; \mathbf{a}), \quad (24)$$

where $H_0(t; \boldsymbol{\eta})$ is the baseline cumulative hazard function, $\boldsymbol{\eta} = (\eta_1, \eta_2, \dots, \eta_K)$, and $\mathbf{a} = (a_1, a_2, \dots, a_J)$ is a vector of coefficients for stress factors. The model in (24) is composed of two independent components. One component measures the influence of the stress factors and the other measures the changes in the baseline. We now assume a log-linear link function to relate the stress levels to the failure times of the

units to obtain the cumulative hazard function from (24) to be

$$H(t; \boldsymbol{\eta}, \mathbf{a}, \mathbf{x}) = H_0(t; \boldsymbol{\eta}) \exp \left(\sum_{j=1}^J a_j x_j \right);$$

the corresponding reliability function is then

$$R(t; \boldsymbol{\eta}, \mathbf{a}, \mathbf{x}) = \exp(-H(t; \boldsymbol{\eta}, \mathbf{a}, \mathbf{x})) = R_0(t; \boldsymbol{\eta})^{\exp(\sum_{j=1}^J a_j x_j)},$$

where $R_0(t; \boldsymbol{\eta}) = \exp(-H_0(t; \boldsymbol{\eta}))$ is the baseline reliability function. It is of interest to note that the baseline reliability function is a decreasing function, that is, $0 < R_0(IT_K; \boldsymbol{\eta}) < R_0(IT_{K-1}; \boldsymbol{\eta}) < \dots < R_0(IT_2; \boldsymbol{\eta}) < R_0(IT_1; \boldsymbol{\eta}) < 1$, and therefore, we let

$$\gamma(\eta_k) = 1 - R_0(IT_k; \boldsymbol{\eta}) = 1 - \exp(-\exp(\eta_k)),$$

and, for $k = 1, 2, \dots, K - 1$,

$$\gamma(\eta_k) = \frac{1 - R_0(IT_k; \boldsymbol{\eta})}{1 - R_0(IT_{k+1}; \boldsymbol{\eta})} = 1 - \exp(-\exp(\eta_k)).$$

Let us now define $G_k = \prod_{m=k}^K \gamma(\eta_m)$. We then have

$$R_0(IT_k; \boldsymbol{\eta}) = 1 - \prod_{m=k}^K \{1 - \exp(-\exp(\eta_m))\} = 1 - G_k.$$

Also, it can be seen that $R_0(IT_k; \boldsymbol{\eta}) \approx R_0(IT_{k+1}; \boldsymbol{\eta})$ when $\eta_k \rightarrow +\infty$, and $R_0(IT_k; \boldsymbol{\eta}) \gg R_0(IT_{k+1}; \boldsymbol{\eta})$ when $\eta_k \rightarrow -\infty$. The value of η_k gives us an idea about the probability of failure between successive times IT_k and IT_{k+1} . It is observed that few devices will fail between IT_k and IT_{k+1} when η_k tends to $+\infty$, and many devices will fail between IT_k and IT_{k+1} when η_k tends to $-\infty$.

The log-likelihood function based on the data is then given by

$$\begin{aligned} \ell(\mathbf{a}, \boldsymbol{\eta}) &= \sum_{i=1}^I \sum_{k=1}^K n_{ik} \ln \left(1 - (1 - G_k)^{\exp(\sum_{j=1}^J a_j x_{ij})} \right) \\ &+ (N_{ik} - n_{ik}) \ln (1 - G_k) \exp \left(\sum_{j=1}^J a_j x_{ij} \right) + \ln(C). \end{aligned}$$

We now present a connection between the proportional hazards model and a parametric model with proportional hazard rates. The two-parameter Weibull distribution is commonly used as a lifetime distribution having proportional hazard rates. In constant-stress accelerated-life tests, if the lifetimes of units subjected to ele-

vated stress levels \mathbf{x}_i follow the Weibull distribution with the same shape parameter $\beta = \exp(b)$ and the scale parameter $\alpha_i = \exp\left(\sum_{j=0}^J c_j x_{ij}\right)$, the cdf of the Weibull distribution is

$$F_T(t, \alpha_i, \beta) = 1 - \exp\left(-\left(\frac{t}{\alpha_i}\right)^\beta\right), \quad t > 0.$$

If the proportional hazards assumption holds, then the baseline reliability and the coefficients of stress factors are given by

$$R_0(IT_k; \boldsymbol{\eta}) = \exp\left(-IT_k^\beta \exp(-\beta c_0)\right)$$

and

$$a_j = -\beta c_j,$$

for $j = 1, 2, \dots, J$.

Furthermore, $\eta_k = \ln\left(-\ln\left(1 - \frac{1-R_0(IT_k)}{1-R_0(IT_{k+1})}\right)\right)$ and $\eta_K = \beta(\ln(IT_K) - c_0)$.

In the present case, the maximum likelihood estimators of the model parameters $\hat{\boldsymbol{\theta}} = (\hat{\mathbf{a}}, \hat{\mathbf{b}})$ have no explicit form but can be determined numerically, for example, by the Newton-Raphson method, which requires the first- and second-order derivatives of the log-likelihood function.

4.2 Test of Proportional Hazard Rates

The proposed model imposes only the proportional hazards assumption and allows hazard rate to change in a nonparametric way. Here, we present a test for the proportional hazards assumption based on one-shot device testing data. Let us consider the distance-based test statistic M of the form in (9), introduced in Sect. 2.5. The test statistic simply assesses the fit of the assumed model to the observed data, and so we would observe a large value of the test statistic M when the assumed model is not a good fit to the data. From (10), we can readily validate the proportional hazards assumption when the p -value is sufficiently large, i.e., p -value > 0.05 or 0.1 .

5 Applications to Tumor Toxicological Data

In this section, two real data from a toxicological study are used to illustrate the models and the estimation methods described in the preceding sections.

5.1 *Benzidine Dihydrochloride Data*

Survival and sacrifice data which involved 1816 mice, of which 553 developed tumors, taken from the National Center for Toxicological Research, is used here to illustrate the models and the inferential results described in the preceding sections. These data have been considered earlier by Kodell and Nelson [16], and Lindsey and Ryan [20]. The original data were classified into five groups, and were reported by Kodell and Nelson [16]. Note that not all mice were sacrificed at prespecified times because some died of tumors naturally before the sacrifice time. Thus, their times of natural death would also be treated as times of sacrifice. We considered the mice sacrificed with tumors, died of tumors, and died of competing risks with liver tumors as those having tumors, while the mice sacrificed without tumors and died of competing risk without liver tumors as those not having tumors.

Let $a_1(d_1)$, $a_2(d_2)$, and $a_3(d_3)$ denote the parameters corresponding to the covariates of strain of offspring, gender, and square root of concentration of the chemical of benzidine dihydrochloride in the scale parameter of the Weibull (gamma) distribution; and $b_1(c_1)$, $b_2(c_2)$, and $b_3(c_3)$ similarly for the shape parameters, respectively. The MLEs of the model parameters were computed by means of the EM algorithm, along with the corresponding standard errors and the 95 % asymptotic confidence intervals for all the model parameters. These results are all presented in Table 3. It is evident that both the shape and scale parameters vary with all the covariates. The goodness of fit test with p -values reveals that the Weibull and gamma distributions incorporated with the log-linear links are suitable for these data. Also, we have grouped the data into three inspection times $IT = (10, 16, 22)$ and then tested the proportional hazard rates using the proportional hazards model. The test provides strong evidence against the proportional hazards model for these data since the calculated p -value of the test turns out to be as small as 0.0019. This finding is consistent with the results in Table 3, wherein it can be readily seen that the parametric distributions without proportional hazard rates provide a good fit to these data.

5.2 *Carcinogen 2-AAF Data*

Here we apply the competing risk model to a dataset concerning rodent tumorigenicity described in [20]. The experimental results were observed by National Center for Toxicological Research in 1974. 3355 out of 24,000 female mice were randomized to a control group ($w = 0$) or groups that were injected with a high dose (150 parts per million) of a known carcinogen, called 2-AAF ($w = 1$), to different parts of the body. The inspection times used on the mice were 12, 18 and 33 months and the outcome of mice were death without tumour (DNT), with tumour (DWT), and sacrificed without tumour (SNT) and with tumour (SWT). In this analysis, we ignore the information about parts of mouse bodies where the drugs were injected. We combine SNT and SWT as the sacrificed group ($r = 0$), and denote the cause of DNT as natural death

Table 3 MLEs of model parameters, along with their standard errors (within brackets) and the corresponding 95 % asymptotic confidence intervals

Weibull ($M = 6.4416, p\text{-value} = 0.1565$)				
	Scale parameter α			
	a_0	a_1	a_2	a_3
Estimates (s.e.)	3.2996 (0.0365)	0.0485 (0.0219)	0.5178 (0.0498)	-0.0508 (0.0028)
Asymptotic CI	(3.2281, 3.3711)	(0.0055, 0.0915)	(0.4202, 0.6154)	(-0.0563, -0.0452)
	Shape parameter η			
	b_0	b_1	b_2	b_3
Estimates (s.e.)	2.1933 (0.1638)	-0.2389 (0.0895)	-0.4741 (0.1140)	-0.0327 (0.0114)
Asymptotic CI	(1.8722, 2.5143)	(-0.4143, -0.0635)	(-0.6875, -0.2507)	(-0.0551, -0.0104)
Gamma ($M = 6.3067, p\text{-value} = 0.2272$)				
	Shape parameter α			
	c_0	c_1	c_2	c_3
Estimates (s.e.)	3.0597 (0.0287)	-0.0576 (0.0230)	-0.9780 (0.0477)	-0.0330 (0.0025)
Asymptotic CI	(3.0034, 3.1160)	(-0.1026, -0.0125)	(-1.0714, -0.8846)	(-0.0379, -0.0281)
	Scale parameter β			
	d_0	d_1	d_2	d_3
Estimates (s.e.)	0.2616 (0.0421)	0.0781 (0.0319)	1.5627 (0.0729)	-0.0237 (0.0037)
Asymptotic CI	(0.1790, 0.3442)	(0.0156, 0.1406)	(1.4198, 1.7057)	(-0.0310, -0.0165)

Table 4 Numbers of mice sacrificed ($r = 0$) and died (without tumour $r = 1$, with tumour $r = 2$) from the ED01 experiment data

		$\delta_{ijk} = 0$	$\delta_{ijk} = 1$	$\delta_{ijk} = 2$
$IT_1 = 12$	$w_1 = 0$	115	22	8
	$w_2 = 1$	110	49	16
$IT_2 = 18$	$w_1 = 0$	780	42	8
	$w_2 = 1$	540	54	26
$IT_3 = 33$	$w_1 = 0$	675	200	85
	$w_2 = 1$	510	64	51

($r = 1$) and the cause of DWT as death due to cancer ($r = 2$). These modified data are presented in Table 4 and the corresponding estimates of model parameters obtained by the proposed estimation method are presented in Table 5.

From Table 5, we note that the estimate of α_{11} is negative which means that the drug will decrease the hazard rate of natural death. The reason for this is that the carcinogenic drug will increase the chances of developing a tumour which will of course

Table 5 Estimates of various parameters of interest determined by the EM algorithm

Parameter	α_{10}	α_{11}	α_{20}	α_{21}	
Estimate	6.169e-03	-1.279e-01	2.347e-03	2.532e-01	
Parameter	$p_{01 w=0}$	$p_{02 w=0}$	$p_{03 w=0}$	$E(T w = 0)$	$P.d1_{w=0}$
Estimate	9.029e-01	8.579e-01	7.55e-01	1.174e+02	7.244e-01
Parameter	$p_{01 w=1}$	$p_{02 w=1}$	$p_{03 w=1}$	$E(T w = 1)$	$P.d1_{w=1}$
Estimate	9.036e-01	8.589e-01	7.566e-01	1.183e+02	6.423e-01

decrease the chances of death without tumour, meaning that $P.d1_{w=0} > P.d1_{w=1}$. The estimate of α_{21} is positive suggesting that the drug is indeed carcinogenic. Apparently, there are no significant differences between the life expectancies of mice receiving the drug and not receiving the drug.

Acknowledgments Our sincere thanks go to the editors of this volume for extending an invitation to us which provided an impetus for preparing this overview article.

References

- Balakrishnan N, Ling MH (2012) EM algorithm for one-shot device testing under the exponential distribution. *Comput Stat Data Anal* 56:502–509
- Balakrishnan N, Ling MH (2012) Multiple-stress model for one-shot device testing data under exponential distribution. *IEEE Trans Reliab* 61:809–821
- Balakrishnan N, Ling MH (2013) Expectation maximization algorithm for one shot device accelerated life testing with Weibull lifetimes, and variable parameters over stress. *IEEE Trans Reliab* 62:537–551
- Balakrishnan N, Ling MH (2014) Gamma lifetimes and one-shot device testing analysis. *Reliab Eng Syst Saf* 126:54–64
- Balakrishnan N, So HY, Ling MH (2015) EM algorithm for one-shot device testing with competing risks under exponential distribution. *Reliab Eng Syst Saf* 137:129–140
- Balakrishnan N, So HY, Ling MH (2015) A Bayesian approach for one-shot device testing with exponential lifetimes under competing risks. *IEEE Trans Reliab*
- Bishop YMM, Feinberg SE, Holland PW (1975) *Discrete multivariate analysis: theory and practice*. MIT Press, Cambridge
- Cheng SC, Wei LJ (2000) Inferences for a semiparametric model with panel data. *Biometrika* 87:89–97
- Cox DR, Oakes D (1984) *Analysis of survival data*. Chapman & Hall, London
- Fan TH, Balakrishnan N, Chang CC (2009) The Bayesian approach for highly reliable electro-explosive devices using one-shot device testing. *J Stat Comput Simul* 79:1143–1154
- Finkelstein DM, Wolfe RA (1985) A semiparametric model for regression analysis of interval-censored failure time data. *Biometrics* 41:933–945
- Hoyle MH (1973) Transformations: an introduction and a bibliography. *Int Stat Rev* 41:203–223
- Husak GJ, Michaelsen J, Funk C (2007) Use of the gamma distribution to represent monthly rainfall in Africa for drought monitoring applications. *Int J Climatol* 27:935–944
- Johnson NL, Kotz S, Balakrishnan N (1994) *Continuous univariate distributions*, vol 1, 2nd edn. Wiley, New York

15. Johnson NL, Kotz S, Balakrishnan N (1995) Continuous univariate distributions, vol 2, 2nd edn. Wiley, New York
16. Kodell RL, Nelson CJ (1980) An illness-death model for the study of the carcinogenic process using survival/sacrifice data. *Biometrics* 36:267–277
17. Kwon K, Frangopol DM (2010) Bridge fatigue reliability assessment using probability density functions of equivalent stress range based on field monitoring data. *Int J Fatigue* 32:1221–1232
18. Lee HL, Cohen MA (1985) A multinomial logit model for the spatial distribution of hospital utilization. *J Bus Econ Stat* 3:159–168
19. Liew CK (1976) Inequality constrained least-squares estimation. *J Amer Stat Assoc* 71:746–751
20. Lindsey JC, Ryan LM (1993) A three-state multiplicative model for rodent tumorigenicity experiments. *J R Stat Soc Ser C* 42:283–300
21. Ling MH, So HY, Balakrishnan N (2015) Likelihood inference under proportional hazards model for one-shot device testing. *IEEE Trans Reliab*
22. Louis TA (1982) Finding the observed information matrix when using the EM algorithm. *J R Stat Soc Ser B* 44:226–233
23. McLachlan GJ, Krishnan T (2008) The EM Algorithm and extensions, 2nd edn. Wiley, Hoboken
24. Meeter CA, Meeker WQ (1994) Optimum accelerated life tests with a nonconstant scale parameter. *Technometrics* 36:71–83
25. Morris MD (1987) A sequential experimental design for estimating a scale parameter from quantal life testing data. *Technometrics* 29:173–181
26. Newby M (2008) Monitoring and maintenance of spares and one shot devices. *Reliab Eng Syst Saf* 93:588–594
27. Ng HKT, Chan PS, Balakrishnan N (2002) Estimation of parameters from progressively censored data using EM algorithm. *Comput Stat Data Anal* 39:371–386
28. Noguiera E, Vázquez M, Núñez N (2009) Evaluation of AlGaInP LEDs reliability based on accelerated tests. *Mircoelectron Reliab* 49:1240–1243
29. Shaked M, Singpurwalla ND (1990) A Bayesian approach for quantile and response probability estimation with applications to reliability. *Ann Inst Stat Math* 42:1–19
30. Vázquez M, Núñez N, Noguiera E, Borreguero A (2010) Degradation of AlInGaP red LEDs under drive current and temperature accelerated life tests. *Mircoelectron Reliab* 50:1559–1562
31. Viveros R, Balakrishnan N (1993) Statistical inference from start-up demonstration test data. *J Qual Tech* 25:119–130

Probabilistic Graphical Models for Fault Diagnosis in Complex Systems

Ali Abdollahi, Krishna R. Pattipati, Anuradha Kodali, Satnam Singh, Shigang Zhang and Peter B. Luh

Abstract In this chapter, we discuss the problem of fault diagnosis for complex systems in two different contexts: static and dynamic probabilistic graphical models of systems. The fault diagnosis problem is represented using a tripartite probabilistic graphical model. The first layer of this tripartite graph is composed of components of the system, which are the potential sources of failures. The condition of each component is represented by a binary state variable which is zero if the component is healthy and one otherwise. The second layer is composed of tests with binary outcomes (pass or fail) and the third layer is the noisy observations associated with the test outcomes. The cause–effect relations between the states of components and the observed test outcomes can be compactly modeled in terms of detection and false alarm probabilities. For a failure source and an observed test outcome, the probability of fault detection is defined as the probability that the observed test outcome is a fail given that the component is faulty, and the probability of false alarm is defined as the probability that the observed test outcome is a fail given that the component is healthy.

A. Abdollahi (✉) · K.R. Pattipati · S. Singh · S. Zhang · P.B. Luh
Department of Electrical and Computer Engineering, University of Connecticut,
371 Fairfield Way, U-4157, Storrs, CT 06269, USA
e-mail: ali.abdollahi@uconn.edu

K.R. Pattipati
e-mail: krishna@enr.uconn.edu

S. Singh
e-mail: satnam74@yahoo.com

S. Zhang
e-mail: shigang391@foxmail.com

P.B. Luh
e-mail: peter.luh@uconn.edu

A. Kodali
University of California Santa Cruz, NASA Ames Research Center,
Mail Stop 269-1, Moffett Field, CA 94035, USA
e-mail: anuradha.kodali@nasa.gov

When the probability of fault detection is one and the probability of false alarm is zero, the test is termed perfect; otherwise, it is deemed imperfect. In static models, the diagnosis problem is formulated as one of maximizing the posterior probability of component states given the observed fail or pass outcomes of tests. Since the solution to this problem is known to be NP-hard, to find near-optimal diagnostic solutions, we use a Lagrangian (dual) relaxation technique, which has the desirable property of providing a measure of suboptimality in terms of the approximate duality gap. Indeed, the solution would be optimal if the approximate duality gap is zero. The static problem is discussed in detail and some interesting properties, such as the reduction of the problem to a set covering problem in the case of perfect tests, are discussed. We also visualize the dual function graphically and introduce some insights into the static fault diagnosis problem. In the context of dynamic probabilistic graphical models, it is assumed that the states of components evolve as independent Markov chains and that, at each time epoch, we have access to some of the observed test outcomes. Given the observed test outcomes at different time epochs, the goal is to determine the most likely evolution of the states of components over time. The application of dual relaxation techniques results in significant reduction in the computational burden as it transforms the original coupled problem into separable subproblems, one for each component, which are solved using a Viterbi decoding algorithm. The problems, as stated above, can be regarded as *passive monitoring*, which relies on synchronous or asynchronous availability of sensor results to infer the most likely state evolution of component states. When information is sequentially acquired to isolate the faults in minimum time, cost, or other economic factors, the problem of fault diagnosis can be viewed as active probing (also termed *sequential testing* or *troubleshooting*). We discuss the solution of active probing problems using the information heuristic and rollout strategies of dynamic programming. The practical applications of passive monitoring and active probing to fault diagnosis problems in automotive, aerospace, power, and medical systems are briefly mentioned.

1 Introduction

With increasing number of subsystems and components in complex engineered systems, the need for system monitoring, anomaly (fault) detection, and root cause analysis is paramount for improved system availability. However, the process of detecting and isolating faults in complex systems is challenging, because

- The numbers of faults and monitoring signals (“processed sensor measurements,” “tests,” “symptoms,” “visual observations”) in these systems are large (running into tens of thousands).
- Each test outcome may be caused by faults in multiple components of possibly multiple subsystems (“many-to-many” fault–test relationships).
- Faults propagate from one subsystem to another (“cross-subsystem fault propagation”) with delays.

- Test outcomes, which are uncertain, are observed with delays caused by fault propagation, computation, and communication.
- Simultaneous occurrence of multiple faults is frequent.

This makes traditional single-fault diagnosis approaches untenable. Uncertain test outcomes pose particularly difficult challenges to fault diagnosis: while, in a perfect binary test outcome situation, a passed test indicates the normal status of its associated components and a failed test implies the existence of at least one faulty component associated with the test, neither can be inferred when the tests are imperfect. In the binary test outcome case, an imperfect test outcome may be reported as *passed*, even when there are faulty component(s) associated with the test, a situation referred to as *imperfect (missed) detection*. On the other hand, an imperfect test outcome may be reported as *failed*, even though there is no faulty component associated with the test, a condition referred to as a *false alarm*.

In broad terms, fault diagnosis problems can be categorized into two groups: passive monitoring and active monitoring (“probing”). In passive monitoring, the fault diagnosis subsystem (“diagnoser”) relies on synchronous or asynchronous availability of test outcomes to detect abnormal conditions in the system and to isolate the faulty component or components in the system. This is also termed *abductive reasoning*. One application of passive monitoring is in disease diagnosis, such as Quick Medical Reference Decision-Theoretic (QMR-DT) problem, wherein bipartite belief networks are used to model the probabilistic cause–effect relations of a set of diseases and a set of findings [108, 133]. On the other hand, in active probing, the aim is to adaptively sequence the application of tests based on the outcomes of previously applied tests in order to minimize the expected troubleshooting time or the expected testing cost. Evidently, hybrid passive and active monitoring (e.g., passive monitoring followed by active probing to troubleshoot problems) is a common practice in complex system diagnosis.

Graphical models combine graph theory and probability theory into an elegant formalism for visualizing models, gaining insights into conditional independence properties, inference, and learning. Since fault diagnosis is inherently an inference problem, it is natural to employ graphical models for fault diagnosis. In this vein, a fault diagnosis system can be conceptualized as a tripartite directed graph (digraph) as shown in Fig. 1. The first (top) layer contains the components (failure modes or failure sources) and the second (middle) layer is comprised of tests. The cause–effect relations between the component health states (herein termed failure source states) and the test outcomes may be perfect or imperfect (probabilistic). The third (bottom) layer encompasses the observations of test outcomes, which may be perfect or imperfect, observed synchronously or asynchronously and, in the synchronous case, not all observations of test outcomes may be available at each epoch. Note that the observations may be different from the test outcomes, for example, due to uncertainty and communication errors. When the observations are perfect, we have $o_j(k) = t_j(k)$ and the tripartite digraph is reduced to a bipartite one. In the following, we assume that observations are perfect and represent the system as a

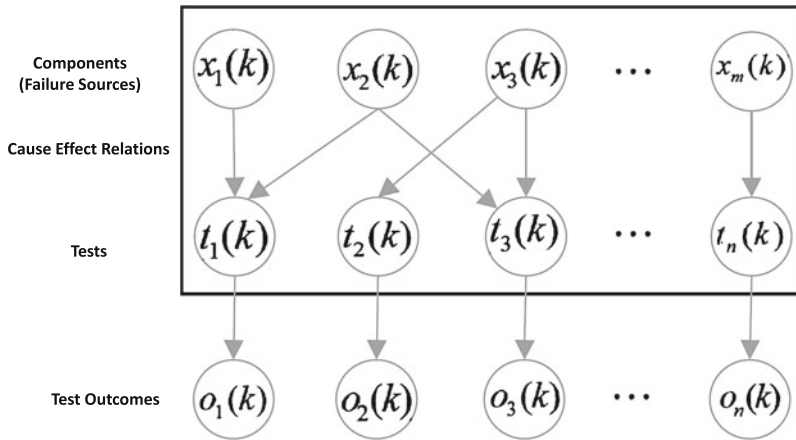


Fig. 1 Tripartite digraph of fault diagnosis system

bipartite digraph. Also in the case of static fault diagnosis, the dependence on time is omitted in the bipartite digraph.

The problem has three basic elements, namely failure sources (associated with components), tests, and dependency relations between failure sources and tests. Each of these elements can be abstracted in various ways to capture the nature of the fault diagnosis problem in a complex system. For example, failure sources associated with a component can be permanent (static) or intermittent (dynamic). They may have binary states (normal, abnormal) or multivalued states (nominal and various degraded modes of operation). The failure sources may be independent or coupled (see Fig. 2). In the same vein, a test can be categorized as having binary or multivalued outcomes,

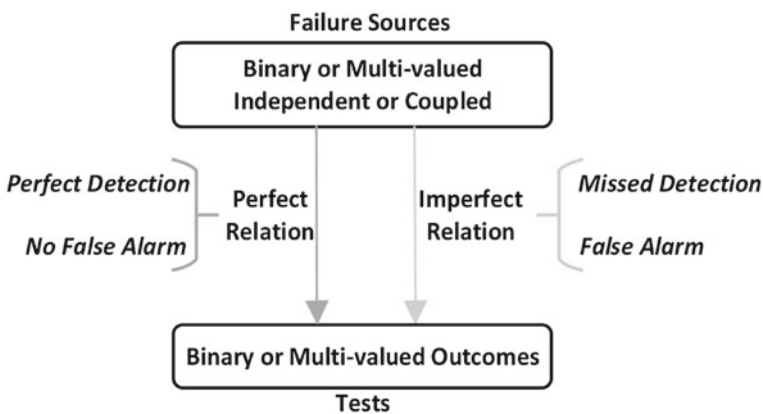


Fig. 2 Modeling abstractions in fault diagnosis subsystems

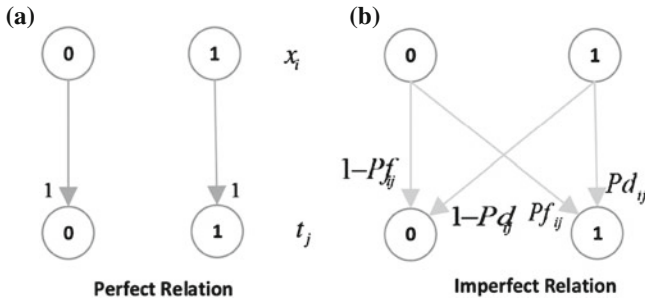


Fig. 3 Perfect and imperfect relations of a failure source and a test outcome

and the relationship between the failure sources and test outcomes can be perfect or imperfect, as alluded to earlier.

For ease of exposition and simplicity of notation, we consider failure sources and test outcomes with binary states. Figure 3a shows a component and a test, each with binary (0-1) states, having a perfect dependency relationship. Evidently, in this case, with probability one, $x_i = 1$ is mapped to $t_j = 1$, and $x_i = 0$ to $t_j = 0$.

Figure 3b shows the dependency relationship of a failure source and a test outcome in an imperfect test setting. An imperfect binary relation can be represented by probability of detection and probability of false alarm as follows:

Probability of Detection: If $x_i = 1$, then there is a probability Pd_{ij} that test t_j fails. Here, Pd_{ij} denotes the probability of detection. Formally, $Pd_{ij} = \Pr(t_j = 1 | x_i = 1)$.

Probability of False Alarm: If $x_i = 0$, then there is a probability Pf_{ij} that test t_j fails. Here, Pf_{ij} denotes the probability of false alarm. Formally, $Pf_{ij} = \Pr(t_j = 1 | x_i = 0)$.

In the sequel, we discuss the static and dynamic form of multiple fault diagnosis and also passive and active monitoring using this simplified model. Extensions of the method to coupled and delayed failure source state propagation, and delayed observations can be found in [65, 66, 110, 120, 135].

2 Static Multiple Fault Diagnosis

In this section, we consider the multiple fault diagnosis problem in a static context, and refer to it as static multiple fault diagnosis (SMFD). The SMFD problem is comprised of the following:

- The system consists of m components c_1, c_2, \dots, c_m . Without loss of generality, a single failure mode is associated with each component. The failure modes are assumed to be conditionally independent. Let $S = \{s_1, s_2, \dots, s_m\}$ be the set of

independent potential failure modes (failure sources), respectively, associated with the system components c_1, c_2, \dots, c_m .

- To each component (potential failure source), c_i (s_i) is assigned a binary state variable x_i , where $x_i = 1$ represents the fault state of the component and $x_i = 0$ represents the normal state of the component.
- Each potential failure source is assumed to have a prior probability p_{s_i} of being faulty; in other words, $\Pr(x_i = 1) = p_{s_i}$.
- The system is assumed to have n tests, $\mathbf{t} = \{t_1, t_2, \dots, t_n\}$.
- Tests are assumed to be independent.
- When test t_j has a pass outcome, it is represented by $t_j = 0$, otherwise by $t_j = 1$.
- The cause–effect relation between the potential failure sources and the tests are assumed to be probabilistic with detection probabilities and false alarm probabilities, as discussed in the previous section.
- In a real-world system, if component c_i is associated with test t_j , the detection probability Pd_{ij} is a number close to 1, for example, in the range $[0.75, 1]$ and the false alarm probability Pf_{ij} is a number close to 0, for example, in the range $[0, 0.20]$. The situation that the component c_i is not associated with test t_j is represented by $Pd_{ij} = Pf_{ij} = 0$.
- We represent the states of failures sources as \mathbf{x} . In other words $\mathbf{x} = \{x_1, x_2, \dots, x_m\}$.

Based on the above assumptions, the problem of SMFD is defined as follows:

Static Multiple Fault Diagnosis (SMFD) Problem: *Given T , a subset of all test outcomes \mathbf{t} , i.e., $T \subseteq \mathbf{t}$, what are the most likely states of failure sources, \mathbf{x} ?*

Formally, the problem is stated as follows:

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} \Pr(\mathbf{x}|T). \quad (1)$$

Using Bayes' rule, we can write $\Pr(\mathbf{x}|T)$ as

$$\Pr(\mathbf{x}|T) = \frac{\Pr(T|\mathbf{x}) \Pr(\mathbf{x})}{\Pr(T)}. \quad (2)$$

Since maximization of $\Pr(\mathbf{x}|T)$ is equivalent to maximization of $\Pr(T|\mathbf{x}) \Pr(\mathbf{x})$, we can simplify the problem further by maximizing the logarithm of $\Pr(T|\mathbf{x}) \Pr(\mathbf{x})$. Thus, the problem is

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} \ln(\Pr(T|\mathbf{x}) \Pr(\mathbf{x})). \quad (3)$$

In tackling the problem, we first classify the given tests T into two subsets of passed tests and failed tests, respectively, represented by T_p and T_f . Since the test outcomes are assumed to be conditionally independent, we have $\Pr(T|\mathbf{x}) = \Pr(T_p|\mathbf{x}) \Pr(T_f|\mathbf{x})$. Therefore, by converting the logarithm of the product to the sum of logarithms, we can write (3) as follows:

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} \{ \ln(\Pr(T_f|\mathbf{x})) + \ln(\Pr(T_p|\mathbf{x})) + \ln(\Pr(\mathbf{x})) \}. \quad (4)$$

Since the potential failure sources are assumed to be independent, we have

$$\Pr(\mathbf{x}) = \prod_{i=1}^m \Pr(x_i). \quad (5)$$

Based on problem assumptions, we have $\Pr(x_i = 1) = p_{s_i}$; therefore $\Pr(x_i) = (p_{s_i})^{x_i} (1 - p_{s_i})^{1-x_i}$, which by defining $p_i = \frac{p_{s_i}}{1-p_{s_i}}$ can be simplified as

$$\Pr(x_i) = (p_i)^{x_i} (1 - p_{s_i}). \quad (6)$$

Taking the logarithm of (5) and using (6), we have

$$\ln(\Pr(\mathbf{x})) = \sum_{i=1}^m \ln(p_i)x_i + \sum_{i=1}^m \ln(1 - p_{s_i}). \quad (7)$$

Note that $\sum_{i=1}^m \ln(1 - p_{s_i})$ is a known constant. The next term to calculate is $\ln(\Pr(T_p|\mathbf{x}))$. Since the test outcomes are assumed to be conditionally independent, we can write

$$\Pr(T_p|\mathbf{x}) = \prod_{t_j \in T_p} \Pr(t_j = \text{pass}|\mathbf{x}). \quad (8)$$

In order that test t_j has passed outcome conditioned on \mathbf{x} , it should pass conditioned on each component state. In other words,

$$\Pr(t_j = \text{pass}|\mathbf{x}) = \prod_{i=1}^m \Pr(t_j = \text{pass}|x_i). \quad (9)$$

From Fig. 3b, we have

$$\Pr(t_j = \text{pass}|x_i) = \begin{cases} 1 - Pf_{ij} & x_i = 0 \\ 1 - Pd_{ij} & x_i = 1 \end{cases}. \quad (10)$$

Equation (10) can be represented in the compact form of $\Pr(t_j = \text{pass}|x_i) = (1 - Pd_{ij})^{x_i} (1 - Pf_{ij})^{1-x_i}$, which by defining $\overline{Pd}_{ij} = 1 - Pd_{ij}$ and $\overline{Pf}_{ij} = 1 - Pf_{ij}$ can be further compactly written as

$$\Pr(t_j = \text{pass}|x_i) = \overline{Pf}_{ij} \left(\frac{\overline{Pd}_{ij}}{\overline{Pf}_{ij}} \right)^{x_i}. \quad (11)$$

Inserting (11) into (9) and taking the logarithm, we have

$$\ln(\Pr(t_j = \text{pass}|\mathbf{x})) = \ln(y_j) = h_j + \sum_{i=1}^m \beta_{ij} x_i, \quad (12)$$

where $\beta_{ij} = \ln\left(\frac{\overline{Pd}_{ij}}{\overline{Pf}_{ij}}\right)$, $h_j = \sum_{i=1}^m \ln(\overline{Pf}_{ij})$, and the continuous variable y_j is defined as

$$y_j = \Pr(t_j = \text{pass}|\mathbf{x}). \quad (13)$$

Note that β_{ij} and h_j are known constants. By taking the logarithm of both sides of (8) and then using relation (12), we can write

$$\ln(\Pr(T_p|\mathbf{x})) = \sum_{t_j \in T_p} \ln(y_j) = \sum_{t_j \in T_p} h_j + \sum_{t_j \in T_p} \sum_{i=1}^m \beta_{ij} x_i. \quad (14)$$

Note that the summation $\sum_{t_j \in T_p} h_j$ is simply a constant. The last term to be characterized from (4) is $\ln(\Pr(T_f|\mathbf{x}))$. Similarly to the passed tests, for failed tests we have $\Pr(T_f|\mathbf{x}) = \prod_{t_j \in T_f} \Pr(t_j = \text{fail}|\mathbf{x})$ and as $\Pr(t_j = \text{fail}|\mathbf{x}) = 1 - \Pr(t_j = \text{pass}|\mathbf{x}) = 1 - y_j$, we have the following relation:

$$\Pr(T_f|\mathbf{x}) = \prod_{t_j \in T_f} (1 - y_j), \quad (15)$$

where based on (12) and the definition of y_j , we have

$$\ln(y_j) = h_j + \sum_{i=1}^m \beta_{ij} x_i. \quad (16)$$

By taking the logarithm of (15), we have

$$\ln(\Pr(T_f|\mathbf{x})) = \sum_{t_j \in T_f} \ln(1 - y_j). \quad (17)$$

Now inserting (7), (14), and (17) into (4), and discarding the constant terms of $\sum_{i=1}^m \ln(1 - p_{s_i})$ and $\sum_{t_j \in T_p} h_j$, the SMFD problem simplifies to

$$\begin{aligned} \hat{\mathbf{x}} &= \arg \max_{\mathbf{x}, \mathbf{y}} J(\mathbf{x}, \mathbf{y}), \\ \text{subject to:} & \\ \ln(y_j) &= h_j + \sum_{i=1}^m \beta_{ij} x_i \quad \forall t_j \in T_f. \end{aligned} \quad (18)$$

where $J(\mathbf{x}, \mathbf{y})$, the primal objective function, is as follows:

$$J(\mathbf{x}, \mathbf{y}) = \sum_{t_j \in T_f} \ln(1 - y_j) + \sum_{t_j \in T_p} \sum_{i=1}^m \beta_{ij} x_i + \sum_{i=1}^m \ln(p_i) x_i. \quad (19)$$

Before we proceed to solve this problem, let us consider the case where the tests are perfect, i.e., $Pd_{ij} = 1$ and $Pf_{ij} = 0$, for components associated with tests.

Property 1 *If tests are perfect, for any passed test t_j , i.e., $t_j \in T_p$, the set of all components that are associated with test t_j should be healthy.*

Proof Using the definition of perfect test, if a component c_i is faulty, i.e., $x_i = 1$, and if it is associated with test t_j , i.e., $Pd_{ij} = 1$, then test t_j should fail. In other words, $x_i = 1 \Rightarrow t_j = \text{fail}$, whose contrapositive is $t_j = \text{pass} \Rightarrow x_i = 0$. Therefore, $x_i = 0 \quad \forall t_j \in T_p \ \& \ Pd_{ij} > 0$; in other words, for any test t_j where $t_j \in T_p$, the set of all components that are associated with test t_j , i.e., $Pd_{ij} > 0$, should be healthy. \square

Property 1 substantially reduces the cardinality of failure sources, S , by discarding the failure sources covered by passed tests. Let the reduced set of failure sources be denoted by S^- .

Property 2 *If tests are perfect, for any failed test t_j , i.e., $t_j \in T_f$, among all components associated with the failed test, i.e., $Pd_{ij} > 0$, at least one should be faulty.*

Proof Since, for perfect tests, we have $Pf_{ij} = 0 \quad \forall i, j$, $h_j = \sum_{i=1}^m \ln(\overline{Pf_{ij}}) = \sum_{i=1}^m \ln(1) = 0$, the constraints in (16) are reduced to $\ln(y_j) = \sum_{i=1}^m \beta_{ij} x_i \quad \forall t_j \in T_f$. Suppose all components that are associated with the failed test t_j are healthy, that

is, $x_i = 0 \quad \forall x_i, Pd_{ij} > 0$, then, $\ln(y_j) = 0$. Equivalently, $\ln(1 - y_j)$ in the objective function (19) will be unbounded. Therefore, for any $t_j \in T_f$, for all $Pd_{ij} > 0$ at least one of the x_i 's should be 1. \square

This condition can be compactly expressed as $\sum_{x_i \in S^-} Pd_{ij} x_i \geq 1 \quad \forall t_j \in T_f$. Note that for any $t_j \in T_f$, and for any $Pd_{ij} > 0$, we have $\beta_{ij} = \ln\left(\frac{Pd_{ij}}{Pf_{ij}}\right) = \ln\left(\frac{1-1}{1-0}\right) = -\infty$. Since at least one of the corresponding x_i 's is 1, $\ln(y_j) = -\infty \quad \forall t_j \in T_f$ or $y_j = 0 \quad \forall t_j \in T_f$, hence $\sum_{t_j \in T_f} \ln(1 - y_j) = \sum_{t_j \in T_f} \ln(1) = 0$.

Therefore, the SMFD problem reduces to the following set covering problem:

$$\begin{aligned} & \max \sum_{i \in S^-} \ln(p_i) x_i, \\ & \text{such that} \\ & \sum_{i \in S^-} x_i Pd_{ij} \geq 1, \quad \forall t_j \in T_f. \end{aligned} \quad (20)$$

Since the set covering problem is NP-hard, the general problem in (18) is NP-hard as well. We solve the problem in (18) using Lagrangian relaxation. Note that in (18) and in the following discussion, the tests are in general imperfect. By relaxing the constraints in (18), performing some simple manipulations, and defining $\alpha_i = \ln(p_i) + \sum_{t_j \in T_p} \beta_{ij}$, $c_i(\boldsymbol{\lambda}) = \alpha_i - \sum_{t_j \in T_f} \lambda_j \beta_{ij}$, the relaxed objective function will be as follows:

$$L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) = \sum_{t_j \in T_f} \{\ln(1 - y_j) + \lambda_j \ln(y_j) - h_j \lambda_j\} + \sum_{i=1}^m c_i(\boldsymbol{\lambda}) x_i. \quad (21)$$

The advantage of (21) is that the maximization with respect to \mathbf{x} and \mathbf{y} can be done separately. As $\lambda_j \geq 0$, by taking derivative with respect to y_j and equating to zero, we obtain $y_j^* = \frac{\lambda_j}{1 + \lambda_j}$, where y_j^* denotes the optimal value of y_j . By inserting y_j^* into (21) and simplifying, we obtain

$$L(\boldsymbol{\lambda}, \mathbf{x}) = \sum_{t_j \in T_f} \{\lambda_j \ln(\lambda_j) - (1 + \lambda_j) \ln(1 + \lambda_j) - h_j \lambda_j\} + \sum_{i=1}^m c_i(\boldsymbol{\lambda}) x_i. \quad (22)$$

In [105], the above problem is solved iteratively by initializing λ_j 's to unity, obtaining the optimal solution \mathbf{x}^* of (22) via a set covering algorithm and then updating λ_j 's via a subgradient approach until a stopping condition is met. For details, the reader is referred to [105]. Here, we discuss a purely dual approach to the problem in (22). For any given $\boldsymbol{\lambda}$, the first summation in (22) is just a constant and therefore can be

discarded, and the maximization of the second summation completely depends on the sign of $c_i(\lambda)$ terms. For any component c_i , if $c_i(\lambda)$ is positive, then $x_i^* = 1$, and if it is negative, $x_i^* = 0$. Therefore,

$$x_i^*(\lambda) = u(c_i(\lambda)) = u\left(\alpha_i - \sum_{t_j \in T_f} \lambda_j \beta_{ij}\right). \quad (23)$$

where $u(\cdot)$ is the unit step function. Hence, the SMFD problem in the dual form is as follows:

$$\begin{aligned} Q(\lambda) &= L(\lambda, \mathbf{x}^*) = Q_1(\lambda) + Q_2(\lambda), \\ Q_1(\lambda) &= \sum_{t_j \in T_f} q_j(\lambda_j), \quad q_j(\lambda_j) = \lambda_j \ln(\lambda_j) - (1 + \lambda_j) \ln(1 + \lambda_j) - h_j \lambda_j, \quad (24) \\ Q_2(\lambda) &= \sum_{i=1}^m c_i(\lambda) u(c_i(\lambda)) = \sum_{i=1}^m \max(0, c_i(\lambda)). \end{aligned}$$

Note that $c_i(\lambda)$ is negative with a high magnitude, if

- Failure source s_i has a low a priori probability, i.e., p_{s_i} is close to zero.
- Passed tests have high detection probabilities as long as false alarm probabilities are reasonable (less than 0.25).
- Failed tests have low detection probabilities as long as false alarm probabilities are reasonable (less than 0.25).

Next, we characterize some properties of the SMFD problem using the dual cost function.

Property 3 *In a real-world system, any failure source which is not associated with any of the failed tests, is healthy based on maximum likelihood estimation.*

Proof Since in a real-world system the probability of a failure source being faulty, i.e., p_{s_i} , is small (for example, around 0.01–0.2), $\ln(p_i) < 0$ and since $\beta_{ij} \leq 0 \quad \forall i, j$, $\alpha_i = \ln(p_i) + \sum_{t_k \in T_p} \beta_{ik} < 0$. If a failure source x_i is not related to any of the failed tests ($t_j \in T_f$), it is equivalent to saying that it gives neither a false alarm nor a detection to test $t_j \in T_f$, that is, $Pf_{ij} = Pd_{ij} = 0$, therefore $\beta_{ij} = \ln\left(\frac{Pd_{ij}}{Pf_{ij}}\right) = \ln(1/1) = 0 \quad \forall t_j \in T_f$. Evidently, $c_i(\lambda) = \alpha_i$, which is always negative. As a result, $x_i^* = u(c_i(\lambda^*)) = u(\alpha_i) = 0$. \square

Property 3 can be used to substantially reduce the size of the fault diagnosis problem. Let $N = \{1, 2, \dots, m\}$ and remove any failure source which is not related to any of the failed tests. Let us call the remaining indices as $S' \subseteq N$. Consequently, $Q_2(\lambda)$ can be simplified as $\sum_{i \in S'} c_i(\lambda)u(c_i(\lambda))$. Since $\alpha_i < 0 \quad \forall i$ and $\beta_{ij} \leq 0 \quad \forall i, j$, for a failure source to be inferred faulty, the norm $\|\lambda^*\|$ should be sufficiently large. As a result for a given λ^* , the more negative α_i is, the more likely component c_i is healthy. Since $\alpha_i = \ln(p_i) + \sum_{t_k \in T_p} \beta_{ik}$ if a component c_i is associated with some failed tests (that is, $i \in S'$) and if it is not associated with any passed tests (that is, $\sum_{t_k \in T_p} \beta_{ik} = 0$) then $\alpha_i = \ln(p_i) < 0$. However, if the failure source is associated with some of the passed tests (that is, $\sum_{t_k \in T_p} \beta_{ik}$) then $\alpha_i < \ln(p_i) < 0$; hence, in this case the probability of $x_i^* = 1$ reduces even further. In other words, the more passed tests a failure source s_i is associated with, the more negative α_i becomes and consequently, the less likely the failure source s_i is in state 1.

Property 4 In a real-world system, for any $t_j \in T_f$, we have $\lambda_j^* \leq \lambda_j^{\max}$, where

$$\lambda_j^{\max} = \frac{e^{h_j}}{1 - e^{h_j}}.$$

Proof Note that $Q_1(\lambda)$ is composed of $|T_f|$ separate elements, each equal to $q_j(\lambda_j)$, and each $q_j(\lambda_j)$ is a convex function whose minimum value occurs at $\frac{e^{h_j}}{1 - e^{h_j}}$. As $Q_2(\lambda)$ does not contribute a decrease in $Q(\lambda)$, increasing λ_j at most up to $\frac{e^{h_j}}{1 - e^{h_j}}$ may help in decreasing $Q(\lambda)$; thus, $\lambda_j^{\max} = \frac{e^{h_j}}{1 - e^{h_j}}$. \square

Property 5 In a real-world system, for any $i \in S'$ if we have $\alpha_i < \sum_{t_j \in T_f} \frac{\beta_{ij} e^{h_j}}{1 - e^{h_j}}$, then $x_i^* = 0$.

Proof Since $\lambda_j^* \leq \lambda_j^{\max}$ based on Property 4, therefore if $\alpha_i < \sum_{t_j \in T_f} \beta_{ij} \lambda_j^{\max}$ or $\alpha_i < \sum_{t_j \in T_f} \frac{\beta_{ij} e^{h_j}}{1 - e^{h_j}}$, then $c_i(\lambda^*) \leq c_i(\lambda^{\max}) < 0$. Thus, $x_i^* = u(c_i(\lambda^*)) = 0$. \square

Based on Property 5, we can exclude from S' any i such that $\alpha_i < \sum_{t_j \in T_f} \frac{\beta_{ij} e^{h_j}}{1 - e^{h_j}}$, call the remaining indices S'' and search for failure sources among S'' . Therefore, $Q_2(\lambda)$ can be simplified as $\sum_{i \in S''} c_i(\lambda)u(c_i(\lambda))$.

In the remainder of this section, we consider a simple example to gain insights into the SMFD problem. Consider a system with three components and two tests, where test t_1 is affected by components c_1 and c_2 ; test t_2 is affected by components c_2 and c_3 . The prior probabilities are $\mathbf{p}_s = [0.15, 0.10, 0.05]^T$, and the nonzero

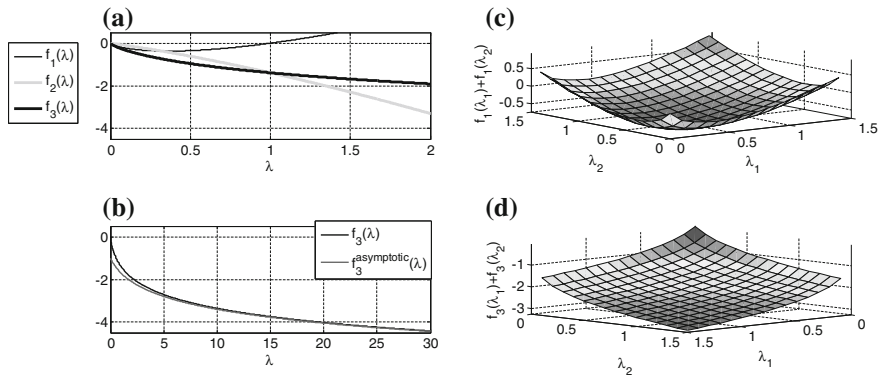


Fig. 4 Some plots regarding the dual function

detection and false alarm probabilities are $Pd_{11} = 0.85$, $Pd_{21} = 0.90$, $Pd_{22} = 0.80$, $Pd_{32} = 0.95$, $Pf_{11} = 0.06$, $Pf_{21} = 0.03$, $Pf_{22} = 0.07$, $Pf_{32} = 0.08$.

In the sequel, we consider the individual terms in the dual function to gain insights into the nature of the dual function. First, consider the function $f_1 = \lambda_j \ln(\lambda_j)$, $\lambda_j > 0$. At $\lambda_j = 1$, $f_1 = 0$. For the extreme value of $\lambda_j \rightarrow 0^+$, we have $f_1 \rightarrow 0^-$ and for the other extreme value of $\lambda_j \rightarrow +\infty$, we have $f_1 \rightarrow +\infty$. Therefore, for $\lambda_j \in (0, 1)$, the convex function f_1 is negative (see Fig. 4a) with the minimum value of $f_1^* = -e^{-1}$ occurring at $\lambda_j^* = e^{-1}$, and for $\lambda_j \in [1, \infty)$, the function monotonically increases from zero to infinity (see Fig. 4a).

Next, we consider $f_2 = -(\lambda_j + 1) \ln(\lambda_j + 1)$, which is the negative and time-shifted version of f_1 . Note that f_2 can be sketched by shifting graph of f_1 by one unit to the left and then flipping it around the x -axis; thus it is evident that for positive λ_j , the function f_2 is always negative (see Fig. 4a, b). Therefore, for $\lambda_j \in (0, 1)$ as both f_1 and f_2 are negative, the summation of these two functions, i.e., $f_3 = f_1 + f_2$, is also negative (see Fig. 4). For $\lambda_j \in [1, \infty)$, the magnitude of f_2 is always bigger than the magnitude of f_1 and as f_2 is always negative, therefore f_2 dominates f_1 . Thus, f_3 is negative for all positive values of λ_j and it goes unboundedly to $-\infty$ as $\lambda_j \rightarrow +\infty$. In fact, it can be shown that as $\lambda_j \rightarrow +\infty$, the function f_3 asymptotically approaches $f_3^{\text{asymptotic}} = -1 - \ln(\lambda_j + 1)$ (see Fig. 4b). Note that in the case of multiple faults, there exist multiple Lagrangian multipliers. First, we look into the $Q_1(\lambda)$ part of the dual function. For now, we consider $\sum_{t_j \in T_f} \{\lambda_j \ln(\lambda_j)\}$, which is actually $\sum_{t_j \in T_f} f_1(\lambda_j)$ and $\sum_{t_j \in T_f} \{\lambda_j \ln(\lambda_j) - (1 + \lambda_j) \ln(1 + \lambda_j)\}$, which is actually $\sum_{t_j \in T_f} f_3(\lambda_j)$. From the equations, it is clear that both of these functions are symmetric with respect to the Lagrange multipliers. Figure 4c, d shows the plots of these functions in a two-dimensional space (which corresponds to a two-failed-tests scenario). The plot of $\sum_{t_j \in T_f} f_1(\lambda_j)$ has a cup-shape surface which first heads down and reaches the minimum value of $-2e^{-1}$ at $\lambda^* = [e^{-1}, e^{-1}]^T$, and thereafter, it heads up and monotonically goes to infinity (see Fig. 4c).

As we saw, $f_3(\lambda)$ is a monotonically decreasing function (Fig.4b). Thus $\sum_{t_j \in T_f} f_3(\lambda_j)$ is also a monotonically decreasing function. The function for the case of the two-failed-tests scenario is shown in Fig.4d. For clarity of presentation, the direction of λ_1 and λ_2 in Fig.4d is chosen to be opposite of those in Fig.4c.

Next, we consider the effects of $-h_j\lambda_j$ in the $Q_1(\lambda)$ part of the dual function. Note that $h_j = \sum_{i=1}^m \ln(\overline{P}f_{ij})$ is always negative, and in the above example, $|h_2| > |h_1|$, because false alarm probabilities from components to test t_2 are greater than those to t_1 ($h_1 = -0.092, h_2 = -0.156$). The line $-h_j\lambda_j$ has always a positive slope, and it finally dominates f_3 , because f_3 is asymptotic to the logarithmic function $-1 - \ln(\lambda_j + 1)$; thus $(f_3 - h_j\lambda_j) \rightarrow +\infty$ as $\lambda_j \rightarrow +\infty$. Figure 5a shows the plots of $f_3(\lambda)$, $-h_1\lambda$, $-h_2\lambda$, $f_3(\lambda) - h_1\lambda$, and $f_3(\lambda) - h_2\lambda$. Figure 5b shows the $Q_1(\lambda)$ part of the dual cost function, which is asymmetric in the Lagrangian dimensions due to unequal h_j 's, with a sharper slope in the direction of λ_2 , because $|h_2| > |h_1|$. Intuitively, if the probabilities of the false alarms from different failure sources to the failed tests are high, the magnitudes of h_j 's are high, which in turn results in an increase in the slope of $-h_j\lambda_j$ line and this results in λ_j^* 's to have low values. As the parameters α_i and β_{ij} are always negative, the low values of λ_j^* 's result in more arguments of $c_i(\lambda^*)$ to become negative; hence more components will likely be healthy.

Next, we consider the $Q_2(\lambda)$ part of the dual cost function, which is $\sum_{i \in S''} (c_i(\lambda)) u(c_i(\lambda))$. Note that for each failure source of this summation, $c_i(\lambda) = 0$ is a hyperplane in $|T_f|$ dimensions; the failure source makes no contribution to the dual cost function if $c_i(\lambda) \leq 0$ (because $u(c_i(\lambda)) = 0$), while it makes a positive linear contribution if $c_i(\lambda) > 0$; thus sharp corners are created in the dual cost function and make the dual cost function nondifferentiable. In our example, as we have three components (failure sources), there exist three hyperplanes (here they are just lines because $|T_f| = 2$), one for each component. Figure 5c shows these three lines and also the place where the minimum dual cost occurs. Figure 5d shows the dual

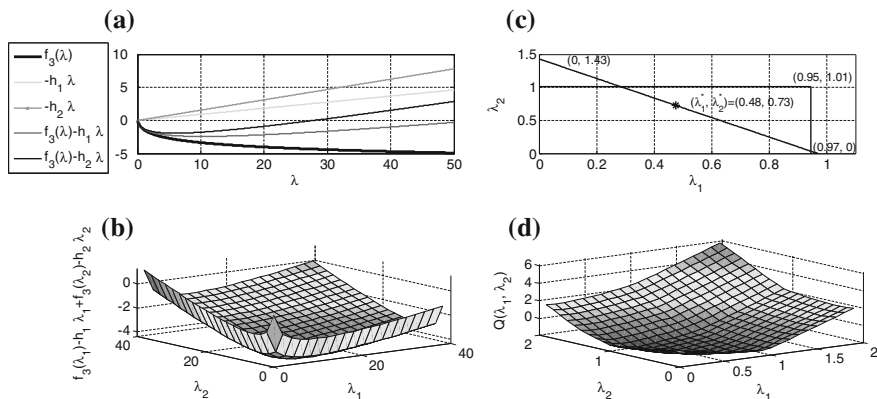


Fig. 5 Dual cost function analysis

cost function, which is obtained by adding the contributions of the nondifferentiable function $Q_2(\lambda)$ to the differentiable function $Q_1(\lambda)$ shown in Fig. 5b.

Note that, at the optimal point λ^* , if $c_i(\lambda^*)$ is negative, then $x_i^* = 0$ and if it is positive, then $x_i^* = 1$. However, for some components, we may have $c_i(\lambda^*) = 0$, and we cannot decisively assign a value to x_i^* because the contribution of $(c_i(\lambda^*)) x_i^*$ to $Q(\lambda)$ is zero irrespective of x_i^* being one or zero. For these cases, we should either check the primal cost function or use the set covering ideas. Note that for the following reasons, there are not many cases of this kind:

1. In practice, the number of failed tests $|T_f|$ is not large. As a result, the probability that $c_i(\lambda^*) = 0$ tends to be low.
2. In practice, the cause–effect relation of failure sources and tests is sparse (not all failure sources affect all tests). Therefore, not all hyperplanes have full dimension $(|T_f - 1|)$.
3. As we saw in the illustrative example, the optimal point may occur on one hyperplane or few hyperplanes whose intersection is not a point but lower dimensioned hyperplanes.
4. The possibility of the optimal point being on a low-dimension hyperplane dramatically restricts the number of hyperplanes that include the optimal point. For example, given that the optimal point is on a hyperplane with dimension one (a line), then there exist only two generic possibilities: (a) the only hyperplane that includes the optimal point is that line and (b) a line generally can intersect only a full-dimension (with dimension $|T_f| - 1$) hyperplane.

In our example, the optimal Lagrange multipliers (to three digits of accuracy) are $\lambda_1^* = 0.476$, $\lambda_2^* = 0.726$. The evaluation of $c_i(\lambda^*)$ for the three components are, respectively, -0.86 , 7.2×10^{-5} , and -0.83 . Evidently, $x_1^* = 0$ and $x_3^* = 0$. Since for x_2 , the argument is very close to zero, we should either evaluate the primal cost for both candidates of $x_2 = 1$ and $x_2 = 0$, or use the set covering idea for deciding on the assignment for x_2^* . Thus, we need to evaluate the primal cost for $x_1x_2x_3 = 000$ and $x_1x_2x_3 = 010$ and choose the one with the maximum value. The primal cost function for these two candidates are, respectively, -4.36 and -2.50 , therefore $x_1x_2x_3 = 010$ is the most likely candidate. Also, since component c_2 covers both failed tests, it should be faulty; thus we reach the same conclusion either way.

The nondifferentiability of dual cost function demands the use of numerical optimization tools, such as subgradient algorithm, to find its optimal point. Recently, surrogate Lagrangian relaxation (SLR) method [21], which provides a general purpose rapidly converging algorithm for mixed integer programming problems, has been proposed. The method, fundamentally, is based on two key ideas: (a) decreasing the distance between Lagrange multipliers in consecutive iterations, by selecting appropriate step sizes, and (b) preventing the algorithm from premature termination, by keeping step sizes sufficiently large. This algorithm has been used for solving the dual problem in (24).

3 Dynamic Multiple Fault Diagnosis

In this section, we discuss the dynamic multiple fault diagnosis (DMFD) problem. The difference between the DMFD and SMFD is that the states of the potential failure sources and the tests and their observations are functions of time. The additional assumptions for DMFD are the following:

- Time epochs of the system evolve in a discrete manner, from $k = 0$ to $k = K$.
- At any time epoch k , the state variable of component c_i (or failure source s_i) is $x_i(k)$ and the test outcomes are $t_j(k)$.
- Prior probability p_{s_i} is defined as $p_{s_i} = \Pr(x_i(0) = 1)$.
- The dynamics of states of components are assumed to be Markovian; in other words, there is a probability of fault appearance and probability of fault vanishing (disappearance) as follows:

Probability of fault appearance: $Pa_i(k) = \Pr(x_i(k) = 1 | x_i(k-1) = 0)$.

Probability of fault vanishing: $Pv_i(k) = \Pr(x_i(k) = 0 | x_i(k-1) = 1)$.

Based on the above assumptions, the DMFD problem is defined as follows:

Dynamic Multiple Fault Diagnosis (DMFD) Problem: Given a set of test observations in $K + 1$ epochs (namely T^K) where $T^K \subseteq \mathbf{t}^K$, and given the initial states of components (namely $\mathbf{x}(0)$), what is the most likely evolution of state sequence \mathbf{x}^K of each potential failure source?

Note that the observed test outcome sequence T^K may not include all of the test outcomes. Formally, we can represent the problem as follows:

$$\hat{\mathbf{x}}^K = \arg \max_{\mathbf{x}^K} \Pr(\mathbf{x}^K | T^K, \mathbf{x}(0)). \quad (25)$$

As before, using Bayes' rule, the problem is equivalent to

$$\hat{\mathbf{x}}^K = \arg \max_{\mathbf{x}^K} \Pr(T^K | \mathbf{x}^K, \mathbf{x}(0)) \Pr(\mathbf{x}^K | \mathbf{x}(0)). \quad (26)$$

Using the assumptions that (i) passed and failed tests at a given epoch and the tests at different epochs are conditionally independent, (ii) invoking the Markovian nature of failure source state evolution, and (iii) using the fact that maximizing the posterior is equivalent to maximizing the log-posterior, we can simplify (26) as follows:

$$\hat{\mathbf{x}}^K = \arg \max_{\mathbf{x}^K} \sum_{k=1}^K f_k(\mathbf{x}(k), \mathbf{x}(k-1)), \quad (27)$$

where

$$f_k(\mathbf{x}(k), \mathbf{x}(k-1)) = \ln(\Pr(T_p(k)|\mathbf{x}(k))) + \ln(\Pr(T_f(k)|\mathbf{x}(k))) + \ln(\Pr(\mathbf{x}(k)|\mathbf{x}(k-1))). \quad (28)$$

Now, let us find each of the three elements in the right-hand side of (28). The first two terms are similar to the SMFD case. Therefore,

$$\ln(\Pr(T_p(k)|\mathbf{x}(k))) = \ln(y_j(k)) = \gamma(k) + \sum_{t_j(k) \in T_p(k)} \sum_{i=1}^m \beta_{ij} x_i(k), \quad (29)$$

where

$$\gamma(k) = \sum_{t_j(k) \in T_p(k)} h_j, \quad (30)$$

$$\ln(\Pr(T_f(k)|\mathbf{x}(k))) = \sum_{i_j(k) \in T_f(k)} \ln(1 - y_j(k)). \quad (31)$$

The third term in (28), using the Markov property, can be computed as follows:

$$\ln(\Pr(\mathbf{x}(k)|\mathbf{x}(k-1))) = \sum_{i=1}^m \ln(\Pr(x_i(k)|x_i(k-1))). \quad (32)$$

As each of $x_i(k-1)$ and $x_i(k)$ has two possible values, there exist four combinations for $\Pr(x_i(k)|x_i(k-1))$. Therefore, $\Pr(x_i(k)|x_i(k-1))$ can be compactly represented as follows:

$$\Pr(x_i(k)|x_i(k-1)) = (1 - Pa_i(k))^{(1-x_i(k-1))(1-x_i(k))} (Pa_i(k))^{(1-x_i(k-1))x_i(k)} (1 - Pv_i(k))^{x_i(k-1)(1-x_i(k))} (Pv_i(k))^{x_i(k-1)x_i(k)}. \quad (33)$$

Inserting (33) into (32), and after some simplifications, we get the following formula:

$$\begin{aligned} \ln(\Pr(\mathbf{x}(k)|\mathbf{x}(k-1))) &= \sum_{i=1}^m \mu_i(k)x_i(k) + \sum_{i=1}^m \sigma_i(k)x_i(k-1) \\ &\quad + \sum_{i=1}^m h_i(k)x_i(k)x_i(k-1) + g(k), \\ \mu_i(k) &= \ln\left(\frac{Pa_i(k)}{1-Pa_i(k)}\right), \quad \sigma_i(k) = \ln\left(\frac{Pv_i(k)}{1-Pa_i(k)}\right), \\ h_i(k) &= \ln\left(\frac{(1-Pa_i(k))(1-Pv_i(k))}{Pa_i(k)Pv_i(k)}\right), \quad g(k) = \sum_{i=1}^m \ln(1 - Pa_i(k)). \end{aligned} \quad (34)$$

Thus, the DMFD problem is as follows:

$$\begin{aligned} \hat{X}^K &= \arg \max_{X^K} \sum_{k=1}^K f_k(\mathbf{x}(k), \mathbf{x}(k-1), \mathbf{y}(k)), \\ f_k(\mathbf{x}(k), \mathbf{x}(k-1)) &= \sum_{t_j(k) \in T_p(k)} \sum_{i=1}^m \beta_{ij} x_i(k) + \gamma(k) + \sum_{t_j(k) \in T_f(k)} \ln(1 - y_j(k)) \quad (35) \\ &+ \sum_{i=1}^m \mu_i(k) x_i(k) + \sum_{i=1}^m \sigma_i(k) x_i(k-1) \\ &+ \sum_{i=1}^m \varphi_i(k) x_i(k) x_i(k-1) + g(k), \end{aligned}$$

subject to

$$\ln(y_j(k)) = h_j + \sum_{i=1}^m \ln(\beta_{ij}) x_i(k). \quad (36)$$

The next step, as we did in SMFD, is to use Lagrangian relaxation. For this purpose, the constraint (36) is relaxed using Lagrange multipliers $\lambda_j(k)$. The resulting Lagrangian function is

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) &= \sum_{k=1}^K f_k(\mathbf{x}(k), \mathbf{x}(k-1), \mathbf{y}(k)) \\ &+ \sum_{t_j(k) \in T_f(k)} \lambda_j(k) \left(\ln(y_j(k)) - h_j - \sum_{i=1}^m \ln(\beta_{ij}) x_i(k) \right), \quad (37) \end{aligned}$$

where $\boldsymbol{\lambda} = \{\lambda_j(k) \geq 0, k \in \{1, \dots, K\}, t_j(k) \in T_f(k)\}$ is the set of Lagrange multipliers.

The dual of primal DMFD problem can be written as

$$\begin{aligned} \min_{\boldsymbol{\lambda}} Q(\boldsymbol{\lambda}), \\ \text{subject to: } \boldsymbol{\lambda} &= \{\lambda_j(k) \geq 0, k \in \{1, \dots, K\}, t_j(k) \in T_f(k)\}, \quad (38) \end{aligned}$$

where the dual function is

$$Q(\boldsymbol{\lambda}) = \max_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}). \quad (39)$$

Taking derivative of $L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda})$ with respect to $y_j(k)$ and equating it to zero yields the optimal $y_j^*(k)$ as $\frac{\lambda_j(k)}{1 + \lambda_j(k)}$. Inserting $\mathbf{y}^*(k)$ into (39), we get $Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}) = L(\mathbf{x}, \mathbf{y}^*, \boldsymbol{\lambda})$,

which, after some manipulation, yields

$$Q(\boldsymbol{\lambda}) = \max_{\mathbf{x}} \sum_{i=1}^m Q_i(x_i, \boldsymbol{\lambda}), \quad (40)$$

where

$$Q_i(x_i, \boldsymbol{\lambda}) = \sum_{k=1}^K \left\{ \xi_i(x_i(k), x_i(k-1), \lambda_j(k)) + \frac{1}{m} \omega_k(\boldsymbol{\lambda}) \right\}, \quad (41)$$

$$\begin{aligned} \xi_i(x_i(k), x_i(k-1), \lambda_j(k)) = & \left(\sum_{t_j(k) \in T_p(k)} \beta_{ij} + \mu_i(k) + \sum_{t_j(k) \in T_f(k)} \beta_{ij} \lambda_j(k) \right) x_i(k) \\ & + \sigma_i(k) x_i(k-1) + \varphi_i(k) x_i(k) x_i(k-1), \end{aligned} \quad (42)$$

$$\begin{aligned} \omega_k(\boldsymbol{\lambda}) = & \gamma(k) + g(k) + \sum_{t_j(k) \in T_f(k)} \lambda_j(k) \ln(\lambda_j(k)) - \lambda_j(k) h_j \\ & - \sum_{t_j(k) \in T_f(k)} (1 + \lambda_j(k)) \ln(1 + \lambda_j(k)). \end{aligned} \quad (43)$$

Note that the original problem has been converted to a separable problem in (40), where the problem is one of solving m (one problem per component) much simpler problems. The dual problem can be solved in an iterative two-level strategy where separable problems of maximization $Q_i(x_i, \boldsymbol{\lambda})$ with respect to x_i is performed using the Viterbi algorithm (dynamic programming) and then the Lagrange multipliers are updated using surrogate subgradient methods. For more details about the implementation of the algorithm, extensions of this method to coupled, delayed failure source state propagation, and delayed observations, the reader is referred to [65, 66, 110, 120, 135].

4 Fault Diagnosis in Active Probing (Sequential Fault Diagnosis) and Fault Diagnosis Applications

Passive monitoring, discussed as SMFD and DMFD in the previous sections, may still result in residual ambiguity as to sources of failure. The diagnosis from passive monitoring is followed by active probing to troubleshoot the source of failures. In this section, we consider the active probing problem used for sequential fault diagnosis and point to the applications of fault diagnosis in a number of industrial contexts, including automotive, aerospace, and power systems. In the context of a

static single-fault diagnosis problem with perfect binary tests and failure sources with binary outcomes, the test sequencing problem can be conceptualized as a four-tuple $(S, \mathbf{p}, T, \mathbf{c})$, where S is the finite set of system states composed of the fault-free state s_0 and m failure sources denoted by s_1, s_2, \dots, s_m . Associated with each failure source s_i and fault-free state s_0 is the a priori probability denoted by $\mathbf{p} = [p_0, p_1, \dots, p_m]^T$. The vector \mathbf{p} is therefore the a priori probability vector. The test set T is composed of n tests, i.e., $T = \{t_1, t_2, \dots, t_n\}$ and the cost vector $\mathbf{c} = [c_1, c_2, \dots, c_n]^T$ associates a cost to each test. The diagnostic dictionary (code book) matrix D has the dimension of $(m + 1) \times n$ whose ij -th element is one if test t_j is able to detect the fault state of s_i (i.e., s_i is associated with t_j) and zero otherwise. The problem is to design a sequential testing algorithm that unambiguously identifies the fault states using the set of tests, while minimizing the expected test cost given by

$$J = \mathbf{p}^T \mathbf{A} \mathbf{c} = \sum_{i=0}^m \sum_{j=1}^n a_{ij} p_i c_j, \quad (44)$$

where A is an $(m + 1) \times n$ matrix whose ij -th element is one if test t_j is used in the path leading to the identification of failure source s_i and zero otherwise. The problem is a perfectly observed Markov decision problem (MDP), whose solution is a deterministic AND/OR binary decision tree. In this tree, each OR node is labeled by a subset of S , which is called the ambiguity subset (state in an MDP), and each AND node denotes a test at an OR node (control or action in an MDP), and divides its input ambiguity subset into two disjoint ambiguity subsets at the output. As shown in [54], however, the construction of the optimal decision tree is NP-complete. Therefore, a way of tackling the problem is to use heuristic search strategies with tight bounds on the cost-to-go [33, 86, 122].

In [86], the test sequencing problem (TSP) is solved using an ordered, best-fit search on an AND/OR graph using different heuristic evaluation functions (HEF) based on Huffman coding and entropy. It is shown that among the HEFs used [86], a HEF based on Huffman code length is the best choice for medium-sized problems ($m < 100$) and that a HEF based on entropy plus one is suitable for larger problems. Rollout strategies have been employed to extend the range of applicability to even larger problems [119]. Reference [87] generalizes the test sequencing problem (TSP) [86] to modular diagnosis, wherein testing stops when a faulty module is isolated. The dynamic programming recursion for this generalized TSP is derived in [87] and lower bounds on the optimal cost-to-go are derived based on information theory. The problem is generalized to include test setups, precedence constraints on tests, multiple test outcomes, multiple system modes, and hierarchical test sequencing in [20, 97, 101]. In [98], the TSP was extended to consider the following cases:

- Minimize the maximum test cost.
- TSP with an upper bound on expected test time.
- TSP that achieves the lowest average ambiguity group size subject to a constraint on the number of tests.
- TSP that achieves the lowest expected test storage cost.

Reference [99] extends the test sequencing problem to the case where the tests are imperfect. Optimal and near-optimal test sequence construction methods for multiple fault diagnosis are discussed in [106]. The test sequencing problem becomes even more difficult in hybrid systems with multiple modes of operation. This is because, in a multimode system, the availability of tests depends on the mode of the system and even the same test may have different diagnostic capabilities in different modes. The multimode test sequencing problem is discussed in detail in [101]. TEAMS (Testability Engineering And Maintenance System) [32, 88] is a package designed for automatic test sequencing and testability analysis of complex modular systems for multimode systems with multivalued failure source states and multivalued test outcomes [117].

Before we close this section, we mention some of the real-world applications of fault diagnosis. Two of the applications of passive monitoring appear in tools such as QMR-DT [108, 133] and ARES-I [117]. Some of the aerospace applications of active probing include UH-60, SH-60B, and Sikorsky S92 helicopters (transmission system, engine subsystem, landing gear control unit), and a receiver synthesizer (e.g., JTIDS-RS). A few of these applications can be found in [119, 120]. Fault diagnosis in automotive systems (engine control systems, antilock braking systems, electric power generation, and storage systems) is discussed in [25, 67, 76, 77]. Model-based diagnosis of an automotive engine is discussed in [85]. In [4], the fault diagnosis technique is used for identifying and evaluating power quality problems. Reference [24] discusses fault diagnosis in heating, ventilation, and air conditioning (HVAC) systems. Application of hierarchical test sequencing in a lithographic machine can be found in [20]. TSP has been effectively used to troubleshoot problems in semiconductor fabrication facilities as well.

5 Relevant Work

In this section, we place passive monitoring and active probing discussed in the previous sections in the context of literature on fault detection and diagnosis (FDD). FDD methods have mainly evolved upon three major paradigms, viz., model-based, data-driven, and knowledge-based approaches. The FDD model-based approaches require mathematical representation of the system, hence, they are effectively applicable when satisfactory physics-based models of the system and an adequate number of sensors for state observation are available. Most applications of model-based diagnosis are restricted to systems with a relatively small number of inputs, outputs, and states. The main advantage of a model-based approach is incorporating a physical understanding into the process monitoring scheme. However, it is difficult to apply the model-based approach to large-scale systems because it requires detailed analytical models of failures in order to be effective.

The FDD data-driven approaches are preferred when system models are not available, but instead system monitoring data is available. This situation arises frequently when subsystem vendors seek to protect their intellectual property by not

providing internal system details to the system integrators. In these cases, experimental data from an operating system or simulated data from a black box simulator will be the major source of system knowledge for FDD. Neural network and statistical classification methods are illustrative of data-driven techniques. Significant amount of data is needed from monitored variables under nominal and faulty scenarios for data-driven analysis.

The FDD knowledge-based approaches require qualitative models for process monitoring and troubleshooting. These approaches are especially well suited for systems for which detailed mathematical models are not available. Most knowledge-based techniques are based on casual analysis, expert systems, and/or ad hoc rules. Because of the qualitative nature of these models, knowledge-based approaches have been applied to many complex systems. Graphical models such as Petri nets, multisignal flow graphs, and Bayesian networks are applied for diagnostic knowledge representation and inference in automotive systems. Bayesian networks subsume the deterministic fault diagnosis models embodied in the Petri net and multisignal models. Model-based, data-driven, and knowledge-based approaches provide the sand box that test designers can use to experiment with, and systematically select relevant models or combinations thereof to satisfy the requirements on diagnostic accuracy, computational speed, memory, online versus offline diagnosis, and so on. Ironically, no single technique alone can serve as the diagnostic approach for complex systems. Thus, an integrated diagnostic process that naturally employs data-driven techniques, graph-based dependency models, and mathematical/physical models is necessary for fault diagnosis, thereby enabling efficient maintenance of these systems.

The graphical methods we discussed in previous sections belong to knowledge-based methods using cause–effect relations between the failure sources and test outcomes using false alarm and detection probabilities. When the false alarm probabilities of all tests are zero, the problem simplifies to the parsimonious covering theory [34, 35, 93, 94, 102]. In [93], based on probabilistic causal methods, a competition-based connectionist method is proposed to overcome the combinatorial explosion of computing the posterior probability of all possible combinations of failure sources. This method, however, does not guarantee a global optimum and is computationally expensive, even for small problems (e.g., $m = 26$). Genetic algorithm-based methods for MFD are used in [15, 82]. These algorithms, however, converge extremely slowly and have been applied to small-size problems (e.g., $m = 20$, $n = 20$). In [132], a symptom clustering method is used which exploits the weak causal intersections in partially decomposable diagnosis structures. This approach, however, does not scale to systems with large numbers of nondecomposable causes and symptoms. In [105], the MFD problem is formulated as one of maximizing the log of the posterior probability of the hypothesized faults and the resulting constrained optimization problem is solved using Lagrangian relaxation [11, 40] and a subgradient method [12, 14]. It is shown that when tests are perfect (no false alarms and no missed detections), the MFD is reduced to a set covering problem [105]. However, it is well known that set covering problem (SCP) is NP-hard [51], and different algorithms have been proposed for SCP, including tree search procedures [10, 123], genetic algorithm [13], and greedy heuristics [27].

The dynamic fault diagnosis problem is discussed in [111, 112], using linear difference equations relating HMM and neural network-based pattern recognition. The drawback of this approach is that building a neural network for a large number of classes (here, faults) is difficult [84, 127]. Graph theory has been widely used in fault diagnosis and safety-critical systems can be modeled at an abstract level as directed graphs [23, 63, 68, 100]. In [30, 38, 114], the multiple fault diagnosis algorithms are proposed, assuming that at most k components in the system are faulty (the system is k -diagnosable). In [5, 16, 31] probabilistic models are proposed for fault diagnosis in these contexts.

The fault diagnosis problem has also been extensively studied in the control and estimation literature. A classic survey on the traditional model-based fault detection techniques is by Willsky [130]. Here, a system is represented by two sets of equations: system dynamics and sensor equations.

$$\mathbf{x}(k+1) = \Phi(k)\mathbf{x}(k) + B(k)\mathbf{u}(k) + \mathbf{w}(k), \quad (45)$$

$$\mathbf{z}(k) = H(k)\mathbf{x}(k) + J(k)\mathbf{u}(k) + \mathbf{v}(k), \quad (46)$$

where \mathbf{x} , \mathbf{u} , and \mathbf{z} are, respectively, the state vector, input vector, and measurement vector, Φ , B , H , and J are matrices, \mathbf{w} and \mathbf{v} are zero-mean, independent, white Gaussian noise processes, defined by the following covariances [130]:

$$E\{\mathbf{w}(k)\mathbf{w}^T(j)\} = Q\delta(k, j), \quad E\{\mathbf{v}(k)\mathbf{v}^T(j)\} = R\delta(k, j), \quad (47)$$

where $\delta(k, j)$ is the Kronecker delta function, which is “one” if $k = j$ and “zero” otherwise. Equations (45)–(46) represent the “normal operation” or “no failure” model of the system [130]. A failure is defined as an abrupt change in the behavior of the system which could be caused, for example, by a malfunction in actuators, plant, or sensors. The failure diagnosis problem here is comprised of three tasks: alarm, isolation, and estimation [130]. The alarm task is a binary decision of existence or nonexistence of failure in the system. The isolation task is determining the source of failure, and the estimation task is to evaluate the extent of failure; for example, is it a complete failure such as a sensor burnout or is it a partial failure such as a sensor bias? [130]. In recent terminology, however, “alarm” is often referred to as “detection,” and “isolation with or without estimation” as “diagnosis.” Among elementary algorithms for failure detection are the Shewhart control chart, geometric moving average (GMA), finite moving average (FMA), filtered derivative algorithm, and some more advanced approaches such as cumulative sum (CUSUM)-type algorithms, Bayesian-type algorithms, and generalized likelihood ratio (GLR) test [6]. The traditional approaches discussed in [130] include “failure-sensitive” filters [9, 37, 59, 60, 62, 64, 116], voting systems (for systems with high degree of redundancy in parallel hardware), multiple hypothesis filter-detectors [2, 28, 74], jump process techniques [18, 19], and innovation-based detection systems [52, 79, 80, 96, 115, 131].

The failure-sensitive filters are categorized into two groups: indirect and direct approaches. Indirect failure detection approaches, such as exponentially age-weighted filter [37, 116], limited memory filter [59], noise covariance increase [60], respond faster than a normal filter and one can make failure detection decision by abrupt changes in state estimates. Direct failure detection approaches, however, assign “failure states” to failure modes (e.g., bias onset in a sensor), and failure is detected once a failure state deviates notably from its nominal value [64]. This method provides failure alarm, isolation, and estimation, all at once, at the cost of dimensionality enlargement and also performance degradation during normal conditions [130]. A systematic direct approach, which is applicable to a wide variety of abrupt changes in linear time-invariant (LTI) systems, is discussed in [9, 62], where a filter, with the dynamical form of (48), is assigned to the LTI system of $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$ with the measurement equation of $\mathbf{z}(t) = \mathbf{C}\mathbf{x}(t)$.

$$\frac{d}{dt}\hat{\mathbf{x}}(t) = \mathbf{A}\hat{\mathbf{x}}(t) + D(\mathbf{z}(t) - \mathbf{C}\hat{\mathbf{x}}(t)) + \mathbf{B}\mathbf{u}(t). \quad (48)$$

Here, the gain matrix D is designed in a manner to highlight the effects of certain failures in the residuals of $\mathbf{z}(t) - \mathbf{C}\hat{\mathbf{x}}(t)$. In other words, D is chosen so that specific failure modes have distinct directions (“signatures”) in the space of residuals [130].

A geometrical formulation of the filter is presented in [78]. The method in [9, 62] was reformulated as an eigensystem assignment problem in [128], which greatly simplifies the design process. Multiple hypothesis filter-detectors are based on using a bank of filters based on different hypotheses for the system behavior; innovations from these hypothesized models are used to find the most likely model [130]. A simple innovation-based detection approach is the chi-squared test [79, 129]. The chi-squared test is an alarm method with a binary decision output and it is useful in detecting failure modes that have noticeable effects on innovations, but is not sensitive in detecting subtle failure modes [130]. The drawbacks of the simple chi-squared test was partly the motivation for developing the generalized likelihood ratio (GLR) test [129, 131]; a modified version of GLR test was proposed in [7] to overcome the two drawbacks of GLR, namely the coupling effect between the window size and hypothesis testing threshold and the possibly high sensitivity to the hypothesis testing threshold [8]. Since both faults and model uncertainties affect the residuals, the task of a “robust” FDI system is to be sensitive to faults and insensitive to uncertainties [90]. The various aspects of robustness in fault diagnosis systems are discussed in [26, 43, 90].

Another line of attack for fault diagnosis is the knowledge-based approaches using artificial intelligence techniques, such as qualitative reasoning [17, 46, 47, 73], fuzzy systems [107], and neural networks [44, 83]. The QSIM algorithm is a purely qualitative algorithm that is used in the medical context [70–72]. An example of the use of qualitative reasoning in the automotive industry is [113]. Fault diagnosis using fuzzy and neuro-fuzzy methods are discussed in [3, 36, 57, 75, 92]. Both shallow knowledge and deep knowledge fuzzy models are used for fault diagnosis [36]. The neural network is another fault diagnosis method, which acts as a mapping from

observations of sensor outputs and the alarms (which themselves are the outputs of some fault detection systems) to the faults or the hypothesized failure modes. However, the use of neural networks as a fault diagnosis tool is viable only in the absence of an accurate system model and abundance of process history data [121].

Another work related to the diagnosis problem is discriminability, diagnosability, and optimal sensor placement [69, 118]. Discriminability level of a system is the number of faults that can be discriminated given a set of sensors [118]. Diagnosability degree is how the discriminability level is related to the total anticipated faults in the system [118], and sensor placement deals with optimal placement of sensors to increase the diagnosability of the system.

Another direction in the literature is diagnosis approaches for discrete event system (DES) [61, 103, 104, 109] which are based on the hypothesis that any executed faulty event in a DES is diagnosed within a bounded number of state transitions/events [61].

For more information on fault diagnosis methods, the interested reader is referred to the following papers and books: [1, 6, 8, 22, 26, 29, 39, 41–43, 45, 48–50, 53, 55, 56, 58, 60, 73, 81, 83, 89–91, 95, 103, 124–126, 130, 134].

6 Summary

In this chapter, we discussed the problem of fault diagnosis in complex systems using knowledge-based probabilistic graphical models in two different contexts: static and dynamic. The fault diagnosis problem is represented using a tripartite probabilistic graphical model. The first layer of this tripartite graph is composed of components of the system, which are the potential sources of failures. The healthy or faulty condition of each component is represented by a binary state variable which is zero if the component is healthy and one otherwise. The second layer is composed of tests with binary outcomes (pass or fail) and the third layer is the noisy observations associated with the test outcomes. The cause–effect relations between the states of the components and the test outcomes can be compactly modeled in terms of detection and false alarm probabilities. When the probability of fault detection is one and the probability of false alarm is zero, the test is termed perfect; otherwise, it is deemed imperfect. In the case of perfect tests, the static multiple fault diagnosis (SMFD) problem reduces to a set covering problem, which itself is an NP-hard problem. We discussed the SMFD problem in its general form by maximizing the posterior probability of component states given the fail or pass outcomes of tests. Since the solution to this problem is known to be NP-hard, we used a Lagrangian (dual) relaxation technique to find near-optimal diagnostic solutions, which has the desirable property of providing a measure of suboptimality in terms of the approximate duality gap. Indeed, the solution would be optimal if the approximate duality gap is zero. The static problem is discussed in detail and a pure dual cost function is derived. By presenting some graphical illustrations, we provided insights into the properties of the nondifferentiable dual function.

We also discussed the multiple fault diagnosis in a dynamic context (DMFD), where it is assumed that the states of components evolve as independent Markov chains and that, at each time epoch, we have access to some of the test outcomes. Finally, we discussed the fault diagnosis problem in the context of active probing (also termed sequential testing or troubleshooting), where information is sequentially acquired to isolate the faults in minimum time, cost, or other economic factors, and we briefly mentioned some of the applications of fault diagnosis.

Acknowledgments The work reported in this chapter was partially supported by NSF grants ECCS-0931956 (NSF CPS), ECCS-1001445 (NSF GOALI), CCF-1331850 (NSF CyberSEES). Pattipati's work was also supported by ONR grant N00014-10-1-0029, ONR grant N00014-12-1-0238, and QUBE grant N00173-12-2-C902. We thank NSF, ONR, and QUBE for their support of this work. Any opinions expressed in this chapter are solely those of the authors and do not represent those of the sponsors.

References

1. Angeli C, Chatzinikolaou A (2004) On-line fault detection techniques for technical systems: a survey. *IJCSA* 1(1):12–30
2. Athans M, Dunn KP, Greene CS, Lee WH, Sandell NR, Segall I, Willsky AS (1975) The stochastic control of the F-8C aircraft using the multiple model adaptive control (MMAC) method. In: *IEEE conference on decision and control including the 14th symposium on adaptive processes*, pp 217–228
3. Ayoubi M, Isermann R (1997) Neuro-fuzzy systems for diagnosis. *Fuzzy sets Syst* 89(3):289–307
4. Azam M, Tu F, Pattipati KR, Karanam R (2004) A dependency model based approach for identifying and evaluating power quality problems. *IEEE Trans Power Deliv* 19(3):1154–1166
5. Baah GK, Podgursk A, Harrold MJ (2010) The probabilistic program dependence graph and its application to fault diagnosis. *IEEE Trans Softw Eng* 36(4):528–545
6. Basseville M, Nikiforov IV (1993) *Detection of abrupt changes: theory and application*, vol 104. Prentice Hall, Englewood Cliffs
7. Basseville M, Benveniste A (1983) Design and comparative study of some sequential jump detection algorithms for digital signals. *IEEE Trans Acoust Speech Signal Process* 31(3):521–535
8. Basseville M (1988) Detecting changes in signals and systems—a survey. *Automatica* 24(3):309–326
9. Beard RV (1971) *Failure accommodation in linear systems through self-reorganization*. Doctoral dissertation, Massachusetts Institute of Technology
10. Beasley JE (1987) An algorithm for set covering problem. *Eur J Oper Res* 31(1):85–93
11. Beasley JE (1990) A lagrangian heuristic for set-covering problems. *Naval Res Logist (NRL)* 37(1):151–164
12. Beasley JE, Jörnsten K (1992) Enhancing an algorithm for set covering problems. *Eur J Oper Res* 58(2):293–300
13. Beasley JE, Chu PC (1996) A genetic algorithm for the set covering problem. *Eur J Oper Res* 94(2):392–404
14. Bertsekas Dimitri P (1999) *Nonlinear programming*. Athena scientific, Belmont
15. Binglin Z, Tinglu Y, Ren H, Brandon JA (1993) A genetic algorithm for diagnosis problem solving. In: *Proceedings of the international conference on systems, man and cybernetics, systems engineering in the service of humans*, pp 404–408

16. Blough DM, Pelc A (1992) Complexity of fault diagnosis in comparison models. *IEEE Trans Comput* 41(3):318–324
17. Bobrow DG (ed) (2012) *Qualitative reasoning about physical systems*. Elsevier
18. Boel R, Varaiya P, Wong E (1975) Martingales on jump processes. I: representation results. *SIAM J control* 13(5):999–1021
19. Boel R, Varaiya P, Wong E (1975) Martingales on jump processes. II: applications. *SIAM J Control* 13(5):1022–1061
20. Boumen R, Ruan S, de Jong I, Van De Mortel-Fronczak JM, Rooda JE, Pattipati KR (2009) Hierarchical test sequencing for complex systems. *IEEE Trans Syst Man Cybern Part A Syst Hum* 39(3):640–649
21. Bragin MA, Luh PB, Yan JH, Yu N, Stern GA (2015) Convergence of the surrogate Lagrangian relaxation method. *J Optim Theory Appl* 164(1):173–201
22. Chen J, Patton RJ (2012) *Robust model-based fault diagnosis for dynamic systems*. Springer Publishing Company, Incorporated
23. Chessa S, Santi P (2001) Operative diagnosis of graph-based systems with multiple faults. *IEEE Trans Syst Man Cybern Part A Syst Hum* 31(2):112–119
24. Choi K, Namburu M, Azam M, Luo J, Pattipati K, Patterson-Hine A (2004) Fault diagnosis in HVAC chillers using data-driven techniques. In: *Proceedings of the AUTOTESTCON*. IEEE, pp 407–413
25. Choi K, Singh S, Kodali A, Pattipati KR, Sheppard JW, Namburu SM, Chigusa S, Prokhorov DV, Qiao L (2009) Novel classifier fusion approaches for fault diagnosis in automotive systems. *IEEE Trans Instrum Meas* 58(3):602–611
26. Chow E, Willsky AS (1984) Analytical redundancy and the design of robust failure detection systems. *IEEE Trans Autom Control* 29(7):603–614
27. Chvatal V (1979) A greedy heuristic for the set-covering problem. *Math Oper Res* 4(3):233–235
28. Clark RN, Fosth DC, Walton VM (1975) Detecting instrument malfunctions in control systems. *IEEE Trans Aerosp Electron Syst* 4:465–473
29. Dai X, Gao Z (2013) From model, signal to knowledge: a data-driven perspective of fault detection and diagnosis. *IEEE Trans Ind Inform* 9(4):2226–2238
30. Dahbura AT, Masson GM (1984) An $O(n^{2.5})$ fault identification algorithm for diagnosable systems. *IEEE Trans Comput* 100(6):486–492
31. Dahbura AT, Sabnani KK, King LL (1987) The comparison approach to multiprocessor fault diagnosis. *IEEE Trans Comput* 100(3):373–378
32. Deb S, Pattipati KR, Raghavan V, Shakeri M, Shrestha R (1995) Multi-signal flow graphs: a novel approach for system testability analysis and fault diagnosis. *IEEE Aerosp Electron Syst Mag* 10(5):14–25
33. De Faria JM, Hartmann CR, Gerberich CL, Varshney P (1980) An information theoretic approach to the construction of efficient decision trees
34. De Kleer J, Brown JS (1984) A qualitative physics based on confluences. *Artif Intell* 24(1):7–83
35. De Kleer J, Williams BC (1987) Diagnosing multiple faults. *Artif Intell* 32(1):97–130
36. Dexter AL (1995) Fuzzy model based fault diagnosis. *IEE Proc-Control Theory Appl* 142(6):545–550
37. Fagin SL (1964) Recursive linear regression theory, optimal filter theory, and error analysis of optimal systems. In: *IEEE international convention record, vol 12, part 1*, pp 216–245
38. Fedi G, Giomi R, Luchetta A, Manetti S, Piccirilli MC (1998) On the application of symbolic techniques to the multiple fault location in low testability analog circuits. *IEEE Trans Circ Syst II Analog Digit Signal Proc* 45(10):1383–1388
39. Fink PK, Luth JC (1987) Expert systems and diagnostic expertise in the mechanical and electrical domains. *IEEE Trans Syst Man Cybern* 17(3):340–349
40. Fisher ML (2004) The Lagrangian relaxation method for solving integer programming problems. *Manag Sci* 50(12 supplement):1861–1871

41. Frank PM (1990) Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy: a survey and some new results. *Automatica* 26(3):459–474
42. Frank PM (1996) Analytical and qualitative model-based fault diagnosis survey and some new results. *Eur J control* 2(1):6–28
43. Frank PM, Ding X (1997) Survey of robust residual generation and evaluation methods in observer-based fault detection systems. *J process control* 7(6):403–424
44. Frank PM, Köppen-Seliger B (1997) New developments using AI in fault diagnosis. *Eng Appl Artif Intell* 10(1):3–14
45. Frank PM, Ding SX, Marcu T (2000) Model-based fault diagnosis in technical processes. *Trans Inst Meas Control* 22(1):57–101
46. Forbus KD (1984) Qualitative process theory. *Artif Intell* 24(1):85–168
47. Forbus KD (1987) Interpreting observations of physical systems. *IEEE Trans Syst Man Cybern* 17(3):350–359
48. Gertler JJ (1988) Survey of model-based failure detection and isolation in complex plants. *IEEE Control Syst Mag* 8(6):3–11
49. Gertler J (1998) *Fault detection and diagnosis in engineering systems*. CRC Press
50. Gao Z, Cecati C, Ding S (2015) A survey of fault diagnosis and fault-tolerant techniques part I: fault diagnosis. *IEEE Trans Ind Electron* 62(6):3757–3767
51. Garey MR, Johnson DS (1979) Computer and intractability. A guide to the theory of NP-completeness
52. Handschin E, Schweppe FC, Kohlas J, Fiechter A (1975) Bad data analysis for power system state estimation. *IEEE Trans Power Apparatus Syst* 94(2):329–337
53. Hwang I, Kim S, Kim Y, Seah CE (2010) A survey of fault detection, isolation, and reconfiguration methods. *IEEE Trans Control Syst Technol* 18(3):636–653
54. Hyafil L, Rivest RL (1976) Constructing optimal binary decision trees is NP-complete. *Inf Process Lett* 5(1):15–17
55. Isermann R (1984) Process fault detection based on modeling and estimation methods a survey. *Automatica* 20(4):387–404
56. Isermann R, Balle P (1997) Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Eng Pract* 5(5):709–719
57. Isermann R (1998) On fuzzy logic applications for automatic control, supervision, and fault diagnosis. *IEEE Trans Syst Man Cybern Part A Syst Hum* 28(2):221–235
58. Isermann R (2005) Model-based fault-detection and diagnosis status and applications. *Annu Rev Control* 29(1):71–85
59. Jazwinski AH (1968) Limited memory optimal filtering. *IEEE Trans Autom Control* 13(5):558–563
60. Jazwinski AH (2007) *Stochastic processes and filtering theory*. Courier Corporation
61. Jiang S, Kumar R (2004) Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *IEEE Trans Autom Control* 49(6):934–945
62. Jones HL (1973) *Failure detection in linear systems*. Doctoral dissertation, Massachusetts Institute of Technology
63. Jordan MI (1998) *Learning in graphical models*. MIT Press, Cambridge
64. Kerr TH (1974) A two ellipsoid overlap test for real time failure detection and isolation by confidence regions. In: *IEEE conference on decision and control including the 13th symposium on adaptive processes*. IEEE, pp 735–742
65. Kodali A, Pattipati KR, Singh S (2013) Coupled factorial hidden Markov models (CFHMM) for diagnosing multiple and coupled faults. *IEEE Trans Syst Man Cybern Syst* 43(3):522–534
66. Kodali A, Singh S, Pattipati KR (2013) Dynamic set-covering for real-time multiple fault diagnosis with delayed test outcomes. *IEEE Trans Syst Man Cybern Syst* 43(3):547–562
67. Kodali A, Zhang Y, Sankavaram C, Pattipati KR, Salman M (2013) Fault diagnosis in the automotive electric power generation and storage system (EPGS). *IEEE/ASME Trans Mechatron* 18(6):1809–1818
68. Kokawa M, Miyazaki S, Shingai S (1983) Fault location using digraph and inverse direction search with application. *Automatica* 19(6):729–735

69. Krysander M, Frisk E (2008) Sensor placement for fault diagnosis. *IEEE Trans Syst Man Cybern Part A Syst Hum* 38(6):1398–1410
70. Kuipers B (1985) The limits of qualitative simulation. In: *IJCAI*, pp 128–136
71. Kuipers B (1986) Qualitative simulation. *Artif Intell* 29(3):289–338
72. Kuipers B (1987) Qualitative simulation as causal explanation. *IEEE Trans Syst Man Cybern* 17(3):432–444
73. Kuipers B (1994) *Qualitative reasoning: modeling and simulation with incomplete knowledge*. MIT press
74. Lainiotis DG (1971) Joint detection, estimation and system identification. *Inf control* 19(1):75–92
75. Leonhardt S, Ayoubi M (1997) Methods of fault diagnosis. *Control Eng Pract* 5(5):683–692
76. Luo J, Pattipati KR, Qiao L, Chigusa S (2007) An integrated diagnostic development process for automotive engine control systems. *IEEE Trans Syst Man Cybern Part C Appl Rev* 37(6):1163–1173
77. Luo J, Namburu M, Pattipati KR, Qiao L, Chigusa S (2010) Integrated model-based and data-driven diagnosis of automotive antilock braking systems. *IEEE Trans Syst Man Cybern Part A Syst Hum* 40(2):321–336
78. Massoumnia MA (1986) A geometric approach to the synthesis of failure detection filters. *IEEE Trans Autom Control* 31(9):839–846
79. Mehra RK, Peschon J (1971) An innovations approach to fault detection and diagnosis in dynamic systems. *Automatica* 7(5):637–640
80. Merrill HM (1972) Bad data suppression in state estimation, with applications to problems in power. Doctoral dissertation, Massachusetts Institute of Technology
81. Merrill WC (1985) Sensor failure detection for jet engines using analytical redundancy. *J Guidance Control Dyn* 8(6):673–682
82. Miller JA, Potter WD, Gandham RV, Lapena CN (1993) An evaluation of local improvement operators for genetic algorithms. *IEEE Trans Syst Man Cybern* 23(5):1340–1351
83. Milne R (1987) Strategies for diagnosis. *IEEE Trans Syst Man Cybern*
84. Ng K, Lippmann RP (1991) A comparative study of the practical characteristics of neural network and conventional pattern classifiers. In: *Advances in neural information processing systems*, pp 970–976
85. Nyberg M (2002) Model-based diagnosis of an automotive engine using several types of fault models. *IEEE Trans Control Syst Technol* 10(5):679–689
86. Pattipati KR, Alexandridis MG (1990) Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Trans Syst Man Cybern* 20(4):872–887
87. Pattipati KR, Dontamsetty M (1992) On a generalized test sequencing problem. *IEEE Trans Syst Man Cybern* 22(2):392–396
88. Pattipati KR, Raghavan V, Shakeri M, Deb S, Shrestha R (1994) TEAMS: testability engineering and maintenance system. In: *American control conference*, vol 2. IEEE, pp 1989–1995
89. Patton RJ (1991) Fault detection and diagnosis in aerospace systems using analytical redundancy. *Comput Control Eng J* 2(3):127–136
90. Patton RJ (1997) Robustness in model-based fault diagnosis: the 1995 situation. *Annu Rev Control* 21:103–123
91. Patton RJ, Clark RN, Frank PM (eds) (2000) *Issues of fault diagnosis for dynamic systems*. Springer Science and Business Media
92. Patton RJ, Uppal FJ, Lopez-Toribio CJ (2000) Soft computing approaches to fault diagnosis for dynamic systems: a survey. In: *Proceedings of the 4th IFAC symposium on fault detection supervision and safety for technical processes*, pp 198–211
93. Peng Y, Reggia JA (1987) A probabilistic causal model for diagnostic problem solving part I: integrating symbolic causal inference with numeric probabilistic inference. *IEEE Trans Syst Man Cybern* 17(2):146–162
94. Peng Y, Reggia JA (1987) A probabilistic causal model for diagnostic problem solving part II: diagnostic strategy. *IEEE Trans Syst Man Cybern* 17(3):395–406

95. Peng Y, Reggia JA (1989) A connectionist model for diagnostic problem solving. *IEEE Trans Syst Man Cybern* 19(2):285–298
96. Peterson DW (1975) Hypothesis, estimation, and validation of dynamic social models: energy demand modeling. Doctoral dissertation, Massachusetts Institute of Technology
97. Raghavan V, Shakeri M, Pattipati K (1999) Optimal and near-optimal test sequencing algorithms with realistic test models. *IEEE Trans Syst Man Cybern Part A Syst Hum* 29(1):11–26
98. Raghavan V, Shakeri M, Pattipati KR (1999) Test-sequencing problems arising in design planning and design for testability. *IEEE Trans Syst Man Cybern* 29(2):153–163
99. Raghavan V, Shakeri M, Pattipati KR (1999) Test sequencing algorithms with unreliable tests. *IEEE Trans Syst Man Cybern Part A Syst Hum* 29(4):347–357
100. Rao NS (1996) On parallel algorithms for single-fault diagnosis in fault propagation graph systems. *IEEE Trans Parallel Distrib Syst* 7(12):1217–1223
101. Ruan S, Tu F, Pattipati KR, Patterson-Hine A (2004) On a multimode test sequencing problem. *IEEE Trans Syst Man Cybern Part B Cybern* 34(3):1490–1499
102. Reiter R (1987) A theory of diagnosis from first principles. *Artif Intell* 32(1):57–95
103. Sampath M, Sengupta R, Lafortune S, Sinnamohideen K, Teneketzis D (1995) Diagnosability of discrete-event systems. *IEEE Trans Autom Control* 40(9):1555–1575
104. Sampath M, Sengupta R, Lafortune S, Sinnamohideen K, Teneketzis DC (1996) Failure diagnosis using discrete-event models. *IEEE Trans Control Syst Technol* 4(2):105–124
105. Shakeri M, Pattipati KR, Raghavan V, Patterson-Hine A (1998) Optimal and near-optimal algorithms for multiple fault diagnosis with unreliable tests. *IEEE Trans Syst Man Cybern Part C Appl Rev* 28(3):431–440
106. Shakeri M, Raghavan V, Pattipati KR, Patterson-Hine A (2000) Sequential testing algorithms for multiple fault diagnosis. *IEEE Trans Syst Man Cybern Part A Syst Hum* 30(1):1–14
107. Shen Q, Leitch R (1993) Fuzzy qualitative simulation. *IEEE Trans Syst Man Cybern* 23(4):1038–1061
108. Shwe MA, Middleton B, Heckerman DE, Henrion M, Horvitz EJ, Lehmann HP, Cooper GF (1991) Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base. *Methods Inform Med* 30(4):241–255
109. Sinnamohideen K (1991) Discrete-event based diagnostic supervisory control system. In: AICHE annual meeting, Los Angeles
110. Singh S, Kodali A, Choi K, Pattipati KR, Namburu SM, Sean SC, Prokhorov DV, Qiao L (2009) Dynamic multiple fault diagnosis: mathematical formulations and solution techniques. *IEEE Trans Syst Man Cybern Part A Syst Hum* 39(1):160–176
111. Smyth P (1994) Hidden Markov models for fault detection in dynamic systems. *Pattern Recogn* 27(1):149–164
112. Smyth P (1994) Markov monitoring with unknown states. *IEEE J Sel Areas Commun* 12(9):1600–1612
113. Struss P, Price C (2003) Model-based systems in the automotive industry. *AI Mag* 24(4):17–34
114. Sullivan GF (1988) A $O(t^3 + |E|)$ fault identification algorithm for diagnosable systems. *IEEE Trans Comput* 37(4):388–397
115. Schweppe FC, Handschin EJ (1974) Static state estimation in electric power systems. *Proc IEEE* 62(7):972–982
116. Tarn TJ, Zaborszky J (1970) A practical nondiverging filter. *AIAA J* 8(6):1127–1133
117. <http://www.teamqsi.com/tag/industrial/>
118. Travé-Massuyes L, Escobet T, Olive X (2006) Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Trans Syst Man Cybern Part A Syst Hum* 36(6):1146–1160
119. Tu F, Pattipati KR (2003) Rollout strategy for sequential fault diagnosis. *IEEE Trans Syst Man Cybern Part A Syst Hum* 33(1):86–99
120. Tu F, Pattipati KR, Deb S, Malepati VN (2003) Computationally efficient algorithms for multiple fault diagnosis in large graph-based systems. *IEEE Trans Syst Man Cybern Part A Syst Hum* 33(1):73–85

121. Ungar LH, Powell BA, Kamens SN (1990) Adaptive networks for fault diagnosis and process control. *Comput Chem Eng* 14(4):561–572
122. Varshney PK, Hartmann CRP, De Faria Jr J M (1982) Application of information theory to sequential fault diagnosis. *IEEE Trans Comput* 100(2):164–170
123. Vasko FJ, Wilson GR (1984) Using a facility location algorithm to solve large set covering problems. *Oper Res Lett* 3(2):85–90
124. Venkatasubramanian V, Rengaswamy R, Yin K, Kavuri SN (2003) A review of process fault detection and diagnosis: part I: quantitative model-based methods. *Comput Chem Eng* 27(3):293–311
125. Venkatasubramanian V, Rengaswamy R, Kavuri SN (2003) A review of process fault detection and diagnosis: part II: qualitative models and search strategies. *Comput Chem Eng* 27(3):313–326
126. Venkatasubramanian V, Rengaswamy R, Kavuri SN, Yin K (2003) A review of process fault detection and diagnosis: part III: process history based methods. *Comput Chem Eng* 27(3):327–346
127. Weiss SM, Kapouleas I (1989) An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In: *Proceedings of the international joint conference artificial intelligence*, pp 781–787
128. White JE, Speyer JL (1987) Detection filter design: spectral theory and algorithms. *IEEE Trans Autom Control* 32(7):593–603
129. Willsky AS, Deyst J, Crawford BS (1975) Two self-test methods applied to an inertial system problem. *J Spacecraft Rockets* 12(7):434–437
130. Willsky AS (1976) A survey of design methods for failure detection in dynamic systems. *Automatica* 12(6):601–611
131. Willsky AS, Jones HL (1976) A generalized likelihood ratio approach to the detection and estimation of jumps in linear systems. *IEEE Trans Autom Control* 21(1):108–112
132. Wu TD (1991) A problem decomposition method for efficient diagnosis and interpretation of multiple disorders. *Comput Methods Programs Biomed* 35(4):239–250
133. Yu F, Tu F, Tu H, Pattipati KR (2007) A Lagrangian relaxation algorithm for finding the MAP configuration in QMR-DT. *IEEE Trans Syst Man Cybern Part A Syst Hum* 37(5):746–757
134. Zhang P, Ding SX (2008) On fault detection in linear discrete-time, periodic, and sampled-data systems. *J Control Sci Eng*
135. Zhang S, Pattipati KR, Hu Z, Wen X, Sankavaram C (2013) Dynamic coupled fault diagnosis with propagation and observation delays. *IEEE Trans Syst Man Cybern Syst* 43(6):1424–1439

Part II
Principles of Performance and Reliability
Modeling and Evaluation

From Performability to Uncertainty

Raymond A. Marie

Abstract Starting from the expertise in reliability and in performance evaluation, we present the notion of performability introduced by John Meyer in his famous paper. We recall that in the past, few industry leaders believed in stochastic models, most of them placing greater confidence in the development of deterministic models and the use of coefficients of security to take into account the different uncertainties. But now, the notion of risk has been emphasized by the development of new technologies, the generalization of insurance policies, and the practice of service level agreements. Therefore, this is the time to consider stochastic models, where former deterministic parameters are replaced by random variables, with the encouragement of industrial leaders. We illustrate these latter models through two variants of a case study.

1 Motivation

Motivated by the first objective of the book, this chapter aims to present a part of the evolution of the performance and of the dependability evaluations in our field of computation, telecommunication and up to some degree in discrete events systems. Starting from the expertises in reliability and in performance evaluation,¹ we present the notion of performability introduced by John Meyer in his seminal paper. Let us remember that in ancient times, many industry leaders did not believe in stochastic models, most of them being more confident in the development of deterministic models and the use of coefficients of security to take into account the different uncertainties, or the development of worst case studies. But now, the notion of risk has been emphasized by the development of the new technologies, the generalization of insurance policies and the practice of service agreements. Therefore, the time has arrived to consider stochastic models, where former deterministic parameters are replaced by random variables, with the encouragement of industrial leaders. We

¹Which are two of Kishor Trivedi's areas of expertise, as already illustrated in his first book.

R.A. Marie (✉)

IRISA Campus de Beaulieu, Rennes University, 35042 Rennes Cedex, France
e-mail: Raymond.Marie@irisa.fr

© Springer International Publishing Switzerland 2016
L. Fiondella and A. Puliafito (eds.), *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering,
DOI 10.1007/978-3-319-30599-8_6

illustrate these latter models with the help of two variants of a case study regarding the expression of the asymptotic unavailability, where the mean up time is considered a random variable following, respectively, a uniform and a triangular distribution.

2 Emergence of the Performability

At the end of the 1960s, when the computers were very big and expensive but had low computational speed and small main memory capacities, reliability studies were still mainly applied to series-parallel structures. On the other hand, the set of system resource models developed for performance evaluation were mostly consisting of isolated queues. Most of the research was dedicated to the study of many different types of unique queues (cf. for example [10]). However, in the seventies, researchers took an active interest in the study of queueing networks, in particular with the purpose of predicting the performance of computer architectures. The new results on the so called product form queueing networks [2, 6] allowed the researchers to obtain amazingly accurate performance predictions on the main frame computers used in that epoch with time sharing service policies, despite all the simplifying assumptions made in the model constructions. A classical such queueing network model of a computer architecture of that time is shown on Fig. 1.

Meanwhile, pioneers of the reliability studies were investigating simple non-series-parallel structures with the help of the notion of decomposition introduced by Shannon. As a little example, we can consider the computer architecture illustrated in Fig. 2 which consists of two processors and two memories in parallel and of two power supplies. The first power supply A_1 being in charge of the first processor P_1 and of the first memory M_1 and the second being in charge of the second ones (we can see that the existence of the two power supplies does not allow us to express the system reliability according to a series-parallel structure). This approach was soon called the factorization method, from the fact that Shannon's decomposition theorem, when applied to reliability of redundant structures, was also called the

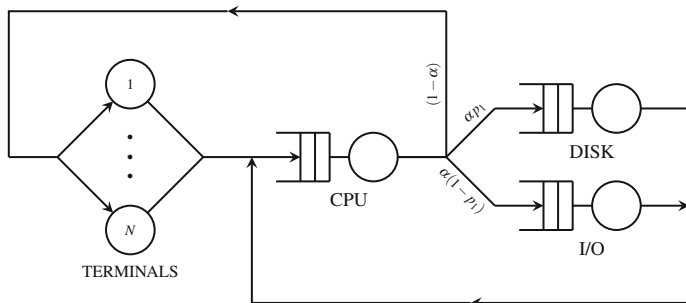


Fig. 1 Representation of a computer architecture model

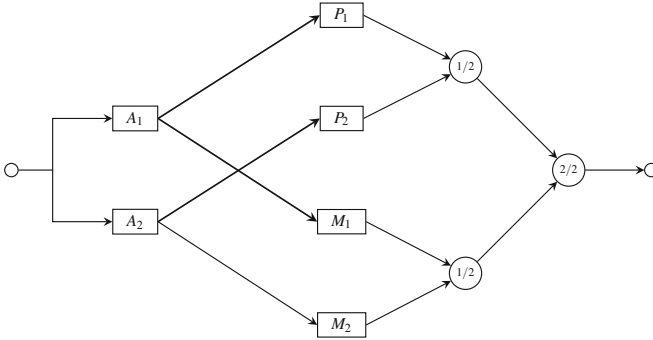


Fig. 2 Reliability diagram of the computing system

factoring theorem of Boolean algebra (Moskowitz [51]). Let us use this example to explain the factorization method. Let R_{A_i} denotes the reliability of the i th power supply, $i = 1, 2$, and R_S denotes the reliability of the system. Let us define the Boolean variables x_1 and x_2 such that, for $i = 1, 2$:

$$x_i = \begin{cases} 1 & \text{if } A_i \text{ is operational} \\ 0 & \text{if } A_i \text{ is down} \end{cases} \tag{1}$$

In the same way, we introduce the Boolean variable x_S such that we have $R_S = \mathbb{P}(x_S = 1)$. The first step of the factorization method consists of conditioning on the state of a perturbing element, such as the first power supply, and to write :

$$R_S = R_{A_1} \times \mathbb{P}(x_S = 1|x_1 = 1) + (1 - R_{A_1}) \times \mathbb{P}(x_S = 1|x_1 = 0) . \tag{2}$$

We then continue the conditioning phase writing successively

$$\begin{aligned} \mathbb{P}(x_S = 1|x_1 = 1) &= R_{A_2} \times \mathbb{P}(x_S = 1|x_1 = 1, x_2 = 1) \\ &\quad + (1 - R_{A_2}) \times \mathbb{P}(x_S = 1|x_1 = 1, x_2 = 0) , \end{aligned} \tag{3}$$

and

$$\begin{aligned} \mathbb{P}(x_S = 1|x_1 = 0) &= R_{A_2} \times \mathbb{P}(x_S = 1|x_1 = 0, x_2 = 1) \\ &\quad + (1 - R_{A_2}) \times \mathbb{P}(x_S = 1|x_1 = 0, x_2 = 0) . \end{aligned} \tag{4}$$

For any given values of x_1 and x_2 , the corresponding reliability diagram is of the series-parallel type and its reliability is easy to calculate. There are 4 cases corresponding to the 4 possible states of the tuple (x_1, x_2) but it is obvious that the reliability is null when $x_1 = x_2 = 0$. Knowing the reliability of the 3 others series-parallel structures, the value of R_S is obtained by successive deconditioning. In its generality, the factorization method consists of two phases. The first one corresponds

to the construction of a binary tree until the subsystems associated to the nodes are of the series-parallel type (these nodes becomes the leaves of the final binary tree). The reliabilities of these series-parallel subsystems are easily computed. The second phase consists in executing successive deconditioning until we get the reliability R_S of the original system associated with the root node of the binary tree.

Note that, up to the early seventies, the results were obtained for reliability diagrams that could be associated with a directed graph connecting a source node S to a terminal T (see for example Moore and Shannon [50], Moskowitz [51], Misra [49], Aggarwal et al. [1]). The components under consideration were basic elements such as relays or connections. Let us remark that the Boolean function characterizing the reliability of the structure presented in the example above does not admit a directed graph representation. In the seventies, some research was also conducted on the use of the simulation in order to determine the Source-Terminal reliability for general directed graphs modeling communication networks [61].

The appearance of multiprocessors systems played a role in the evolution of the perception of reliability, especially the arrival of the so called gracefully degradable system. Up to that time, reliability of a component was associated with two possible states: the operational and nonoperational states (also called the up and down states). Then appeared the notion of potential production of a system. In the case of computers, this notion was defined as the computation capacity of a system (see Beaudry [3]).

The idea is as follows. We assume that a computing system can be characterized in several different states belonging to a given state set \mathbb{E} . This set can be partitioned into the subset of the operational states \mathbb{E}_O and the subset of the down states \mathbb{E}_F . The computation capacity is not the same for all the operational states of the system (for example, to an operational state i may correspond a given number of operational processors). Consequently, we assume we can associate with any operational state i a parameter r_i such that

$$dT = r_i dt , \quad (5)$$

where dT is the achievable computation capacity during a time interval dt , given that the computing system is in state i . Moreover, we now assume that the behavior of the computing system has been modeled by a continuous time Markov chain (CTMC) $X(t)$ evolving on the state set \mathbb{E} , according to an infinitesimal generator $A = (a_{ij})$. If $p_i(t)$ denotes the state probability that $X(t) = i$, we know that this probability satisfies the following differential Kolmogorov equation:

$$\frac{dp_i(t)}{dt} = a_{ii} p_i(t) + \sum_{j \neq i} a_{ji} p_j(t) . \quad (6)$$

Similarly, we can consider on the computation domain a probability $P_i(T)$ defined as the probability that the computing system is in state i after having produced a computation capacity T . Assuming r_i is strictly positive and using Eq. 5, we can write

$$a_{ij}dt = b_{ij}dT , \tag{7}$$

where

$$b_{ij} = \frac{a_{ij}}{r_i} . \tag{8}$$

The evolution of the probability $P_i(T)$ on the computation domain between T and $(T + dT)$ satisfies the following differential Kolmogorov equation:

$$P_i(T + dT) = b_{ii}P_i(T)dT + \sum_{j \neq i} b_{ji}P_j(T)dT . \tag{9}$$

Doing so, we may consider a new CTMC $Z(T)$ evolving on a state set included in the previous state set \mathbb{E} , according to an infinitesimal generator $B = (b_{ij})$.

Let us consider a simple but classical example such as a two processor system with perfect coverage like the one illustrated by Fig. 3. λ is the processor failure rate and μ is the repair rate.

If we are interested in expressing the reliability of this system, we transform the model such that state 0 becomes an absorbing state and states 1 and 2 become transient states (see Fig. 4).

For a given order of the states, there exists a matrix which describes the infinitesimal generator of the CTMC. Let us focus on the sub-matrix associated to the two transient states. Ordering the states of the chain according to the order (2, 1, 0), and denoting A this sub-matrix, we have

Fig. 3 Markovian model of a two processor system

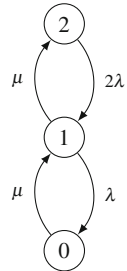
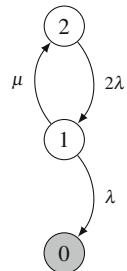


Fig. 4 Reliability model of a two processor system



$$A = \begin{bmatrix} -2\lambda & 2\lambda \\ \mu & -(\mu + \lambda) \end{bmatrix} \quad (10)$$

The reliability of the system is given by

$$R(t) = \mathbf{p}(0)e^{At} \mathbf{e}, \quad (11)$$

where $\mathbf{p}(0) = (p_2(0), p_1(0))$ and \mathbf{e} is the unity vector of dimension 2, its size being determined by the context of the equation. Note that in this little example, $R(t)$ is also equal to $(1 - p_0(t))$.

Let us now switch to the computation domain and consider the available computation capacity of this system. We assume that $r_i = i\alpha$, $i = 1, 2$. Then, with respect to these two productive states, matrix $B = (b_{ij})$ (here of order 2) is given by

$$B = \begin{bmatrix} -\frac{\lambda}{\alpha} & \frac{\lambda}{\alpha} \\ \frac{\mu}{\alpha} & -\frac{\mu+\lambda}{\alpha} \end{bmatrix} \quad (12)$$

and defining $\mathbf{P}(T) = (P_2(T), P_1(T))$, we have

$$\mathbf{P}(T) = \mathbf{P}(0)e^{BT}, \quad (13)$$

where $\mathbf{P}(0) = \mathbf{p}(0)$. The probability that the system is able to produce a computation capacity T , given that the initial conditions of the system correspond to initial vector $\mathbf{p}(0)$, is the following

$$C_S(T) = \mathbf{P}(0)e^{BT} \mathbf{e}. \quad (14)$$

Knowing this probability function, we can compute the *mean computation before failure* (MCBF) using

$$\text{MCBF} = \int_0^{\infty} C_S(u) du. \quad (15)$$

For a given operational state i , we can also determine the probability $C_i(T)$ that the system is able to execute a task T given that the system was originally in state i :

$$C_i(T) = \mathbf{e}_i e^{BT} \mathbf{e}, \quad (16)$$

where \mathbf{e}_i is a vector having its component i equal to one and all its other components equal to zero, its size being determined by the context of the equation.

Let us introduce now the notion of “computation reliability,” defined as the probability that the system is able, at time t , to execute a task T , given the initial state probability vector $\mathbf{p}(0)$ of the system. Starting from the previous result (16), it is possible to compute this new notion that we denote $\tilde{R}(t, T)$ by conditioning on the state of the system at time t :

$$\tilde{R}(t, T) = \sum_{i \in \mathbb{E}_o} p_i(t) C_i(T) . \quad (17)$$

Let us remark that $\tilde{R}(t, 0) = R(t)$ and that $\tilde{R}(0, T) = C_S(T)$.

But this innovative work presents some drawback. For example, it cannot effectively model repairable systems such as the one on Figure 3. Indeed, the method is less efficient because we cannot consider null coefficients r_i in Eq. 8. The notion of expected computation availability at time t has to be obtained from the transient solution of the CTMC $X(t)$:

$$\mathbb{E}[A_c(t)] = \sum_{i \in \mathbb{E}_o} r_i p_i(t) . \quad (18)$$

Since here the system is repairable, we assume that the CTMC $X(t)$ is irreducible. Let π_i denote the probability of being in state i in the steady state. In asymptotic behavior, the expected computation availability is given by

$$\mathbb{E}[A_c] = \sum_{i \in \mathbb{E}_o} r_i \pi_i . \quad (19)$$

Almost in the same epoch that the pioneering work of Beaudry was published, Meyer started to use the term “performability” (see [24, 48]) as a shortcut for referring to an performance-reliability measure. However, this is in a paper of August 1980 (see [45]) that Meyer described what he called *a general modeling framework that permits the definition, formulation, and evaluation of a unified performance-reliability measure referred to as “performability”*.

The model of the system under consideration is elaborated at the level of the different resources and takes into account the environment such as resource requests and the reliability of the different elements. The model is built as a stochastic dynamic one. So, at time t , the performance depends both on the reliability and environmental behaviors. It is assumed that to a state of the modeled system corresponds a value of a capability indicator and that when we observe the evolution of the model describing a random trajectory ω over a time interval $[0, T]$, we can obtain a quantitative value Y_T of the observed performance/capability over the time interval. For example, Y_T can be the percentage of requests satisfied over the time interval $[0, T]$. To summarize the notion, let us say that performability expresses performance of systems with variable capability such as gracefully degradable and eventually repairable architectures. If $Y(\omega)$ is the system performance realized on the time interval considered, Y is a random variable with probability distribution function F_Y called the *performability*. As suspected, the determination of this distribution function is generally not really an easy task! Meyer presented in 1982 (see [46]) a case study where the model structure was reduced to an acyclic Markov chain and where the performability was obtained as a closed form solution. This was the time for more transient studies.

While the heart of the problem for the specialists of performance evaluation was the determination of steady-state solutions of more and more sophisticated models, performability studies involved working on the transient behavior of the models.

3 Need for More Transient Analysis

In the 1980s, a substantial effort was invested by the research community on the performability concept, bringing, step by step, successive solutions to models of increasing complexity. A significant part of the research was done on a class of models which was soon called “Markov reward models” (MRM). Such a model corresponds to a stochastic process $Z(t)$ (generally a CTMC), designed as the model structure and to the attribution to each state i of a reward rate r_i , (a real number, the meaning of which may depends on the purpose of the study). At this point, the model seems close to the one introduced by Beaudry [3], but here we keep working on the time domain and introduce a new process $X(t)$ that gives the reward rate at time t :

$$X(t) = r_{Z(t)} . \quad (20)$$

Note that if $Z(t)$ is a CTMC, then $X(t)$ is also a CTMC (on a different state space). The key random variable is the accumulated reward up to time t :

$$Y(t) = \int_0^t X(u)du . \quad (21)$$

Again, the performability corresponds to the probability distribution function of $Y(t)$ and the challenge was to determine it. Some other measures can be associated with this probability distribution function with different levels of difficulty. For example, if we consider the expectation $\mathbb{E}[Y(t)]$ of the accumulated reward up to time t , it can be shown that

$$\mathbb{E}[Y(t)] = \sum_i r_i \int_0^t p_i(u)du . \quad (22)$$

Defining $L_i(t) = \int_0^t p_i(u)du$ as an element of the row vector $\mathbf{L}(t)$, this latter vector is the solution of the differential linear system

$$\frac{d}{dt}\mathbf{L}(t) = \mathbf{L}(t)\mathbf{A} + \mathbf{p}(0) . \quad (23)$$

This system can be solved in the same way one solves the linear differential system

$$\frac{d}{dt}\mathbf{p}(t) = \mathbf{p}(t)\mathbf{A} . \quad (24)$$

Once vector $\mathbf{L}(t)$ is obtained,

$$\mathbb{E}[Y(t)] = \mathbf{Lr} , \tag{25}$$

where \mathbf{r} is the column vector of the rewards.

Concerning the performability itself, the first general results were obtained for acyclic CTMC and monotonic rewards. Papers of Furchtgott and Meyer [23] and of Goyal and Tantawi [29] were among the first to present the corresponding results. The constraint on the monotonicity of the rewards was raised with the publications of Ciciani and Grassi [9] and of Donatiello and Iyer [21]. Note that at the same time, another challenge was to find algorithms with an acceptable complexity.

The results of cyclic CTMC (for the case of repairable systems) were not easy to obtain and arrived first for the special case of the study of the interval availability (In this special case, the reward equals one if the state is operational or equals zero if the state is nonoperational). Recall the following publications: Goyal et al. [31] in 1985, de Souza and Gail [17] in 1986, Goyal and Tantawi [30] in 1988. Note that de Souza and Gail obtained their results by using the uniformization technique. A comparison between the uniformization and numerical linear multi-step methods was made by Reibman and Trivedi (1988) [55]. Meanwhile, in 1986, Iyer et al. obtained the moments of the performability thanks to a recursive technique. Also in 1986, Kulkarni et al. [39] presented a numerical inversion of the double Laplace transform system characterizing the performability. The efficiency of this numerical algorithm was improved in a paper of Smith et al. in 1988 [58].

Using the uniformization technique, de Souza and Gail [13] presented a quite general method to obtain the performability measures of cyclic CTMC, allowing impulse-based and rate-based rewards. However, this latter approach possessed an exponential complexity with the number of different reward coefficients. However, in 1991 Donatiello and Grassi [20] proposed a new algorithm (also based on the uniformization technique) with polynomial computational complexity, but the stability of this latter algorithm seemed questionable. Almost five years later in 1996, another algorithm still based on the uniformization technique, published by Nabli and Sericola [52], exhibited stability property thanks to a transformation on the rewards values in order to deal only with nonnegative values bounded by 1 (see also the comment [59]).

Let us mention two publications that extend the domain of homogeneous Markov chains. First, there is the publication [8] of 1990 where Ciardo et al. presented an algorithm for the computation of the accumulated reward for a semi-Markov reward process. Secondly, there is the publication [53] of 1993 where Pattipati et al. obtained the distribution of the accumulated reward for a nonhomogeneous Markov reward process using hyperbolic linear partial differential equation (PDE) with time-dependent coefficients which are solved thanks to a numerical method. Meanwhile de Souza et al. [16] presented in 1995 a more efficient algorithm than the one published in [13], with linear complexity with the number of states (see in addition the extended publication of 1998 [12]). Note also the state of the art in transient solutions for Markov chain that was published later in 2000 by de Souza and Gail [15].

The development of tools, especially those based on the notion of stochastic timed Petri networks, constituted a booster on research involving the studies of performability. More information, especially on the development of tools, can be found in the three surveys published by Meyer [47] in 1992, Trivedi et al. [60] in 1993, and Haverkort and Niemegeers in 1996 [34]. The paper of de Souza and Gail [14] first develops the aspects of model specification in correlation with available software tools and also the use of uniformization to determine the different measures. In [40], Lanus et al. (2003) proposed a hierarchical construction and aggregation of models in order to control the size of models for extracting the performability measures of large systems. Let us also mention the paper of Colbourn (1999) [11] where a survey on performability is presented in the context of telecommunication network planning.

Meanwhile, several papers were published on applications or in connection with the use of software tools such as Rubino and Sericola [57], Iyer et al. [37], Hirel et al. [36]. Let us also mention the paper of Fricks et al. [22] that, in 1998, presented an introduction of performability evaluation on Markov renewal models, which do not satisfy the memoryless property at every instant. This subject was also investigated by individuals such as German and Telek (see [26–28]).

4 Sensitivity Analysis Studies

During the design of a system, sensitivity analysis helps the designer to identify the critical parameters with respect to a chosen metric, those which may be targeted for improvement, or may have dramatic consequences if parameter estimates are inaccurate.

The idea is straightforward. Considering the state probabilities $p_i(t)$, $i \in \mathbb{E}$, and a particular parameter λ , we consider the partial derivatives $\frac{\partial p_i(t)}{\partial \lambda}$ that we denote by $S_i(t)$, $i \in \mathbb{E}$. Starting from Eq. 24 and taking the partial derivatives with respect to λ we get

$$\frac{d}{dt} \mathbf{S}(t) = \mathbf{S}(t) \mathbf{A} + \mathbf{p}(t) \mathbf{V}. \quad (26)$$

where \mathbf{V} is the derivative of the infinitesimal generator \mathbf{A} , *i.e.*, $v_{ij} = \frac{d}{d\lambda} a_{ij}$. In general, the initial vector $\mathbf{S}(0)$ equals the null vector $\mathbf{0}$. For the case of an acyclic Markov chain, a semi-formal solution was proposed in 1987 by Marie et al. [44]. For the general case, we can obtain the solution from different approaches such as the uniformization technique (Heidelberger and Goyal (1987) [35]), a numerical method (Reibman et al. (1989) [54]), or a transform approach (Grassi and Donatiello (1992) [32]).

The sensitivity of the expectation of the performability $\mathbb{E}[Y(t)]$ can be expressed starting from Eq. 25 from where

$$\begin{aligned} \frac{d}{d\lambda} \mathbb{E}[Y(t)] &= \sum_i \frac{\partial r_i}{\partial \lambda} L_i(t) + \sum_i r_i \frac{\partial L_i(t)}{\partial \lambda} \\ &= \sum_i \frac{\partial r_i}{\partial \lambda} L_i(t) + \sum_i r_i \int_0^t S_i(u) du . \end{aligned} \tag{27}$$

This sensitivity can be obtained using the same methods as above. See for example Blake et al. (1988) [5], Reibman and Trivedi (1989) [56] and Haverkort and Meeuwissen (1995) [33].

These results tell us how important a parameter is with respect to a given measure. Depending on the selected measure, the parameter might be important with respect to a performance or a dependability point of view, or to an optimal solution. In general, a selected measure M_S depends of several parameters. If the changes of these parameters are assumed to be independent, we can construct a global sensitivity indicator as follows:

$$\Delta M_S = \sum_j \Delta \lambda_j \frac{\partial}{\partial \lambda_j} M_S . \tag{28}$$

For the case where the individual parameters are dependent is another challenge. Working with the field of perturbations analysis is a way to extend sensitivity analysis. For example, see the work of Do Van et al. published in 2008 [18] and in 2010 [19] on multi-directional sensitivity measures with functional dependencies.

5 Uncertainty Studies

People in charge of a project are more and more concerned with the notion of risk and with its potential consequences. There are different categories of risk: risk that a bridge breaks down due to wrong dimensioning with respect to wind forces, risk that a new product does not satisfy its clientèle, risk that the availability of a new system does not reach its predicted value, etc. In general, the risk is a consequence of the existence of uncertainties of different kinds such as: values of parameters depending on the environment, accuracy of a design variable, faithfulness of a model, choice of constraints, etc. For a long time, engineers have employed security coefficients to reinforce their projects. The use of sensitivity analysis is also an attempt to evaluate the risk of a deterioration of the performances. But ideally, when some uncertainties cannot be eliminated, the values must be considered as random and this compels us to introduce random variables in the expressions of measures.

Nevertheless, a first step is to consider intervals as input parameters and when possible propagate the determination of new intervals on new parameters, using interval arithmetic until obtaining bounds on output measures. Some publications devoted to this approach are: Majumdar and Revathy (1995) [43], Luthi et al. (1997) [42], Luthi and Haring (1998) [41]. To go further, we have to assume a probability distribution on the interval of variation of the parameter. The choice of the probability

distribution is not evident and obviously depends on the meaning of the parameter and of the type of objective. The favorites seem to be the uniform, the triangular, the Gaussian, the log-uniform and the log-normal distributions. Given the probability distributions of the uncertain parameters, the determination of the expectation of the measure of interest is generally not trivial. However, the Grail is to obtain the probability distribution of the measure of interest because the up-to-date service agreements are associated with penalties depending generally on tail distributions.

Unfortunately, not all situations lead to nice final expressions. This is why there is room for example for simulation approaches in order to evaluate the dispersion of different indicators, thanks to more or less sophisticated Monte Carlo simulations such as Yin et al. (2001) [63] and Wubbeler et al. (2008) [62]. Another way to cope with uncertainty could be to use fuzzy theories such as the fuzzy sets theory (see for example Benetto et al. (2008) [4]). Looking at the work of other communities is of interest (cf. Jackson et al. (1982) [38], Garelli and Ferrero (2012) [25] and Chen et al. (2013) [7]).

Not mentioned yet in this section, we can face uncertainty with functional dependencies between different parameters. Probably in such a situation (with more than two parameters), the solution seems to be simulation (cf. Yin et al. (2001) [63] and Haverkort and Meeuwissen (1995) [33]). Following this section, we will consider two variants of a relatively basic case study in order to be able to obtain analytical solutions with the aim to produce some interesting deductions.

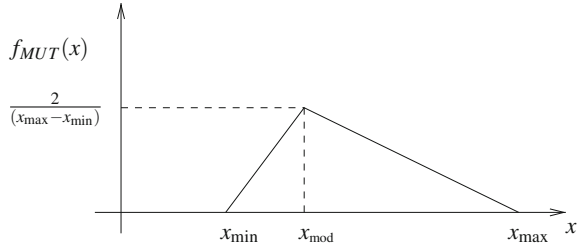
5.1 Two Variants of a Case Study

Let us consider the classic expression of the steady-state unavailability \bar{A}

$$\bar{A} = \frac{MDT}{MUT + MDT} \quad (29)$$

where MUT is the mean up time and MDT is the mean down time. For a new equipment such as a line repairable unit (LRU), the determination of the value of MUT is not evident, even for a specialist. This is why it might be more judicious to ask the expert to give a time interval $[MUT_{\min}, MUT_{\max}]$ for which he believes that this interval contains the true value of the mean up time. In such a situation, without any complementary information, it is natural to consider that MUT is a random variable following a uniform distribution $U(MUT_{\min}, MUT_{\max})$, (which maximizes the entropy of the information). If the expert has more expertise, it may also be possible to consider a different probability distribution, the triangular distribution, the density function of which increases linearly on $[MUT_{\min}, MUT_{mod}]$ and then decreases linearly on $[MUT_{mod}, MUT_{\max}]$ (cf. Fig. 5). This latter unimodal distribution gives us more information on the random variable MUT than the previous one. Here we consider the

Fig. 5 Illustration of the density function of the triangular distribution



MDT as constant because, generally, there are fewer difficulties identifying a value from past experience. So, considering the MUT as a random variable, the quantity \bar{A} becomes itself a random variable. We will now examine the consequences of this transformation when assuming, first that MUT is a random variable following a uniform distribution (case 1) and secondly that it follows a triangular distribution (case 2). As a first step, we would like to exhibit the expectation of \bar{A} .

5.1.1 Expectations of New Random Variables

Let us start by introducing some new notations. Denoting d the value of MDT, let consider the new notations a , b and Δ such that

$$a = \text{MUT}_{\min} + d, \quad b = \text{MUT}_{\max} + d, \quad \text{and} \quad \Delta = (\text{MUT}_{\max} - \text{MUT}_{\min})$$

Also, for case 2, let ρ denotes the ratio $\rho = \frac{\text{MUT}_{\text{mod}} - \text{MUT}_{\min}}{\Delta}$.

Considering the relation (29), the steady-state unavailability \bar{A} can be rewritten as $\bar{A} = \frac{d}{\text{MUT} + d}$ and takes values between \bar{A}_{\min} and \bar{A}_{\max} such that $\bar{A}_{\min} = \frac{d}{b}$ and $\bar{A}_{\max} = \frac{d}{a}$.

In order to obtain the expectation $E[\bar{A}]$ of the unavailability, we consider the conditional expectation knowing the value of MUT

$$\mathbb{E}[\bar{A} \mid \text{MUT} = u] = \frac{d}{u + d}. \tag{30}$$

For case 1 where MUT follows a uniform distribution with density $\frac{1}{\Delta}$, we get by deconditioning

$$\begin{aligned}
 \mathbb{E}[\bar{A}] &= \int_{a-d}^{b-d} \frac{d}{u+d} \frac{du}{\Delta}, \\
 &= \frac{d}{\Delta} [\ln(u+d)]_{a-d}^{b-d} \\
 &= \frac{d}{\Delta} \ln\left(\frac{b}{a}\right)
 \end{aligned}
 \tag{31}$$

Now, introduce an index of uncertainty K defined as $K = \frac{b}{a} - 1$. Note that $K \geq 0$ and that $K = 0$ if MUT is a constant. However, the purpose of K is not to be a general factor. Since $b = a(K + 1)$ and $\Delta = aK$, we can rewrite the last result in the following way:

$$\mathbb{E}[\bar{A}] = \frac{d}{aK} \ln(K + 1)
 \tag{32}$$

In case 2, MUT follows a triangular distribution. Considering the random variable X defined as $X = \text{MUT} + d$, it is clear that X also follows a triangular distribution (because d is a constant), the density of which is given by

$$f_X(x) = \begin{cases} 0 & \text{si } x < a \\ \frac{2}{\rho\Delta^2}(x - a) & \text{si } a \leq x \leq a + \rho\Delta \\ \frac{2}{\Delta^2(1-\rho)}(b - x) & \text{si } a + \rho\Delta \leq x \leq b \\ 0 & \text{si } x > b \end{cases}
 \tag{33}$$

As a function of the random variable X , the steady-state unavailability \bar{A} is equal to $\frac{d}{X}$ and we obtain the expectation $\mathbb{E}[\bar{A}]$ by deconditioning. For $0 < \rho < 1$, we have:

$$\begin{aligned}
 \mathbb{E}[\bar{A}] &= \int_a^{a+\rho\Delta} \frac{d}{x} f_X(x) dx + \int_{a+\rho\Delta}^b \frac{d}{x} f_X(x) dx \\
 &= \frac{2d}{\rho\Delta^2} \int_a^{a+\rho\Delta} \frac{x-a}{x} dx + \frac{2d}{\Delta^2(1-\rho)} \int_{a+\rho\Delta}^b \frac{b-x}{x} dx \\
 &= \frac{2d}{\rho\Delta^2} [x - a \ln(x)]_a^{a+\rho\Delta} + \frac{2d}{\Delta^2(1-\rho)} [b \ln(x) - x]_{a+\rho\Delta}^b \\
 &= \frac{2d}{\Delta^2} \left[\frac{\rho\Delta}{\rho} - \frac{b-a-\rho\Delta}{(1-\rho)} - \frac{a}{\rho} \ln\left(\frac{a+\rho\Delta}{a}\right) + \frac{b}{(1-\rho)} \ln\left(\frac{b}{(a-\rho\Delta)}\right) \right] \\
 &= \frac{2d}{\Delta^2} \left[-\frac{a}{\rho} \ln\left(\frac{a+\rho\Delta}{a}\right) + \frac{b}{(1-\rho)} \ln\left(\frac{b}{(a-\rho\Delta)}\right) \right] \\
 &= \frac{2db}{\Delta^2(1-\rho)} \ln\left(\frac{b}{a+\rho\Delta}\right) - \frac{2da}{\rho\Delta^2} \ln\left(\frac{a+\rho\Delta}{a}\right)
 \end{aligned}
 \tag{34}$$

Using the proposed index of uncertainty K , we can rewrite this result in the following way:

$$\begin{aligned} \mathbb{E}[\bar{A}] &= \frac{2d(K+1)}{a(1-\rho)K^2} (\ln((K+1)) - \ln(1+\rho K)) - \frac{2d}{a\rho K^2} \ln(1+\rho K) \\ &= \frac{2d}{a(1-\rho)K^2} \left((K+1) \ln(K+1) - \frac{\rho(K+1) + (1-\rho)}{\rho} \ln(1+\rho K) \right) \\ &= \frac{2d}{a(1-\rho)K^2} \left((K+1) \ln(K+1) - \frac{(1+\rho K)}{\rho} \ln(1+\rho K) \right) \end{aligned} \quad (35)$$

Following the same line, we get for $\rho = 0$:

$$\begin{aligned} \mathbb{E}[\bar{A}] &= \frac{2d}{\Delta^2} \int_a^b \frac{b-x}{x} dx \\ &= \frac{2d}{aK^2} [(1+K) \ln(1+K) - K] , \end{aligned}$$

while for $\rho = 1$, we obtain

$$\begin{aligned} \mathbb{E}[\bar{A}] &= \frac{2d}{\Delta^2} \int_a^b \frac{x-a}{x} dx \\ &= \frac{2d}{aK^2} [K - \ln(1+K)] . \end{aligned}$$

In Fig. 6, we illustrate the variation of the expectation of $\mathbb{E}[\bar{A}]$ as a function of ρ for $a = 10$, $b = 20$, and $d = 1$ (therefore, $K = 1$). As expected, this expectation is decreasing when ρ is increasing (but not linearly).

Fig. 6 Variation of the expectation of $\mathbb{E}[\bar{A}]$ as a function of ρ when the VA MUT follows a triangular distribution. With $a = 10$, $b = 20$, and $d = 1$

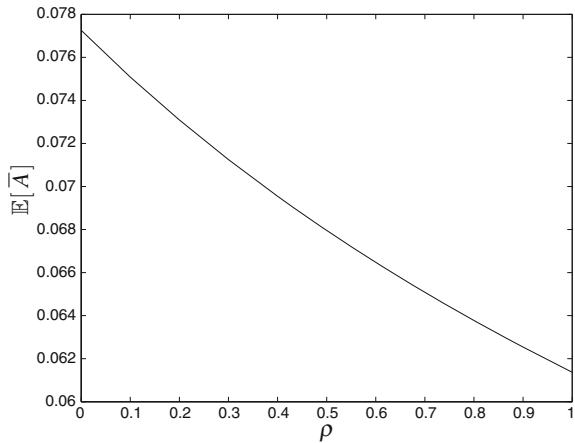
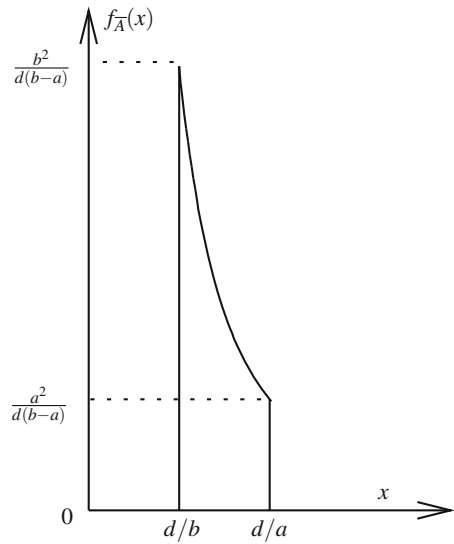


Fig. 7 Density of \bar{A} when MUT follows a uniform distribution



5.1.2 Distributions of the New Random Variables

Let us first consider case 1 where MUT follows a uniform distribution. Then, the density function of \bar{A} can be written as:

$$f_{\bar{A}}(x) = \frac{d}{(b-a)x^2}, \quad x \in \left[\frac{d}{b}, \frac{d}{a} \right] \quad (36)$$

This density function is illustrated Fig. 7.

The cumulative probability function of \bar{A} can be written as:

$$F_{\bar{A}}(x) = \begin{cases} 0 & \text{if } x \leq d/b \\ \frac{b-d/x}{(b-a)} & \text{if } d/b \leq x \leq d/a \\ 1 & \text{if } x \geq d/a \end{cases} \quad (37)$$

We have in particular

$$\mathbb{P}(\bar{A} > x) = \frac{1}{(b-a)} \left[\frac{d}{x} - a \right], \quad x \in \left[\frac{d}{b}, \frac{d}{a} \right] \quad (38)$$

With the aim of normalizing the observations, let us now express the variable x as a function of a parameter $\alpha \in [0, 1]$ in the following way:

$$x(\alpha) = \bar{A}_{\min} + \alpha(\bar{A}_{\max} - \bar{A}_{\min}) = (1 - \alpha)\frac{d}{b} + \alpha\frac{d}{a} \quad (39)$$

or, using the index K

$$x(\alpha) = \frac{d(1 + \alpha K)}{a(1 + K)} \tag{40}$$

We now consider the probability $\mathbb{P}(\bar{A} > x(\alpha))$ that can be written as

$$\begin{aligned} \mathbb{P}(\bar{A} > x(\alpha)) &= \frac{1}{(b-a)} \left[\frac{a(1+K)}{(1+\alpha K)} - a \right], \quad x \in \left[\frac{d}{b}, \frac{d}{a} \right] \\ &= \frac{a}{aK} \left[\frac{(1+K) - (1+\alpha K)}{(1+\alpha K)} \right], \\ &= \frac{(1-\alpha)}{(1+\alpha K)}. \end{aligned} \tag{41}$$

Let us remark that the difference $(1 - \alpha) - \mathbb{P}(\bar{A} > x(\alpha))$ can be written as

$$(1 - \alpha) - \mathbb{P}(\bar{A} > x(\alpha)) = \frac{\alpha(1 - \alpha)K}{(1 + \alpha K)}, \tag{42}$$

and is always positive when $\alpha \in (0, 1)$. This property tells us that $\mathbb{P}(\bar{A} > x(\alpha))$ is always lower than $(1 - \alpha)$ as long as $\alpha \in (0, 1)$.

Figure 8 presents the variation of $\mathbb{P}(\bar{A} \leq x(\alpha))$ as a function of α , $\alpha \in [0, 1]$, for different values of K . The linear segment associated to $K = 0$ corresponds to the limit curve when K tends to zero, i.e., when a tends to b . We observe that $\mathbb{P}(\bar{A} > x(\alpha))$ decreases significantly when the index K increases. A zoom of these variation when $\alpha \in [0.9, 1]$ is shown in Fig. 9. Nevertheless, we have to remember that the interval $[\bar{A}_{\min}, \bar{A}_{\max}]$ increases with index K and that globally, uncertainty remains a penalizing element.

Another point is to look for the value α such that $\mathbb{P}(\bar{A} > x(\alpha)) = \gamma$, when γ takes small values. This enables the determination of α from the equation

Fig. 8 Variation of $\mathbb{P}(\bar{A} \leq x(\alpha))$ as a function of α , $\alpha \in [0, 1]$. Values of K : $\{0, 1, 3, 7\}$ (bottom up). MUT follows a uniform distribution

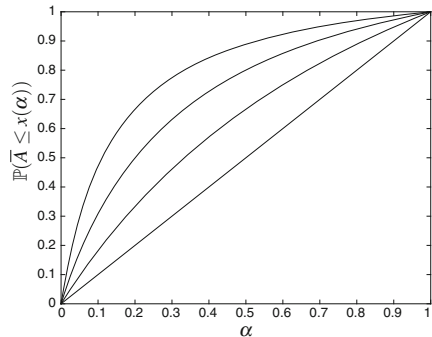
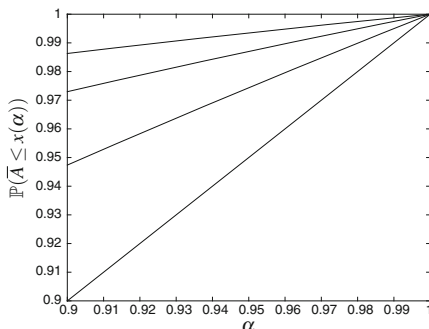


Fig. 9 Variation of $\mathbb{P}(\bar{A} \leq x(\alpha))$ as a function of α , zoom on $\alpha \in [0.9, 1]$. Values of $K : \{0, 1, 3, 7\}$ (bottom up)



$$\frac{(1 - \alpha)}{(1 + \alpha K)} = \gamma . \tag{43}$$

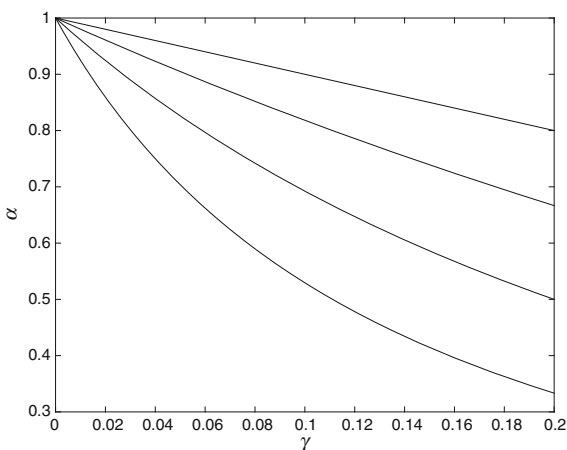
This gives us α as a function of γ :

$$\alpha = \frac{(1 - \gamma)}{(1 + \gamma K)} . \tag{44}$$

We observe that this function is an involution ($f(f(x)) = x$). In particular, it must be symmetric. Figure 10 gives the value of α as a function of γ , $\gamma \in [0, 0.2]$, for different values of K . We observe that the value of α decreases significantly when the index K is increased.

Let us now consider the second case where MUT follows a triangular distribution. Again $\bar{A} = \frac{d}{X}$ but now the density function of X is given by relation (33). Since the

Fig. 10 Variation of α as a function of γ , $\gamma \in [0, 0.2]$. Values of $K : \{0, 1, 3, 7\}$ (top down). MUT follows a uniform distribution



function $\phi(x) = \frac{d}{x}$ is strictly monotone and differentiable, $\bar{A} = \phi(X)$ is a continuous rv with density

$$f_{\bar{A}}(z) = \begin{cases} f_X(\phi^{-1}(z)) \cdot |(\phi^{-1})'(z)| & \text{if } z \in \left[\frac{d}{b}, \frac{d}{a}\right] \\ 0 & \text{else} \end{cases} \tag{45}$$

Here, $\phi^{-1}(z) = \frac{d}{z}$ and $|(\phi^{-1})'(z)| = \frac{d}{z^2}$, producing

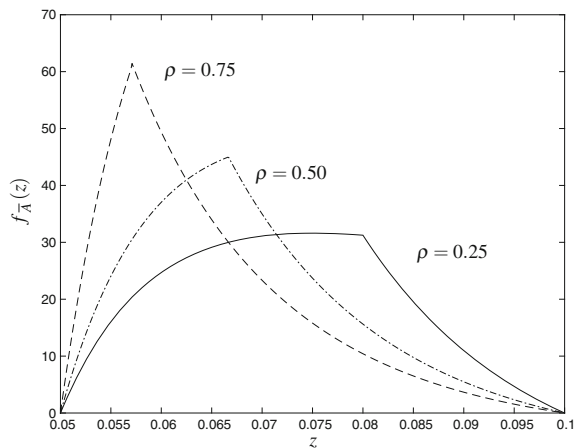
$$f_{\bar{A}}(z) = \begin{cases} 0 & \text{if } z < d/b \\ \frac{2d}{(1-\rho)\Delta^2} \frac{1}{z^2} \left(b - \frac{d}{z}\right) & \text{if } d/b \leq z \leq \frac{d}{a + \rho\Delta} \\ \frac{2d}{\rho\Delta^2} \frac{1}{z^2} \left(\frac{d}{z} - a\right) & \text{if } \frac{d}{a + \rho\Delta} \leq z \leq d/a \\ 0 & \text{if } z > d/a \end{cases} \tag{46}$$

This density function of \bar{A} is illustrated Fig. 11 for three values of ρ (0.25, 0.5 and 0.75).

We obtain the cumulative distribution function by integration

$$F_{\bar{A}}(z) = \begin{cases} 0 & \text{if } z < d/b \\ \frac{1}{(1-\rho)\Delta^2} \left(b - \frac{d}{z}\right)^2 & \text{if } d/b \leq z \leq \frac{d}{a + \rho\Delta} \\ 1 - \frac{1}{\rho\Delta^2} \left(\frac{d}{z} - a\right)^2 & \text{if } \frac{d}{a + \rho\Delta} \leq z \leq d/a \\ 1 & \text{if } z > d/a \end{cases} \tag{47}$$

Fig. 11 Density function of \bar{A} when the VA MUT follows a triangular distribution for three values of ρ . With $a = 10$, $b = 20$ and $d = 1$



We note that $F_{\bar{A}}(z)$ takes the value $(1 - \rho)$ for $z = \frac{d}{a + \rho\Delta}$. Since $\text{MUT}_{\text{mod}} + d = (a + \rho\Delta)$, this result agrees with the fact that the probability $\mathbb{P}(\text{MUT} \geq \text{MUT}_{\text{mod}})$ is equal to $(1 - \rho)$. Note that condition $(\text{MUT} \geq \text{MUT}_{\text{mod}})$ implies condition $(\bar{A} \leq \frac{d}{a + \rho\Delta})$.

Let us consider the probability $\mathbb{P}(\bar{A} > x)$, $\bar{A} \in \left[\frac{d}{b}, \frac{d}{a}\right]$, when x is expressed as a function of the normalizing coefficient α , using relation (40). With this objective, let us first denote α^* the value of α such that $x(\alpha^*) = \frac{d}{a + \rho\Delta}$, i.e., the value of α that corresponds to the unavailability obtained when $\text{MUT} = \text{MUT}_{\text{mod}}$. This value is equal to

$$\alpha^* = \frac{(1 - \rho)}{(1 + \rho K)}. \quad (48)$$

Note that, because K is nonnegative, α^* is always lower than $(1 - \rho)$.

When α is between α^* and 1, we have

$$\begin{aligned} \mathbb{P}(\bar{A} > x(\alpha)) &= 1 - F_{\bar{A}}(x(\alpha)) = \frac{1}{\rho\Delta^2} \left(\frac{a(1 + K)}{(1 + \alpha K)} - a \right)^2 \\ &= \frac{a^2}{\rho\Delta^2} \left(\frac{(1 + K) - (1 + \alpha K)}{(1 + \alpha K)} \right)^2 = \frac{a^2}{a^2\rho K^2} \left(\frac{(1 - \alpha)K}{(1 + \alpha K)} \right)^2 \\ &= \frac{1}{\rho} \left(\frac{(1 - \alpha)}{(1 + \alpha K)} \right)^2. \end{aligned} \quad (49)$$

Note that $\mathbb{P}(\bar{A} > x(\alpha^*)) = \rho$ and does not depend on the index K .

We may also be interested in small values of x , when α lies between 0 and α^* . In that case, we have

$$\begin{aligned} \mathbb{P}(\bar{A} \leq x(\alpha)) &= \frac{1}{(1 - \rho)\Delta^2} \left(b - \frac{d}{z} \right)^2 \\ &= \frac{1}{(1 - \rho)\Delta^2} \left(\frac{a(1 + K)(1 + \alpha K) - a(1 + K)}{(1 + \alpha K)} \right)^2 \\ &= \frac{a^2}{a^2(1 - \rho)K^2} \left(\frac{\alpha K(1 + K)}{(1 + \alpha K)} \right)^2 \\ &= \frac{1}{(1 - \rho)} \left(\frac{\alpha(1 + K)}{(1 + \alpha K)} \right)^2. \end{aligned} \quad (50)$$

It would be possible to check that $\mathbb{P}(\bar{A} \leq x(\alpha^*)) = (1 - \rho)$.

Figure 12 shows the variation of the probability $\mathbb{P}(\bar{A} \leq x(\alpha))$ as a function of α when $\rho = 0.5$, $\alpha \in [0.5, 1]$, for several values of K . Again, the curve labeled

Fig. 12 Variation of $\mathbb{P}(\bar{A} \leq x(\alpha))$ as a function of α , $\alpha \in [0.5, 1]$. $\rho = 0.5$. Values of $K : \{0, 1, 3, 7\}$ (*bottom up*). MUT follows a triangular distribution

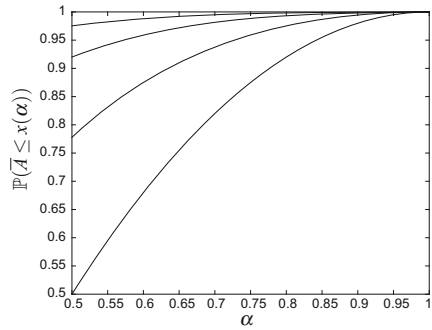
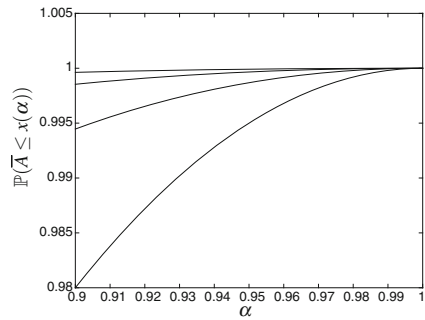


Fig. 13 Variation of $\mathbb{P}(\bar{A} \leq x(\alpha))$ as a function of α with a zoom on $\alpha \in [0.9, 1]$. $\rho = 0.5$. Values of $K : \{0, 1, 3, 7\}$ (*bottom up*). MUT follows a triangular distribution



$K = 0$ corresponds to the limit curve when K tends to zero, i.e., when MUT_{max} tends to MUT_{min} . We may observe that the probability $\mathbb{P}(\bar{A} > x(\alpha))$ decreases significantly when the index K increases. A zoom of this variation is presented on Fig. 13 with the reduced interval $\alpha \in [0.9, 1]$. Comparing this latter figure with Fig. 9, we may realize, for identical values of index K , the advantage of the triangular distribution over the uniform one.

It is possible to find the value of α such that $\mathbb{P}(\bar{A} > x(\alpha)) = \gamma$, for small values of γ by looking for the solution of the following equation:

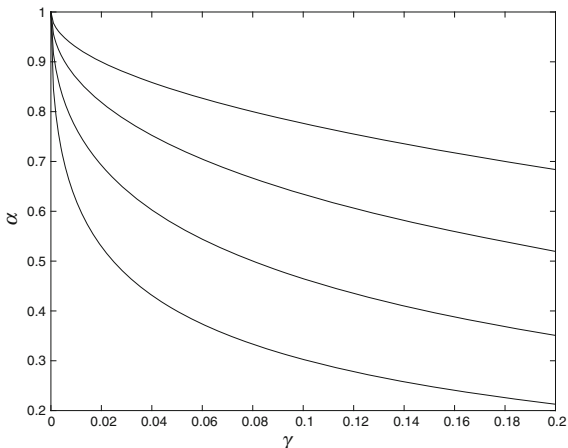
$$\frac{1}{\rho} \left(\frac{(1 - \alpha)}{1 + K\alpha} \right)^2 = \gamma, \tag{51}$$

from which we get α as a function of γ :

$$\alpha = \frac{(1 - \sqrt{\rho\gamma})}{(1 + K\sqrt{\rho\gamma})}. \tag{52}$$

Figure 14 shows the value of α as a function of γ , $\gamma \in [0, 0.2]$, for several values of K . For example, if $K = 1$, $\rho = 0.5$ and $\gamma = 0.1$, then $\alpha = 0.63$. We can observe that the value of α significantly decreases when the index K increases. These results have to be compared with those of Fig. 10 relative to the case of the uniform distribution.

Fig. 14 Variation of α as a function of γ , $\gamma \in [0, 0.2]$. Values of $K : \{0, 1, 3, 7\}$ (top down). MUT follows a triangular distribution



We can also find the value of α such that $\mathbb{P}(\bar{A} \leq x(\alpha)) = \delta$, for small values of δ . It is possible to show that α satisfies the relation

$$\alpha = \frac{\sqrt{(1 - \rho)\delta}}{1 + K(1 - \sqrt{(1 - \rho)\delta})} . \tag{53}$$

In order to synthesize the main results of this study of the steady-state unavailability in the presence of uncertainty on the MUT's value, let us define the function $R_K(x)$ as follows:

$$R_K(x) = \frac{1 - x}{1 + Kx} . \tag{54}$$

We note that this function is an involution: $R_K(R_K(x)) = x$.

For case 1 where the random variable MUT is assumed to follow a uniform distribution,

$$\mathbb{P}(\bar{A} > x(\alpha)) = R_K(\alpha) , \tag{55}$$

and, for a given value γ , the value of α_γ satisfying the equality $\mathbb{P}(\bar{A} > x(\alpha)) = \gamma$ can be written as

$$\alpha_\gamma = R_K(\gamma) . \tag{56}$$

Considering now the case 2 where the random variable MUT is assumed to follow a triangular distribution,

$$\alpha^* = R_K(\rho) . \tag{57}$$

For $\alpha^* \leq \alpha \leq 1$, we have

$$\mathbb{P}(\bar{A} > x(\alpha)) = \frac{1}{\rho} (R_K(\alpha))^2, \quad (58)$$

and

$$\alpha_\gamma = R_K(\sqrt{\rho\gamma}). \quad (59)$$

For $0 \leq \alpha \leq \alpha^*$, we have

$$\mathbb{P}(\bar{A} \leq x(\alpha)) = \frac{1}{(1-\rho)} (1 - R_K(\alpha))^2, \quad (60)$$

and, for a given value δ , the value α_δ satisfying the equality $\mathbb{P}(\bar{A} \leq x(\alpha)) = \delta$ can be written as

$$\alpha_\delta = R_K(1 - \sqrt{(1-\rho)\delta}). \quad (61)$$

Since the function R_K is an involution, we immediately obtain

$$\mathbb{P}(\bar{A} > x(\alpha^*)) = \frac{1}{\rho} (R_K(\alpha^*))^2 = \frac{1}{\rho} (R_K(R_K(\rho)))^2 = \rho, \quad (62)$$

and

$$\mathbb{P}(\bar{A} \leq x(\alpha^*)) = \frac{(1 - R_K(\alpha^*))^2}{(1-\rho)} = \frac{1}{(1-\rho)} (1 - R_K(R_K(\rho)))^2 = (1-\rho). \quad (63)$$

Looking for the situation where there is a unique random variable MDT instead of MUT would lead to the same kind of results with an equivalent difficulty. The situation where both MUT and MDT are considered as random variables, respectively, on intervals $[a_0, b_0]$ and $[c_0, d_0]$ is more difficult to solve and, in particular, we have to consider the orders of the values a_0, b_0, c_0, d_0 in order to obtain the density function of \bar{A} .

6 A Final Remark

The period observed in this chapter started at the epoch where the rare computers were like dinosaurs: big, with a small memory and relatively slow. This was the time where research was focusing on synthetic models and evaluation of steady-state measures. The technological progress of computers achieved over the past forty years has allowed researchers to tackle more sophisticated and larger models and to look for transient measures. But going deeper into the details, we realize that we still have more questions to answer than before!

References

1. Aggarwal Krishan K, Gupta Jagdish S, Misra Krishna B (1973) A new method for system reliability evaluation. *Microelectron Reliab* 12(5):435–440
2. Baskett F, Chandy KM, Muntz RR, Palacios FG (1975) Open, closed, and mixed networks of queues with different classes of customers. *J ACM (JACM)* 22(2):248–260
3. Beaudry MD (1978) Performance-related reliability measures for computing systems. *IEEE Trans Comput* 100(6):540–547
4. Benetto Enrico, Dujet Christiane, Rousseaux Patrick (2008) Integrating fuzzy multicriteria analysis and uncertainty evaluation in life cycle assessment. *Environ Model Softw* 23(12):1461–1467
5. Blake JT, Reibman AL, Trivedi KS (1988) Sensitivity analysis of reliability and performability measures for multiprocessor systems. In: *ACM SIGMETRICS performance evaluation review*, vol 16. ACM, pp 177–186
6. Buzen Jeffrey P (1973) Computational algorithms for closed queueing networks with exponential servers. *Commun ACM* 16(9):527–531
7. Chen J, Brissette FP, Chaumont D, Braun M (2013) Performance and uncertainty evaluation of empirical downscaling methods in quantifying the climate change impacts on hydrology over two north american river basins. *J Hydrol* 479:200–214
8. Ciardo Gianfranco, Marie Raymond A, Sericola Bruno, Trivedi Kishor S (1990) Performability analysis using semi-markov reward processes. *IEEE Trans Comput* 39(10):1251–1264
9. Ciciani Bruno, Grassi Vincenzo (1987) Performability evaluation of fault-tolerant satellite systems. *IEEE Trans Commun* 35(4):403–409
10. Cohen JW (1969) *The single server queue*. North-Holland, New York, Amsterdam
11. Colbourn CJ (1999) Reliability issues in telecommunications network planning. In: *Telecommunications network planning*, pp 135–146. Springer
12. de Souza e Silva E, Gail HR (1998) An algorithm to calculate transient distributions of cumulative rate and impulse based reward. *Stoch Models* 14(3):509–536
13. de Souza e Silva E, Gail HR (1989) Calculating availability and performability measures of repairable computer systems using randomization. *J ACM (JACM)* 36(1):171–193
14. de Souza e Silva E, Gail HR (1992) Performability analysis of computer systems: from model specification to solution. *Perform Eval* 14(3):157–196
15. de Souza e Silva E, Gail HR (2000) Transient solutions for markov chains. In: *Computational probability*, pp 43–79. Springer
16. de Souza e Silva E, Gail HR, Campos RV (1995) Calculating transient distributions of cumulative reward. In: *ACM-SIGMETRICS*, pp 231–240
17. de Souza e Silva E, Gail HR et al (1986) Calculating cumulative operational time distributions of repairable computer systems. *IEEE Trans Comput* 100(4):322–332
18. Do Van P, Barros A, Bérenguer C (2008) Reliability importance analysis of markovian systems at steady state using perturbation analysis. *Reliab Eng Syst Safety* 93(11):1605–1615
19. Do Van P, Barros A, Bérenguer C (2010) From differential to difference importance measures for markov reliability models. *Eur J Oper Res* 204(3):513–521
20. Donatiello Lorenzo, Grassi Vincenzo (1991) On evaluating the cumulative performance distribution of fault-tolerant computer systems. *IEEE Trans Comput* 40(11):1301–1307
21. Donatiello L, Iyer BR (1987) Analysis of a composite performance reliability measure for fault-tolerant systems. *J ACM (JACM)* 34(1):179–199
22. Fricks R, Telek M, Puliafito A, Trivedi K (1998) Markov renewal theory applied to performability evaluation. In: *State-of-the-art in Performance modeling and simulation. modeling and simulation of advanced computer systems: applications and systems*. Gordon & Breach, Newark, NJ, pp 193–236
23. Furchtgott DG, Meyer JF (1984) A performability solution method for degradable nonrepairable systems. *IEEE Trans Comput* 100(6):550–554

24. Furchtgott DG, Meyer JF (1978) Performability evaluation of fault-tolerant multiprocessors. In: Digest 1978 government micro-circuit applications conference, pp 362–365. NTIS Springfield, VA
25. Garelli Marco, Ferrero Andrea (2012) A unified theory for s-parameter uncertainty evaluation. *IEEE Trans Microwave Theory Tech* 60(12):3844–3855
26. German Reinhard (2000) Markov regenerative stochastic petri nets with general execution policies: supplementary variable analysis and a prototype tool. *Perform Eval* 39(1):165–188
27. German Reinhard (2001) Iterative analysis of markov regenerative models. *Perform Eval* 44(1):51–72
28. German R, Telek M (1999) Formal relation of markov renewal theory and supplementary variables in the analysis of stochastic Petri nets. In: Proceedings of the 8th international workshop on Petri nets and performance models, IEEE, pp 64–73
29. Goyal Ambuj, Tantawi Asser N (1987) Evaluation of performability for degradable computer systems. *IEEE Trans Comput* 100(6):738–744
30. Goyal Ambuj, Tantawi Asser N (1988) A measure of guaranteed availability and its numerical evaluation. *IEEE Trans Comput* 37(1):25–32
31. Goyal A, Tantawi AN, Trivedi KS (1985) A measure of guaranteed availability. In: IBM Thomas J, Watson Research Center
32. Grassi Vincenzo, Donatiello Lorenzo (1992) Sensitivity analysis of performability. *Perform Eval* 14(3):227–237
33. Haverkort BR, Meeuwissen AMH (1995) Sensitivity and uncertainty analysis of markov-reward models. *IEEE Trans Reliab* 44(1):147–154
34. Haverkort BR, Niemegeers IG (1996) Performability modelling tools and techniques. *Performance Eval* 25(1):17–40
35. Heidelberger P, Goyal A (1987) Sensitivity analysis of continuous time markov chains using uniformization. In: Proceedings of the second international workshop on applied mathematics and performance/reliability models of computer/communication systems
36. Hirel C, Sahner R, Zang X, Trivedi K (2000) Reliability and performability modeling using sharpe 2000. In: Computer performance evaluation. Modelling techniques and tools, pp 345–349. Springer
37. Iyer Balakrishna R, Donatiello Lorenzo, Heidelberger Philip (1986) Analysis of performability for stochastic models of fault-tolerant systems. *IEEE Trans Comput* 100(10):902–907
38. Jackson PS, Hockenbury RW, Yeater ML (1982) Uncertainty analysis of system reliability and availability assessment. *Nuclear Eng Des* 68(1):5–29
39. Kulkarni VG, Nicola VF, Smith RM, Trivedi KS (1986) Numerical evaluation of performability and job completion time in repairable fault-tolerant systems. In: 16th international symposium on fault-tolerant computing, IEEE
40. Lanus M, Yin L, Trivedi KS (2003) Hierarchical composition and aggregation of state-based availability and performability models. *IEEE Trans Reliab* 52(1):44–52
41. Lüthi Johannes, Haring Günter (1998) Mean value analysis for queueing network models with intervals as input parameters. *Performance Eval* 32(3):185–215
42. Lüthi Johannes, Majumdar Shikharesh, Kotsis Gabriele, Haring Günter (1997) Performance bounds for distributed systems with workload variabilities and uncertainties. *Parallel Comput* 22(13):1789–1806
43. Majumdar S, Ramadoss R (1995) Interval-based performance analysis of computing systems. In: Proceedings of the third international workshop on modeling, analysis, and simulation of computer and telecommunication systems, 1995. MASCOTS'95, pp 345–351. IEEE
44. Marie RA, Reibman AL, Trivedi KS (1987) Transient analysis of acyclic markov chains. *Performance Eval* 7(3):175–194
45. Meyer John F (1980) On evaluating the performability of degradable computing systems. *IEEE Trans Comput* 100(8):720–731
46. Meyer John F (1982) Closed-form solutions of performability. *IEEE Trans Comput* 100(7):648–657

47. Meyer John F (1992) Performability: a retrospective and some pointers to the future. *Performance Eval* 14(3):139–156
48. Meyer John F, Furchtgott David G, Liang TWu (1980) Performability evaluation of the sift computer. *IEEE Trans Comput* 100(6):501–509
49. Misra Krishna B (1970) An algorithm for the reliability evaluation of redundant networks. *IEEE Trans Reliab* 19(4):146–151
50. Moore EF, Shannon CE (1956) Reliable circuits using less reliable relays, part i. *J Franklin Inst* 262(3):191–208
51. Moskowitz Fred (1958) The analysis of redundancy networks. In: *Transactions of the American institute of electrical engineers, part i: communication and electronics* 77(5):627–632
52. Nabli Hédi, Sericola Bruno (1996) Performability analysis: a new algorithm. *IEEE Trans Comput* 45(4):491–494
53. Pattipati KR, Li Y, Blom HAP (1993) A unified framework for the performability evaluation of fault-tolerant computer systems. *IEEE Trans Comput* 42(3):312–326
54. Reibman Andrew, Smith Roger, Trivedi Kishor (1989) Markov and markov reward model transient analysis: an overview of numerical approaches. *Eur J Oper Res* 40(2):257–267
55. Reibman Andrew, Trivedi Kishor (1988) Numerical transient analysis of markov models. *Comput Oper Res* 15(1):19–36
56. Reibman Andrew, Trivedi Kishor (1989) Transient analysis of cumulative measures of markov model behavior. *Stoch Models* 5(4):683–710
57. Rubino Gerardo, Sericola Bruno (1995) Interval availability analysis using denumerable markov processes: application to multiprocessor subject to breakdowns and repair. *IEEE Trans Comput* 44(2):286–291
58. Smith RM, Trivedi KS, Ramesh AV (1988) Performability analysis: measures, an algorithm, and a case study. *IEEE Trans Comput* 37(4):406–417
59. Suné V, Carrasco JA, Nabli H, Sericola B (2010) Comment on performability analysis: a new algorithm. *IEEE Trans Comput* 59(1):137–138
60. Trivedi KS, Malhotra M (1993) Reliability and performability techniques and tools: a survey. In: *Messung, modellierung und bewertung von rechen-und kommunikationssystemen*, pp 27–48. Springer
61. Van Slyke R, Frank Howard (1971) Network reliability analysis: part i. *Networks* 1(3):279–290
62. Wübbeler Gerd, Krystek Michael, Elster Clemens (2008) Evaluation of measurement uncertainty and its numerical calculation by a monte carlo method. *Meas Sci Technol* 19(8):084009
63. Yin L, Smith MAJ, Trivedi KS (2001) Uncertainty analysis in reliability modeling. In: *Proceedings of the reliability and maintainability symposium, 2001, Annual*, pp 229–234. IEEE

Sojourn Times in Dependability Modeling

Gerardo Rubino and Bruno Sericola

Abstract We consider Markovian models of computing or communication systems, subject to failures and, possibly, repairs. The dependability properties of such systems lead to metrics that can all be described in terms of the time that the Markov chain spends in subsets of its state space. Some examples of such metrics are MTTF and MTTR, reliability or availability at a point in time, the mean or the distribution of the interval availability in a fixed time interval, and more generally different performability versions of these measures. This chapter reviews this point of view and its consequences, and discusses some new results related to it.

1 Introduction

In this chapter, we are interested in Markovian models of systems subject to failures and, possibly, repairs. A typical framework is that of a system made of several independently living components or units, each of them belonging to a given class. When the system starts operating, each component has a life-time distributed according to the exponential distribution whose parameter (the component failure rate) is the same for all components in the same class. At the beginning, the system starts with a given number of operational components in each of the classes. Let K be the number of classes, indexed from 1 to K , and N_k the number of class k units in the system. If n_k is the number of class k units that are operational at any point in time, $0 \leq n_k \leq N_k$, a basic case is then when the vector (n_1, \dots, n_K) suffices to obtain a Markovian evolution for the model. For instance, assume that the system is non repairable, and that the only transitions are the individual components' failures. Assume, for instance, that the system works when at least m_k class

G. Rubino (✉) · B. Sericola
Inria Rennes - Bretagne Atlantique, Campus de Beaulieu, 35042 Rennes Cedex, France
e-mail: gerardo.rubino@inria.fr

B. Sericola
e-mail: bruno.sericola@inria.fr

k units are operational, for each k (we say that, from the dependability viewpoint, it works as a series of m_k -out-of- N_k modules). The initial state for the Markovian model $X = \{X_t, t \geq 0\}$ is $X_0 = (N_1, \dots, N_K)$. For this simple structure, we could need to evaluate the system's reliability at time t , $R(t)$, which is here the probability that the system is alive at time t , or the mean system's life-time, typically called Mean Time To Failure, MTTF. For this purpose, we consider that X lives in the state space $S = \{m_1, m_1 + 1, \dots, N_1\} \times \dots \times \{m_K, m_K + 1, \dots, N_K\} \cup \{a\}$, where state a is absorbing and the remaining $(N_1 - m_1 + 1) \dots (N_K - m_K + 1)$ states are transient. If we denote $B = S \setminus \{a\}$, and if T is the absorption time of the chain, we have that T is the sojourn time of X in B , that there is only one such sojourn and that $R(t) = \mathbb{P}\{T > t\}$ and $\text{MTTF} = \mathbb{E}\{T\}$.

With the same state representation as before, and the same system structure (the series of m_k -out-of- N_k modules), we can handle the case where any failed unit is immediately repaired with some rate μ_k when it belongs to class k , the repair times being also independent of anything else in the model. In that case, process X live in $S = \{0, \dots, N_1\} \times \dots \times \{0, \dots, N_K\}$. The Markov chain X has a single recurrent class, and the same B defined above, $B = \{(n_1, \dots, n_K) \in S \mid n_1 \geq m_1, \dots, n_K \geq m_K\}$, together with its complement B^c define a partition of S . The model will spend some random time working, during what we will call its first *operational period*. It is referred to as the first sojourn time of X in B , whose duration will be denoted by $S_{B,1}$. Then, some failure will put it in B^c , that is, in a failed system situation. At that point in time, a sojourn in B^c starts, that from the system point of view, can be seen as a system's repair. Its length in time will be denoted by $S_{B^c,1}$. At some later point in time, the system will come back to subset B and a new operational period will start. This alternate sequence of operational periods and repair or nonoperational ones will thus continue forever. In this setting, it can be of interest to evaluate not only the previously defined metrics (the reliability at t defined by $R(t) = \mathbb{P}\{S_{B,1} > t\}$ and the $\text{MTTF} = \mathbb{E}\{S_{B,1}\}$) but also other ones, such as the mean availability on the interval $[0, t]$, defined as the mean fraction of that interval spent by X in B . Asymptotically, we can consider the widely used asymptotic availability, which can be seen as the limit of the previous fraction when $t \rightarrow \infty$.

As noted in the abstract, most dependability metrics can be associated with the time the system's model spends in subsets of its state space. The examples discussed before illustrate the point, but they can get much more complex. For instance, assume that in the repairable case described above, there is a single repair facility that can handle one component at a time, connected to a buffering area where failed units wait (say, in FIFO order) until the repair server is available. Then, we need richer states to be able to build a Markovian evolution on it, we need to know how many components of each class are working, but also some information about what happens at the repair facility. In all cases, the same connection with sojourn times will hold.

Generally speaking, we often have a stochastic process representing the system, living in some state space S , and associated with each state x we have a function Φ saying if the system is working or not when its state is x . The typical convention is

to use $\Phi(x) = 1$ if the system works when its state is x , and 0 if it is failed. This function Φ is called the system's *structure function*. It naturally defines the set of operational states $B = \Phi^{-1}(1)$ and its complement, where the system doesn't work, $B^c = \Phi^{-1}(0)$. Then, the main dependability metrics are defined as in the initial examples: $\text{MTTF} = \mathbb{E}\{S_{B,1}\}$, $R(t) = \mathbb{P}\{S_{B,1} > t\}$, Mean Time To Repair, $\text{MTTR} = \mathbb{E}\{S_{B^c,1}\}$, etc., and they correspond to different aspects of the sojourns of X in B and/or in B^c .

For a global background on these topics, an appropriate reference is the blue-while-yellow book [20]. All the basic mathematical objects used here are introduced and carefully explained there. Another directly relevant one is the introduction of the authors' recent book [[17], Sect. 1.3]. The reader can also find in Kishor Trivedi's web pages many general presentations of the topics discussed in this chapter. As an example, see [9] for an introduction to Markov models in dependability and extensions to performability, or [8] for generalities about dependability analysis.

The analysis of sojourn times in Markov chains, both in discrete and continuous time, was started in dependability in [10]. In that paper, the distribution of the n th sojourn time of the chain in a subset of its state space, in the irreducible and finite case, is derived, and its asymptotic behavior is analyzed. The connections with state lumping and the so-called "pseudo-aggregation" in [12, 16] are also discussed. In [11] the corresponding absorbing case is analyzed. At the end of the chapter, supplementary bibliographical notes are provided.

It appears that some of the results published in this area, for instance the basic distributions, were known in a specific biological field called "ion channel analysis," as reported in [6]. As stated in [17], the analysis of sojourn times is also relevant in queueing theory (think of the busy period of a single queue, for instance).

This chapter reviews part of the basic material concerning the times spent by a Markovian process in proper subsets of its state space, and adds some new elements and guidelines for obtaining more results. Since we focus here on dependability applications, all our developments are in continuous time, but they have counterparts in discrete time, not discussed here (see the references). We had to make some choices because of the amount of material available. For this reason, until Sect. 5, we limited the discussion to irreducible models. Section 2 reviews the basic facts when studying these objects, namely, the distribution of the sojourn time of a Markov process in a subset of its state space. The asymptotic behavior of this distribution is also discussed. Section 3 presents the joint distribution of sojourn times. In Sect. 4, we focus on a specific function of them, the sum of the n first sojourn times of the chain in a given subset of states, its distribution, the computation of its moments, etc. Section 5 illustrates how to deal with absorbing models, and at the same time, with Markov models whose states are weighted by rewards (Markov reward models). In the last Sect. 6, bibliographical notes are provided, together with some supplementary comments and discussions.

2 Successive Sojourn Times Distribution

We consider a homogeneous irreducible continuous-time Markov chain $X = \{X_t, t \geq 0\}$ with finite state space denoted by S . Its initial probability distribution is given by the row vector α and its transition rate matrix by A . The stationary distribution of X , denoted by π , is the row vector satisfying $\pi A = 0$ and $\pi \mathbb{1} = 1$ where $\mathbb{1}$ is the column vector with all its entries equal to 1, its dimension being given by the context of its use. For every $i \in S$, we denote by λ_i the output rate of state i , that is $\lambda_i = -A_{i,i}$. Let λ be a positive real number such that $\lambda \geq \max\{\lambda_i, i \in S\}$ and let $\{N_t, t \geq 0\}$ be a Poisson process with rate λ . We then define matrix P by $P = I + A/\lambda$, where I is the identity matrix which dimension is also determined by the context of its use. We introduce the discrete-time Markov chain $Z = \{Z_n, n \geq 0\}$ on the state space S , with transition probability matrix P and with initial probability distribution α . Assuming that the processes $\{N_t\}$ and Z are independent, the stochastic processes X and $\{Z_{N_t}, t \geq 0\}$ are equivalent, i.e., they have the same finite-dimensional distributions. This well-known construction is called the uniformization technique. The Markov chain Z is called the discrete-time Markov chain associated with the uniformized Markov chain of X with respect to the uniformization rate λ .

Let B be a proper subset of S , i.e., $B \neq \emptyset$ and $B \neq S$. We denote by B^c the subset $S \setminus B$. Subset B contains the operational states and subset B^c contains the non operational ones. The subsets B, B^c form a partition of the state space S . Ordering the states such that those in B appear first, then those in B^c , the partition induces the following decomposition of matrices A and P and vectors α and π :

$$A = \begin{pmatrix} A_B & A_{BB^c} \\ A_{B^c B} & A_{B^c} \end{pmatrix}, P = \begin{pmatrix} P_B & P_{BB^c} \\ P_{B^c B} & P_{B^c} \end{pmatrix}, \alpha = (\alpha_B \ \alpha_{B^c}) \text{ and } \pi = (\pi_B \ \pi_{B^c}).$$

Lemma 1 *The matrices $I - P_B$ and $I - P_{B^c}$ are invertible.*

Proof Consider the auxiliary discrete-time Markov chain Z' on the state space $B \cup a$, where a is an absorbing state, with transition probability matrix P' given by

$$P' = \begin{pmatrix} P_B & u \\ 0 & 1 \end{pmatrix},$$

where u is the column vector defined by $u = \mathbb{1} - P_B \mathbb{1}$. The Markov chain X being irreducible, the states of B are transient for Markov chain Z' . A well-known result about Markov chains says that if state j is transient, then for all state i , we have $(P^k)_{i,j} \rightarrow 0$ as $k \rightarrow \infty$, if P is the transition probability matrix of the chain, see for instance [18]. Here, this translates into the fact that

$$\lim_{k \rightarrow \infty} (P_B)^k = 0.$$

From the immediate identity

$$(I - P_B) \sum_{k=0}^K (P_B)^k = \left(\sum_{k=0}^K (P_B)^k \right) (I - P_B) = I - (P_B)^{K+1},$$

we obtain, taking the limit as $K \rightarrow \infty$, that the series $\sum_{k \geq 0} (P_B)^k$ converges, and calling M its limit, that $(I - P_B)M = M(I - P_B) = I$. This means that $I - P_B$ is invertible and that $(I - P_B)^{-1} = \sum_{k \geq 0} (P_B)^k$. Replacing B by B^c and using again the same argument, we obtain that $I - P_{B^c}$ is invertible as well. \square

Consider the successive instants at which the Markov chain X enters subsets B and B^c . We define $T_{B,1} = \inf\{t \geq 0 \mid X_t \in B\}$, $T_{B^c,1} = \inf\{t \geq 0 \mid X_t \in B^c\}$ and, for every $n \geq 2$,

$$\begin{aligned} T_{B,n} &= \inf\{t > T_{B,n-1} \mid X_{t^-} \in B^c, X_t \in B\}, \\ T_{B^c,n} &= \inf\{t > T_{B^c,n-1} \mid X_{t^-} \in B, X_t \in B^c\}. \end{aligned}$$

Note that if $X_0 \in B$ (resp. $X_0 \in B^c$) then we have $T_{B,1} = 0$ (resp. $T_{B^c,1} = 0$). The n th sojourn time of X in B is denoted by $S_{B,n}$ and we have, for every $n \geq 1$,

$$S_{B,n} = \begin{cases} T_{B^c,n} - T_{B,n} & \text{if } X_0 \in B, \\ T_{B^c,n+1} - T_{B,n} & \text{if } X_0 \in B^c. \end{cases}$$

Let $V_{B,n}$ (resp. $V_{B^c,n}$), for $n \geq 1$, be the random variable representing the state of B (resp. B^c) in which the n th sojourn of X in B (resp. B^c) starts. With the usual convention saying that the paths of X are right-continuous we have, for $n \geq 1$, $V_{B,n} = X_{T_{B,n}}$ and $V_{B^c,n} = X_{T_{B^c,n}}$. We introduce the matrices R and H defined by

$$R = (I - P_B)^{-1} P_{BB^c} \text{ and } H = (I - P_{B^c})^{-1} P_{B^cB}.$$

Note that both matrices R and H are composed of probabilities, and satisfy $R\mathbb{1}=\mathbb{1}$ and $H\mathbb{1}=\mathbb{1}$.

Theorem 1 *The process $V_B = \{V_{B,n}, n \geq 1\}$ is a homogeneous discrete-time Markov chain with state space B . Its initial probability distribution, denoted by $v^{(1)}$, and its transition probability matrix, denoted by G , are given by*

$$v^{(1)} = \alpha_B + \alpha_{B^c}H \text{ and } G = RH.$$

Proof The sequence of instants $T_{B,n}$ being an increasing sequence of stopping times, the process $V_B = \{V_{B,n}, n \geq 1\}$ is a homogeneous discrete-time Markov chain with state space B . For every $i, j \in B$, we have

$$\mathbb{P}\{V_{B,1} = j \mid X_0 = i\} = \mathbb{P}\{X_0 = j \mid X_0 = i\} = \mathbf{1}_{\{i=j\}}.$$

For every $i \in B^c$ and $j \in B$, we have, using the Markov property,

$$\begin{aligned}
 \mathbb{P}\{V_{B,1} = j \mid X_0 = i\} &= \mathbb{P}\{X_{T_{B,1}} = j \mid X_0 = i\} \\
 &= \sum_{k \in S} P_{i,k} \mathbb{P}\{X_{T_{B,1}} = j \mid Z_1 = k, X_0 = i\} \\
 &= \sum_{k \in B} P_{i,k} 1_{\{k=j\}} + \sum_{k \in B^c} P_{i,k} \mathbb{P}\{X_{T_{B,1}} = j \mid X_0 = k\} \\
 &= P_{i,j} + \sum_{k \in B^c} P_{i,k} \mathbb{P}\{V_{B,1} = j \mid X_0 = k\}.
 \end{aligned}$$

Denoting by M the matrix defined, for $i \in B^c$ and $j \in B$, by $M_{i,j} = \mathbb{P}\{V_{B,1} = j \mid X_0 = i\}$, this last equality leads to $M = P_{B^c B} + P_{B^c} M$, that is, using Lemma 1, $M = (I - P_{B^c})^{-1} P_{B^c B}$, i.e., $M = H$. This leads, for every $j \in B$, to

$$v_j^{(1)} = \mathbb{P}\{V_{B,1} = j\} = \sum_{i \in B} \alpha_i 1_{\{i=j\}} + \sum_{i \in B^c} \alpha_i H_{i,j} = \alpha_j + (\alpha_{B^c} H)_j,$$

that is $v^{(1)} = \alpha_B + \alpha_{B^c} H$.

Using the same arguments and denoting by K the matrix defined, for $i \in B$ and $j \in B^c$, by $K_{i,j} = \mathbb{P}\{V_{B^c,1} = j \mid X_0 = i\}$, we obtain symmetrically $K = R$. For every $i, j \in B$, we get, using the Markov property,

$$\begin{aligned}
 G_{i,j} &= \mathbb{P}\{V_{B,2} = j \mid V_{B,1} = i\} = \mathbb{P}\{V_{B,2} = j \mid X_0 = i\} \\
 &= \sum_{k \in B^c} \mathbb{P}\{V_{B^c,1} = k \mid X_0 = i\} \mathbb{P}\{V_{B,2} = j \mid V_{B^c,1} = k, X_0 = i\} \\
 &= \sum_{k \in B^c} \mathbb{P}\{V_{B^c,1} = k \mid X_0 = i\} \mathbb{P}\{V_{B,1} = j \mid X_0 = k\} \\
 &= \sum_{k \in B^c} R_{i,k} H_{k,j},
 \end{aligned}$$

that is $G = RH$. □

The Markov chain V_B contains only one recurrent class, which we denote by B' , composed of the states of B that are directly accessible from B^c . More precisely,

$$B' = \{j \in B \mid \exists i \in B^c, P_{i,j} > 0\}.$$

If $B' \neq B$, we denote by B'' the set $B \setminus B'$. The subsets B', B'' form a partition of B which induces the following decomposition of matrices G and H ,

$$G = \begin{pmatrix} G' & 0 \\ G'' & 0 \end{pmatrix} \text{ and } H = \begin{pmatrix} H' & 0 \end{pmatrix}. \quad (1)$$

In the same way, the partition B' , B'' , B^c of S leads to the following decomposition of P ,

$$P = \begin{pmatrix} P_{B'} & P_{B'B''} & P_{B'B^c} \\ P_{B''B'} & P_{B''} & P_{B''B^c} \\ P_{B^cB'} & 0 & P_{B^c} \end{pmatrix}.$$

The matrices H' , G' and G'' are given by the following theorem.

Theorem 2 *We have*

$$\begin{aligned} H' &= (I - P_{B^c})^{-1} P_{B^c B'}, \\ G' &= (I - Q_{B'B''} P_{B''B'})^{-1} (Q_{B'B''} P_{B''B^c} + (I - P_{B'})^{-1} P_{B'B^c}) H', \\ G'' &= (I - Q_{B''B'} P_{B'B''})^{-1} (Q_{B''B'} P_{B'B^c} + (I - P_{B''})^{-1} P_{B''B^c}) H', \end{aligned}$$

where

$$Q_{B'B''} = (I - P_{B'})^{-1} P_{B'B''} (I - P_{B''})^{-1} \text{ and } Q_{B''B'} = (I - P_{B''})^{-1} P_{B''B'} (I - P_{B'})^{-1}.$$

Proof Since $P_{B^c B''} = 0$, we get

$$H = (I - P_{B^c})^{-1} P_{B^c B} = (H' \ 0),$$

with $H' = (I - P_{B^c})^{-1} P_{B^c B'}$. Next, from $G = RH$ or equivalently

$$(I - P_B) G = G - P_B G = P_{BB^c} H,$$

we have $G = P_B G + P_{BB^c} H$. Using now the decomposition of matrices G , P_B , P_{BB^c} , and H with respect to the partition $\{B', B''\}$ of B , we obtain

$$\begin{cases} G' = P_{B'} G' + P_{B'B''} G'' + P_{B'B^c} H', \\ G'' = P_{B''B'} G' + P_{B''} G'' + P_{B''B^c} H'. \end{cases}$$

This gives

$$\begin{cases} G' = (I - P_{B'})^{-1} P_{B'B''} G'' + (I - P_{B'})^{-1} P_{B'B^c} H', \\ G'' = (I - P_{B''})^{-1} P_{B''B'} G' + (I - P_{B''})^{-1} P_{B''B^c} H'. \end{cases}$$

Putting the second relation in the first one leads to the expression of G' and putting the first relation in the second one leads to the expression of G'' . \square

We denote by $v^{(n)}$ the distribution of the state from which the n th sojourn of X in B starts, which means that $v^{(n)}$ is the distribution of $V_{B,n}$. Since V_B is a Markov chain, we have, for every $n \geq 1$,

$$v^{(n)} = v^{(1)}G^{n-1}.$$

Using these results we derive the distribution of $S_{B,n}$, the n th sojourn time of X in subset B .

Theorem 3 *For every $n \geq 1$ and for all $t \geq 0$, we have*

$$\mathbb{P}\{S_{B,n} > t\} = v^{(n)}e^{A_B t} \mathbb{1} = v^{(1)}G^{n-1}e^{A_B t} \mathbb{1}.$$

Proof Using a classical backward renewal argument, as for instance in [18], we have, for every $i \in B$,

$$\mathbb{P}\{S_{B,1} > t \mid X_0 = i\} = \sum_{j \in S} P_{i,j} \int_0^\infty \mathbb{P}\{S_{B,1} > t \mid Z_1 = j, T_1 = x, X_0 = i\} \lambda e^{-\lambda x} dx,$$

where T_1 is the first occurrence instant of the Poisson process $\{N_t, t \geq 0\}$. We then have

$$\begin{aligned} \mathbb{P}\{S_{B,1} > t \mid X_0 = i\} &= \int_t^\infty \lambda e^{-\lambda x} dx \\ &+ \sum_{j \in B} P_{i,j} \int_0^t \mathbb{P}\{S_{B,1} > t - x \mid X_0 = j\} \lambda e^{-\lambda x} dx. \end{aligned}$$

The change of variable $x := t - x$ in the second integral leads to

$$\mathbb{P}\{S_{B,1} > t \mid X_0 = i\} = e^{-\lambda t} \left(1 + \sum_{j \in B} P_{i,j} \int_0^t \mathbb{P}\{S_{B,1} > x \mid X_0 = j\} \lambda e^{\lambda x} dx \right).$$

Introducing the column vector $w(t)$ defined by $w_i(t) = \mathbb{P}\{S_{B,1} > t \mid X_0 = i\}$, for every $i \in B$, we get $w(0) = \mathbb{1}$ and

$$w(t) = e^{-\lambda t} \left(\mathbb{1} + \lambda P_B \int_0^t w(x) e^{\lambda x} dx \right).$$

Differentiating with respect to t leads to

$$w'(t) = -\lambda w(t) + \lambda P_B w(t) = A_B w(t),$$

that is

$$w(t) = e^{A_B t} w(0) = e^{A_B t} \mathbb{1}. \tag{2}$$

For every $n \geq 1$ and for all $t \geq 0$, we have, using now the Markov property, the homogeneity of X and (2),

$$\begin{aligned}
\mathbb{P}\{S_{B,n} > t\} &= \sum_{i \in B} v_i^{(n)} \mathbb{P}\{S_{B,n} > t \mid V_{B,n} = i\} \\
&= \sum_{i \in B} v_i^{(n)} \mathbb{P}\{S_{B,1} > t \mid X_0 = i\} \\
&= v^{(n)} e^{A_B t} \mathbb{1},
\end{aligned}$$

which completes the proof. \square

The next theorem gives the limiting distribution of $S_{B,n}$ when n tends to infinity. To determine the limiting distribution, we need the following lemma.

Lemma 2 *The row vector π_B satisfies*

$$\pi_B = \pi_B P_{BB^c} (I - P_{B^c})^{-1} P_{B^c B} (I - P_B)^{-1}.$$

Proof Since $\pi A = 0$, we also have $\pi P = \pi$, that is,

$$\begin{cases} \pi_B = \pi_B P_B + \pi_{B^c} P_{B^c B} \\ \pi_{B^c} = \pi_B P_{BB^c} + \pi_{B^c} P_{B^c}. \end{cases}$$

The second equation gives

$$\pi_{B^c} = \pi_B P_{BB^c} (I - P_{B^c})^{-1}.$$

The first one gives

$$\pi_B = \pi_{B^c} P_{B^c B} (I - P_B)^{-1}.$$

Replacing the value of π_{B^c} in this last relation, we get

$$\pi_B = \pi_B P_{BB^c} (I - P_{B^c})^{-1} P_{B^c B} (I - P_B)^{-1},$$

which completes the proof. \square

Theorem 4 *For every $t \geq 0$, the sequence $\mathbb{P}\{S_{B,n} > t\}$ converges in the Cesàro sense when n tends to infinity to $v e^{A_B t} \mathbb{1}$, where*

$$v = \frac{\pi_{B^c} P_{B^c B}}{\pi_{B^c} P_{B^c B} \mathbb{1}}$$

is the stationary distribution of the Markov chain V_B . The convergence is simple if and only if matrix G' is aperiodic.

Proof It suffices to prove that the sequence $v^{(n)}$ converges in the Cesàro sense. The Markov chain V_B has a finite state space B and a single recurrent class B' , so it

has a unique invariant distribution which we denote by v . We thus have $vG = v$, with $v\mathbb{1} = 1$. According to the partition B', B'' of B , we write $v = (v' \ 0)$ and the decomposition of matrix G described in (1) leads, for every $n \geq 1$, to

$$G^n = \begin{pmatrix} G'^n & 0 \\ G''G'^{n-1} & 0 \end{pmatrix}.$$

The general properties of Markov chains tell us that G'^n converges in the Cesàro sense to $\mathbb{1}v'$ and that the convergence is simple if and only if G' is aperiodic. In the same way, $G''G'^{n-1}$ also converges to $\mathbb{1}v'$, since $G''\mathbb{1} = \mathbb{1}$. Using Lemma 2, we obtain

$$\pi_B(I - P_B) = \pi_B P_{BB^c} (I - P_{B^c})^{-1} P_{B^c B},$$

which implies that $\pi_B(I - P_B) = \pi_B(I - P_B)G$. Normalizing this vector, we have

$$v = \frac{\pi_B(I - P_B)}{\pi_B(I - P_B)\mathbb{1}} = \frac{\pi_{B^c} P_{B^c B}}{\pi_{B^c} P_{B^c B}\mathbb{1}},$$

which completes the proof. □

Remark that there is no relation between the periodicity of matrix P and the periodicity of matrix G' . The four situations in which each matrix is either periodic or aperiodic are possible as shown in Figs. 1 and 2. In all cases, the state space is $S = \{1, 2, 3, 4\}$, and we have $B = \{1, 2\}$ and $B^c = \{3, 4\}$. An arrow between two states means that the corresponding transition probability is positive.



Fig. 1 On the left graph P and G' are both aperiodic. On the right graph both P and G' are periodic



Fig. 2 On the left graph P is periodic and G' is aperiodic. On the right graph P is aperiodic and G' is periodic

3 Joint Distributions of Sojourn Times

In many situations, we are interested in functions of several sojourn times of X in a subset of states B , which means that we need the joint distribution of these random times. For instance, when B is composed of the operational states only, we would like to evaluate the random variable $\min_{n \leq N} S_{B,n}$, or we may want to move control variables in the model in order to obtain that for some $N \in \mathbb{N}$, $t > 0$ and $\varepsilon > 0$, we have $\mathbb{P}\{S_{B,1} > t, S_{B,2} > t, \dots, S_{B,N} > t\} > 1 - \varepsilon$. The point is that the random variables $S_{B,1}, S_{B,2}, \dots$ are in general dependent. To get a feeling of this, just consider the example depicted in Fig. 3, where $B = \{1, 2\}$. If ε is small, the sojourn times of X in state 1 are “very short” and in state 2 are “very long.” The topology of the chain says that knowing that previous sojourn was a short one, it is highly probable that next one will be short again.

If we are interested in studying the correlations between successive operational times, we need to evaluate second order moments, and, again, we need the joint distribution of sojourn times. This is the topic of this subsection.

Recall that the matrices R and H are defined by $R = (I - P_B)^{-1} P_{B B^c}$ and $H = (I - P_{B^c})^{-1} P_{B^c B}$ and that we have $G = RH$. We also recall that $V_{B^c,1} = X_{T_{B^c,1}}$ and that for every $i, \ell \in B$ and $j, k \in B^c$,

$$\mathbb{P}\{V_{B^c,1} = j \mid X_0 = i\} = R_{i,j} \text{ and } \mathbb{P}\{V_{B,1} = \ell \mid X_0 = k\} = H_{k,\ell}.$$

We first give a lemma that will be used in the next theorem.

Lemma 3 *For all $s, t \geq 0$, we have*

$$\mathbb{P}\{S_{B,1} > t, V_{B^c,1} = j \mid X_0 = i\} = (e^{A_B t} R)_{i,j}, \text{ for } i \in B, j \in B^c,$$

$$\mathbb{P}\{S_{B^c,1} > s, V_{B,1} = j \mid X_0 = i\} = (e^{A_{B^c} s} H)_{i,j}, \text{ for } i \in B^c, j \in B,$$

$$\mathbb{P}\{S_{B,1} > t, S_{B^c,1} > s, V_{B,2} = j \mid X_0 = i\} = (e^{A_B t} R e^{A_{B^c} s} H)_{i,j}, \text{ for } i \in B, j \in B.$$

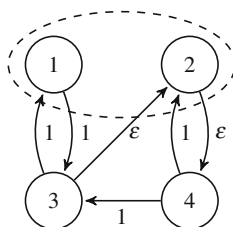


Fig. 3 Illustrating the dependence in the sequence of sojourn times of X in a subset B of states; here, $B = \{1, 2\}$. If $S_{B,n-1}$ was small, it is highly probable that the $(n - 1)$ th sojourn in B was a holding time in state 1. We then expect that the next one, $S_{B,n}$, will be small as well

Proof We introduce the matrix $L(t)$ defined, for every $i \in B$ and $j \in B^c$, by

$$L_{i,j}(t) = \mathbb{P}\{S_{B,1} > t, V_{B^c,1} = j \mid X_0 = i\}.$$

Using again classical backward renewal arguments, as for instance in [18], we have

$$\begin{aligned} L_{i,j}(t) &= \mathbb{P}\{S_{B,1} > t, Z_1 = j \mid X_0 = i\} \\ &\quad + \sum_{k \in B} \mathbb{P}\{S_{B,1} > t, V_{B^c,1} = j, Z_1 = k \mid X_0 = i\} \\ &= P_{i,j} e^{-\lambda t} + \sum_{k \in B} P_{i,k} e^{-\lambda t} \mathbb{P}\{V_{B^c,1} = j \mid X_0 = k\} \\ &\quad + \sum_{k \in B} P_{i,k} \int_0^t L_{k,j}(t-x) \lambda e^{-\lambda x} dx \\ &= e^{-\lambda t} \left(P_{i,j} + \sum_{k \in B} P_{i,k} R_{k,j} + \sum_{k \in B} P_{i,k} \int_0^t L_{k,j}(x) \lambda e^{\lambda x} dx \right). \end{aligned}$$

This gives in matrix notation

$$L(t) = e^{-\lambda t} \left(P_{B B^c} + P_B R + \lambda P_B \int_0^t L(x) e^{\lambda x} dx \right).$$

Differentiating with respect to t leads to

$$L'(t) = -\lambda L(t) + \lambda P_B L(t) = A_B L(t),$$

which gives $L(t) = e^{A_B t} L(0)$ and since $L(0) = R$, we get

$$L(t) = e^{A_B t} R,$$

which completes the proof of the first relation. The second relation follows immediately from the first one by interchanging the role played by subsets B and B^c . The third relation is easily obtained from the first two. Indeed, for every $i, j \in B$, we have, using the Markov property and the homogeneity of X ,

$$\begin{aligned} &\mathbb{P}\{S_{B,1} > t, S_{B^c,1} > s, V_{B,2} = j \mid X_0 = i\} \\ &= \sum_{k \in B^c} \mathbb{P}\{S_{B,1} > t, V_{B^c,1} = k, S_{B^c,1} > s, V_{B,2} = j \mid X_0 = i\} \\ &= \sum_{k \in B^c} \mathbb{P}\{S_{B^c,1} > s, V_{B,2} = j \mid S_{B,1} > t, V_{B^c,1} = k, X_0 = i\} (e^{A_B t} R)_{i,k} \end{aligned}$$

$$\begin{aligned}
&= \sum_{k \in B^c} \mathbb{P}\{S_{B^c,1} > s, V_{B,1} = j \mid X_0 = k\} (e^{A_{B^c} t} R)_{i,k} \\
&= \sum_{k \in B^c} (e^{A_{B^c} t} R)_{i,k} (e^{A_{B^c} s} H)_{k,j} \\
&= (e^{A_{B^c} t} R e^{A_{B^c} s} H)_{i,j},
\end{aligned}$$

which completes the proof. \square

Theorem 5 For every $n \geq 1$, for all $t_1, \dots, t_n \geq 0$ and $s_1, \dots, s_n \geq 0$, we have

$$\begin{aligned}
&\mathbb{P}\{S_{B,1} > t_1, S_{B^c,1} > s_1, \dots, S_{B,n} > t_n, S_{B^c,n} > s_n\} \\
&= \alpha_B \left[\prod_{k=1}^{n-1} e^{A_{B^c} t_k} R e^{A_{B^c} s_k} H \right] e^{A_{B^c} t_n} R e^{A_{B^c} s_n} \mathbb{1} \\
&\quad + \alpha_{B^c} \left[\prod_{k=1}^{n-1} e^{A_{B^c} s_k} H e^{A_{B^c} t_k} R \right] e^{A_{B^c} s_n} H e^{A_{B^c} t_n} \mathbb{1}. \tag{3}
\end{aligned}$$

Proof The proof is made by recurrence over integer n . Consider first the case $i \in B$. For $n = 1$, we have, using the third relation of Lemma 3 and the fact that $H \mathbb{1} = \mathbb{1}$,

$$\begin{aligned}
\mathbb{P}\{S_{B,1} > t_1, S_{B^c,1} > s_1 \mid X_0 = i\} &= \sum_{j \in B} \mathbb{P}\{S_{B,1} > t_1, S_{B^c,1} > s_1, V_{B,2} = j \mid X_0 = i\} \\
&= \sum_{j \in B} (e^{A_{B^c} t_1} R e^{A_{B^c} s_1} H)_{i,j} \\
&= (e^{A_{B^c} t_1} R e^{A_{B^c} s_1} \mathbb{1})_i,
\end{aligned}$$

which is Relation (3) since the product is equal to the identity matrix when $n = 1$. Suppose that Relation (3) is true for integer $n - 1$. We have, using the Markov property, the homogeneity of X , the third relation of Lemma 3 and the recurrence hypothesis,

$$\begin{aligned}
&\mathbb{P}\{S_{B,1} > t_1, S_{B^c,1} > s_1, \dots, S_{B,n} > t_n, S_{B^c,n} > s_n \mid X_0 = i\} \\
&= \sum_{j \in B} \mathbb{P}\{V_{B,2} = j, S_{B,1} > t_1, S_{B^c,1} > s_1, \dots, S_{B,n} > t_n, S_{B^c,n} > s_n \mid X_0 = i\} \\
&= \sum_{j \in B} \mathbb{P}\{S_{B,1} > t_1, S_{B^c,1} > s_1, V_{B,2} = j \mid X_0 = i\} \\
&\quad \times \mathbb{P}\{S_{B,2} > t_2, S_{B^c,2} > s_2, \dots, S_{B,n} > t_n, S_{B^c,n} > s_n \mid V_{B,2} = j\} \\
&= \sum_{j \in B} (e^{A_{B^c} t_1} R e^{A_{B^c} s_1} H)_{i,j} \\
&\quad \times \mathbb{P}\{S_{B,1} > t_2, S_{B^c,1} > s_2, \dots, S_{B,n-1} > t_n, S_{B^c,n-1} > s_n \mid X_0 = j\}
\end{aligned}$$

$$\begin{aligned}
&= \sum_{j \in B} \left(e^{A_B t_1} R e^{A_{B^c} s_1} H \right)_{i,j} \left(\left[\prod_{k=2}^{n-1} e^{A_B t_k} R e^{A_{B^c} s_k} H \right] e^{A_B t_n} R e^{A_{B^c} s_n} \mathbb{1} \right)_j \\
&= \left(\left[\prod_{k=1}^{n-1} e^{A_B t_k} R e^{A_{B^c} s_k} H \right] e^{A_B t_n} R e^{A_{B^c} s_n} \mathbb{1} \right)_i.
\end{aligned}$$

In the same way, interchanging the role played by subsets B and B^c , we obtain, for every $i \in B^c$,

$$\begin{aligned}
&\mathbb{P}\{S_{B,1} > t_1, S_{B^c,1} > s_1, \dots, S_{B,n} > t_n, S_{B^c,n} > s_n \mid X_0 = i\} \\
&= \left(\left[\prod_{k=1}^{n-1} e^{A_{B^c} s_k} H e^{A_B t_k} R \right] e^{A_{B^c} s_n} H e^{A_B t_n} \mathbb{1} \right)_i.
\end{aligned}$$

Unconditioning with respect to the initial state gives the result. \square

Corollary 1 For every $n \geq 1$ and for all $t_1, \dots, t_n \geq 0$, we have

$$\mathbb{P}\{S_{B,1} > t_1, \dots, S_{B,n} > t_n\} = v^{(1)} \left[\prod_{k=1}^{n-1} e^{A_B t_k} G \right] e^{A_B t_n} \mathbb{1}, \quad (4)$$

Proof Putting $s_1 = \dots = s_n = 0$ in Theorem 3 gives the result. \square

We obtain, in the same way the joint distribution of the n first sojourn times in subset B^c by interchanging the role played by subsets B and B^c .

For instance, as illustrated at the beginning of the subsection, if we want to make sure that the first N operational periods are long enough, that is, formally, if we want that the probability that each of these periods lasts at least t units of time is, at least, $1 - \varepsilon$, we must check that

$$\mathbb{P}\{S_{B,1} > t, \dots, S_{B,N} > t\} = v^{(1)} (e^{A_B t} G)^{N-1} e^{A_B t} \mathbb{1} \geq 1 - \varepsilon.$$

The possible independence of the sequences $(S_{B,n})$ and $(S_{B^c,n})$ is discussed in [13].

4 Sum of the n First Sojourn Times

We focus in this section on the distribution of the sum of the n first sojourn times. We denote this random variable by $TS_{B,n}$. We then have

$$TS_{B,n} = \sum_{\ell=1}^n S_{B,\ell}.$$

The distribution of $TS_{B,n}$ is given by the following theorem which uses the next lemma. We first introduce the column vectors $w_B(n, t)$ and $w_{B^c}(n, t)$ defined by

$$w_B(n, t) = (\mathbb{P}\{TS_{B,n} > t \mid X_0 = i\}, i \in B)$$

and

$$w_{B^c}(n, t) = (\mathbb{P}\{TS_{B,n} > t \mid X_0 = i\}, i \in B^c).$$

Lemma 4 *For every $n \geq 1$ and for all $t \geq 0$, we have*

$$w_{B^c}(n, t) = Hw_B(n, t).$$

Proof For $i \in B^c$, we have, using the Markov property and since $X_0 = Z_0$,

$$\begin{aligned} \mathbb{P}\{TS_{B,n} > t \mid X_0 = i\} &= \sum_{j \in S} \mathbb{P}\{TS_{B,n} > t, Z_1 = j \mid Z_0 = i\} \\ &= \sum_{j \in B} P_{i,j} \mathbb{P}\{TS_{B,n} > t \mid X_0 = j\} \\ &\quad + \sum_{j \in B^c} P_{i,j} \mathbb{P}\{TS_{B,n} > t \mid X_0 = j\}. \end{aligned}$$

This gives, in matrix notation,

$$w_{B^c}(n, t) = P_{B^c B} w_B(n, t) + P_{B^c B^c} w_{B^c}(n, t),$$

that is

$$w_{B^c}(n, t) = (I - P_{B^c})^{-1} P_{B^c B} w_B(n, t) = Hw_B(n, t),$$

which completes the proof. \square

Theorem 6 *For every $n \geq 1$ and for all $t \geq 0$, we have*

$$\mathbb{P}\{TS_{B,n} > t\} = w_n e^{M_n t} \mathbb{1},$$

where $w_n = (v^{(1)} \ 0 \ \dots \ 0)$ is the row vector with length $n|B|$ (each 0 represents here the null vector with length $|B|$) and M_n is the $(n|B|, n|B|)$ matrix given by

$$M_n = \begin{pmatrix} Q_1 & Q_2 & 0 & 0 & \cdots & & 0 & 0 \\ 0 & Q_1 & Q_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & Q_1 & Q_2 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & & \cdots & & 0 & Q_1 & Q_2 \\ 0 & 0 & & \cdots & & 0 & Q_1 & Q_2 \end{pmatrix}$$

with $Q_1 = A_B$ and $Q_2 = -A_{BB^c} (A_{B^c})^{-1} A_{B^c B}$ (each 0 represents here the $(n|B|, n|B|)$ null matrix).

Proof For $n = 1$, the result is immediate since $w_1 = v^{(1)}$ and $M_1 = Q_1 = A_B$. Let $n \geq 2$. We use classical backward renewal arguments, see for instance [18]. For every $i \in B$, we have

$$\begin{aligned} \mathbb{P}\{TS_{B,n} > t | X_0 = i\} &= \sum_{j \in S} P_{i,j} \int_0^\infty \mathbb{P}\{TS_{B,n} > t | Z_1 = j, T_1 = x, X_0 = i\} \lambda e^{-\lambda x} dx \\ &= \sum_{j \in S} P_{i,j} \int_0^t \mathbb{P}\{TS_{B,n} > t | Z_1 = j, T_1 = x, X_0 = i\} \lambda e^{-\lambda x} dx \\ &\quad + \int_t^\infty \lambda e^{-\lambda x} dx \\ &= \sum_{j \in B} P_{i,j} \int_0^t \mathbb{P}\{TS_{B,n} > t - x | X_0 = j\} \lambda e^{-\lambda x} dx \\ &\quad + \sum_{j \in B^c} P_{i,j} \int_0^t \mathbb{P}\{TS_{B,n-1} > t - x | X_0 = j\} \lambda e^{-\lambda x} dx + e^{-\lambda t}, \end{aligned}$$

where T_1 is the first occurrence instant of the Poisson process $\{N_t, t \geq 0\}$. The change of variable $x := t - x$ leads to

$$\begin{aligned} \mathbb{P}\{TS_{B,n} > t | X_0 = i\} &= e^{-\lambda t} \left(1 + \sum_{j \in B} P_{i,j} \int_0^t \mathbb{P}\{TS_{B,n} > x | X_0 = j\} \lambda e^{\lambda x} dx \right. \\ &\quad \left. + \sum_{j \in B^c} P_{i,j} \int_0^t \mathbb{P}\{TS_{B,n-1} > x | X_0 = j\} \lambda e^{\lambda x} dx \right). \end{aligned}$$

Using the column vectors $w_B(n, t)$ and $w_{B^c}(n, t)$ defined above, we obtain

$$w_B(n, t) = e^{-\lambda t} \left(\mathbb{1} + \lambda P_B \int_0^t w_B(n, x) e^{\lambda x} dx + \lambda P_{BB^c} \int_0^t w_{B^c}(n-1, x) e^{\lambda x} dx \right).$$

Differentiating with respect to t we get

$$w'_B(n, t) = -\lambda w_B(n, t) + \lambda (P_B w_B(n, t) + P_{BB^c} w_{B^c}(n-1, t)).$$

Using Lemma 4, we have

$$\begin{aligned} w'_B(n, t) &= -\lambda w_B(n, t) + \lambda (P_B w_B(n, t) + P_{BB^c} (I - P_{B^c})^{-1} P_{B^c B} w_B(n-1, t)) \\ &= -\lambda (I - P_B) w_B(n, t) + \lambda P_{BB^c} (I - P_{B^c})^{-1} P_{B^c B} w_B(n-1, t). \end{aligned}$$

Note that since $-\lambda (I - P) = A$, we have $-\lambda (I - P_B) = A_B$, $\lambda P_{BB^c} (I - P_{B^c})^{-1} P_{B^c B} = -A_{BB^c} (A_{B^c})^{-1} A_{B^c B}$ and thus

$$\begin{aligned} w'_B(n, t) &= A_B w_B(n, t) - A_{BB^c} (A_{B^c})^{-1} A_{B^c B} w_B(n-1, t) \\ &= Q_1 w_B(n, t) + Q_2 w_B(n-1, t). \end{aligned}$$

Introducing the column vector $u_B(n, t)$ defined by

$$u_B(n, t) = (w_B(n, t), w_B(n-1, t), \dots, w_B(1, t)),$$

this gives

$$u'_B(n, t) = M_n u_B(n, t)$$

and thus, since $u_B(n, 0) = \mathbb{1}$,

$$u_B(n, t) = e^{M_n t} u_B(n, 0) = e^{M_n t} \mathbb{1}.$$

Finally,

$$\mathbb{P}\{TS_{B,n} > t\} = \alpha_B w_B(n, t) + \alpha_{B^c} w_{B^c}(n, t).$$

Using Lemma 4, we get

$$\mathbb{P}\{TS_{B,n} > t\} = (\alpha_B + \alpha_{B^c} H) w_B(n, t) = v^{(1)} w_B(n, t) = w_n u_B(n, t) = w_n e^{M_n t} \mathbb{1},$$

which completes the proof. \square

To compute the distribution of $TS_{B,n}$ for a fixed n we proceed as follows. Let β be a positive real number such that $\beta \geq \max\{\lambda_i, i \in B\}$. The matrix T_n defined by $T_n = I + M_n/\beta$ is substochastic and given by

$$T_n = \begin{pmatrix} P_1 & P_2 & 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & P_1 & P_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & P_1 & P_2 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & & 0 & P_1 & P_2 \\ 0 & 0 & \cdots & & 0 & P_1 & P_1 \end{pmatrix},$$

where $P_1 = I + Q_1/\beta$ and $P_2 = Q_2/\beta$. We then have

$$\mathbb{P}\{TS_{B,n} > t\} = w_n e^{M_n t} \mathbb{1} = \sum_{k=0}^{\infty} e^{-\beta t} \frac{(\beta t)^k}{k!} w_n (T_n)^k \mathbb{1}.$$

The special form of matrix T_n leads to the following form of its k th power,

$$(T_n)^k = \begin{pmatrix} (P_1)^k & W_{n-1,k} \\ 0 & (T_{n-1})^k \end{pmatrix},$$

where, by writing $(T_n)^k = T_n (T_n)^{k-1}$, the matrix $W_{n-1,k}$, which is a $(|B|, (n-1)|B|)$ matrix, is given for $k, n \geq 2$ by

$$W_{n-1,k} = P_1 W_{n-1,k-1} + W_{n-1,1} (T_{n-1})^{k-1}$$

with $W_{n-1,1} = (P_2 \ 0 \ \cdots \ 0)$. If $x_B(n, k)$ denotes the column vector composed of the $|B|$ first entries of vector $(T_n)^k \mathbb{1}$, we have

$$x_B(n, 0) = \mathbb{1} \text{ for } n \geq 1, \quad x_B(1, k) = (P_1)^k \mathbb{1} \text{ for } k \geq 0$$

and, for $n \geq 2$ and $k \geq 1$,

$$\begin{aligned} x_B(n, k) &= (P_1)^k \mathbb{1} + W_{n-1,k} \mathbb{1} \\ &= (P_1)^k \mathbb{1} + P_1 W_{n-1,k-1} \mathbb{1} + W_{n-1,1} (T_{n-1})^{k-1} \mathbb{1} \\ &= (P_1)^k \mathbb{1} + P_1 W_{n-1,k-1} \mathbb{1} + P_2 x_B(n-1, k-1) \\ &= P_1 ((P_1)^{k-1} \mathbb{1} + W_{n-1,k-1} \mathbb{1}) + P_2 x_B(n-1, k-1) \\ &= P_1 x_B(n, k-1) + P_2 x_B(n-1, k-1). \end{aligned} \tag{5}$$

We then have

$$\mathbb{P}\{TS_{B,n} > t\} = \sum_{k=0}^{\infty} e^{-\beta t} \frac{(\beta t)^k}{k!} v^{(1)} x_B(n, k).$$

Let ε be a given specified error tolerance associated with the computation of the distribution of $TS_{B,n}$ and let K be the integer defined by

$$K = \min \left\{ k \in \mathbb{N} \mid \sum_{j=0}^k e^{-\beta t} \frac{(\beta t)^j}{j!} \geq 1 - \varepsilon \right\}.$$

This gives

$$\mathbb{P}\{TS_{B,n} > t\} = \sum_{k=0}^K e^{-\beta t} \frac{(\beta t)^k}{k!} v^{(1)} x_B(n, k) + e(K),$$

where

$$e(K) = \sum_{k=K+1}^{\infty} e^{-\beta t} \frac{(\beta t)^k}{k!} v^{(1)} x_B(n, k) \leq \sum_{k=K+1}^{\infty} e^{-\beta t} \frac{(\beta t)^k}{k!} = 1 - \sum_{k=0}^K e^{-\beta t} \frac{(\beta t)^k}{k!} \leq \varepsilon.$$

We consider now the moments of the sum of the n first sojourn times of X in subset B . From Theorem 6, we have

$$\mathbb{E} \left\{ (TS_{B,n})^k \right\} = (-1)^k k! w_n (M_n)^{-k} \mathbb{1}.$$

The expected value of $TS_{B,n}$ is given by

$$\mathbb{E}\{TS_{B,n}\} = \sum_{\ell=1}^n \mathbb{E}\{S_{B,\ell}\} = -v^{(1)} \left(\sum_{\ell=0}^{n-1} G^\ell \right) (A_B)^{-1} \mathbb{1}.$$

For the higher moments of $TS_{B,n}$, we need the following lemma. We denote by $M_n[i, j]$ the submatrix of matrix M_n , of dimension $(|B|, |B|)$ defined for $i, j = 1, \dots, n$ by

$$M_n[i, j] = \begin{cases} Q_1 & \text{if } i = j, \\ Q_2 & \text{if } i = j - 1, \\ 0 & \text{otherwise.} \end{cases}$$

We define in the same way the blocks $(M_n)^{-k}[i, j]$ of matrix $(M_n)^{-k}$. For $k = 1$, these blocks are given by the following result.

Lemma 5 For every $i, j = 1, \dots, n$, we have

$$(M_n)^{-1} [i, j] = \begin{cases} (-1)^{j-i} (Q_1)^{-1} (Q_2 (Q_1)^{-1})^{j-i} = G^{j-i} (Q_1)^{-1} & \text{if } i \leq j, \\ 0 & \text{if } i > j. \end{cases}$$

Proof The matrix M_n being upper triangular, the matrix $(M_n)^{-1}$ is also upper triangular. For $i = j$, we clearly have $(M_n)^{-1} [i, i] = (Q_1)^{-1}$. For $i < j$, by writing $I = M_n (M_n)^{-1}$, we have $0 = Q_1 (M_n)^{-1} [i, j] + Q_2 (M_n)^{-1} [i + 1, j]$, that is

$$(M_n)^{-1} [i, j] = - (Q_1)^{-1} Q_2 (M_n)^{-1} [i + 1, j].$$

Since $(M_n)^{-1} [i, i] = (Q_1)^{-1}$ we easily get the first equality recursively. The second one follows immediately from the first one. \square

For every $k \geq 2$, it is easily checked that the matrix $(M_n)^{-k}$ has the same structure as matrices M_n and $(M_n)^{-1}$, that is, that the blocks $(M_n)^{-k} [i, j]$ only depend on the difference $j - i$. We thus get

$$\begin{aligned} \mathbb{E} \left\{ (TS_{B,n})^2 \right\} &= 2w_n (M_n)^{-2} \mathbb{1} \\ &= 2v^{(1)} \sum_{j=1}^n (M_n)^{-2} [1, j] \mathbb{1} \\ &= 2v^{(1)} \sum_{j=1}^n \sum_{h=1}^j (M_n)^{-1} [1, h] (M_n)^{-1} [h, j] \mathbb{1} \\ &= 2v^{(1)} \sum_{j=1}^n \sum_{h=1}^j G^{h-1} (Q_1)^{-1} G^{j-h} (Q_1)^{-1} \mathbb{1}. \end{aligned}$$

For $k \geq 2$, we obtain

$$\begin{aligned} (M_n)^{-k} [1, j] &= \sum_{h=1}^j (M_n)^{-1} [1, h] (M_n)^{-k+1} [h, j] \\ &= \sum_{h=1}^j G^{h-1} (Q_1)^{-1} (M_n)^{-k+1} [1, j - h + 1]. \end{aligned}$$

Introducing the column vectors $\theta_n(k, j)$ of dimension $|B|$, defined by $\theta_n(k, j) = (M_n)^{-k} [1, j] \mathbb{1}$, we have the following recurrence relation to compute them.

$$\begin{aligned} \theta_n(k, j) &= \sum_{h=1}^j G^{h-1} (Q_1)^{-1} (M_n)^{-k+1} [1, j - h + 1] \mathbb{1} \\ &= \sum_{h=1}^j G^{h-1} (Q_1)^{-1} \theta_n(k - 1, j - h + 1) \\ &= \sum_{h=1}^j G^{j-h} (Q_1)^{-1} \theta_n(k - 1, h), \end{aligned}$$

with $\theta_n(1, j) = G^{j-1} (Q_1)^{-1} \mathbb{1}$, for $j = 1, \dots, n$. We then have

$$\begin{aligned} \mathbb{E} \left\{ (TS_{B,n})^k \right\} &= (-1)^k k! w_n (M_n)^{-k} \mathbb{1} \\ &= (-1)^k k! v^{(1)} \sum_{j=1}^n (M_n)^{-k} [1, j] \mathbb{1} \\ &= (-1)^k k! v^{(1)} \sum_{j=1}^n \theta_n(k, j). \end{aligned}$$

5 Extension to Absorbing Chains with Rewards

We consider now the case where the state space S of X is composed of transient states and an absorbing state denoted by a . The subset of operational states is denoted by B and we denote by B' the set of the other transient states. We then have the partition $S = B \cup B' \cup \{a\}$. The set of nonoperational states is thus $B^c = B' \cup \{a\}$. A reward rate or performance level r_i is associated with each state $i \in S$. We suppose that we have $r_i > 0$ for every $i \in B$ and we denote by R_B the $(|B|, |B|)$ diagonal matrix with entries r_i , for $i \in B$. As we will see, the value of the rewards associated with the other states has no influence on the sojourn times considered here. The partition $B, B', \{a\}$ of the state space S induces the following decomposition of matrices A and $P = I + A/\lambda$, and vector α .

$$A = \begin{pmatrix} A_B & A_{BB'} & A_{Ba} \\ A_{B'B} & A_{B'} & A_{B'a} \\ 0 & 0 & 0 \end{pmatrix}, P = \begin{pmatrix} P_B & P_{BB'} & P_{Ba} \\ P_{B'B} & P_{B'} & P_{B'a} \\ 0 & 0 & 1 \end{pmatrix} \text{ and } \alpha = (\alpha_B \ \alpha_{B'} \ \alpha_a).$$

For $n \geq 1$, we denote by $S_{i,B,n}$ the total time spent by X in state $i \in B$ during the n th sojourn of X in B , if the latter exists. If the process gets absorbed before the n th sojourn of X in B , we set $S_{i,B,n} = 0$. For $n \geq 1$, the random variable $S_{B,n}$

representing the accumulated reward during the n th sojourn of X in B is defined by

$$S_{B,n} = \sum_{i \in B} r_i S_{i,B,n}.$$

Following the same steps used for the irreducible case, it is easily checked that the distribution $v^{(n)}$ of the random variable $V_{B,n}$ representing the state of B in which the n th sojourn of X in B starts is given, for every $n \geq 1$, by $v^{(n)} = v^{(1)} G^{n-1}$, where

$$v^{(1)} = \alpha_B + \alpha_{B'}(I - P_{B'})^{-1} P_{B'B} = \alpha_B - \alpha_{B'}(A_{B'})^{-1} A_{B'B},$$

$$G = (I - P_B)^{-1} P_{BB'}(I - P_{B'})^{-1} P_{B'B} = (A_B)^{-1} A_{BB'}(A_{B'})^{-1} A_{B'B}.$$

The distribution of $S_{B,n}$ is given by the following theorem.

Theorem 7 *For every $n \geq 1$ and for all $t \geq 0$, the distribution of the accumulated reward in B during the n th sojourn of X in B , is given by*

$$\mathbb{P}\{S_{B,n} > t\} = v^{(1)} G^{n-1} e^{(R_B)^{-1} A_B t} \mathbb{1}.$$

Proof The proof is quite similar to the proof of Theorem 3. Using a classical backward renewal argument, as for instance in [18], we have, for every $i \in B$,

$$\mathbb{P}\{S_{B,1} > t | X_0 = i\} = \sum_{j \in S} P_{i,j} \int_0^\infty \mathbb{P}\{S_{B,1} > t | Z_1 = j, T_1 = x, X_0 = i\} \lambda e^{-\lambda x} dx,$$

where T_1 is the first occurrence instant of the Poisson process $\{N_t, t \geq 0\}$. We then have

$$\begin{aligned} \mathbb{P}\{S_{B,1} > t | X_0 = i\} &= \int_{t/r_i}^\infty \lambda e^{-\lambda x} dx \\ &+ \sum_{j \in B} P_{i,j} \int_0^{t/r_i} \mathbb{P}\{S_{B,1} > t - r_i x | X_0 = j\} \lambda e^{-\lambda x} dx. \end{aligned}$$

The change of variable $x := t - r_i x$ in the second integral leads to

$$\mathbb{P}\{S_{B,1} > t | X_0 = i\} = e^{-\lambda t/r_i} \left(1 + \sum_{j \in B} P_{i,j} \int_0^t \mathbb{P}\{S_{B,1} > x | X_0 = j\} \frac{\lambda}{r_i} e^{\lambda x/r_i} dx \right).$$

Introducing the column vector $w(t)$ defined by $w_i(t) = \mathbb{P}\{S_{B,1} > t | X_0 = i\}$, for every $i \in B$, we get $w(0) = \mathbb{1}$ and

$$w(t) = e^{-\lambda(R_B)^{-1}t} \left(\mathbb{1} + \int_0^t e^{\lambda(R_B)^{-1}x} \lambda(R_B)^{-1} P_B w(x) dx \right).$$

Differentiating with respect to t leads to

$$w'(t) = -\lambda(R_B)^{-1}w(t) + \lambda(R_B)^{-1}P_Bw(t) = (R_B)^{-1}A_Bw(t),$$

that is,

$$w(t) = e^{(R_B)^{-1}A_Bt}w(0) = e^{(R_B)^{-1}A_Bt}\mathbb{1}. \tag{6}$$

For every $n \geq 1$ and for all $t \geq 0$, we have, using now the Markov property, the homogeneity of X and (6),

$$\begin{aligned} \mathbb{P}\{S_{B,n} > t\} &= \sum_{i \in B} v_i^{(n)} \mathbb{P}\{S_{B,n} > t \mid V_{B,n} = i\} \\ &= \sum_{i \in B} v_i^{(n)} \mathbb{P}\{S_{B,1} > t \mid X_0 = i\} \\ &= v^{(n)} e^{(R_B)^{-1}A_Bt} \mathbb{1}, \end{aligned}$$

which completes the proof. □

In the same way, the distribution of the accumulated reward over the n first sojourn times of X in B is given by the following result.

Theorem 8 *For every $n \geq 1$ and for all $t \geq 0$, we have*

$$\mathbb{P}\{TS_{B,n} > t\} = w_n e^{M_n t} \mathbb{1},$$

where $w_n = (v^{(1)} \ 0 \ \dots \ 0)$ is the row vector with length $n|B|$ (each 0 represents here the null vector with length $|B|$) and M_n is the $(n|B|, n|B|)$ matrix given by

$$M_n = \begin{pmatrix} Q_1 & Q_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & Q_1 & Q_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & Q_1 & Q_2 & 0 & 0 & \dots & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & & 0 & Q_1 & Q_2 \\ 0 & 0 & \dots & & 0 & Q_1 & Q_2 \end{pmatrix}$$

with $Q_1 = (R_B)^{-1}A_B$ and $Q_2 = -(R_B)^{-1}A_{BB'}(A_{B'})^{-1}A_{B'B}$ (each 0 represents here the $(n|B|, n|B|)$ null matrix).

Proof For $n = 1$, the result is immediate since $w_1 = v^{(1)}$ and $M_1 = Q_1 = (R_B)^{-1}A_B$. Let $n \geq 2$. We use the same classical backward renewal arguments already used in

the proof of Theorem 6. For every $i \in B$, we have

$$\begin{aligned} \mathbb{P}\{TS_{B,n} > t \mid X_0 = i\} &= \sum_{j \in B} P_{i,j} \int_0^{t/r_i} \mathbb{P}\{TS_{B,n} > t - r_i x \mid X_0 = j\} \lambda e^{-\lambda x} dx \\ &\quad + \sum_{j \in B'} P_{i,j} \int_0^{t/r_i} \mathbb{P}\{TS_{B,n-1} > t - r_i x \mid X_0 = j\} \lambda e^{-\lambda x} dx \\ &\quad + P_{i,a} e^{-\lambda t/r_i}. \end{aligned}$$

The rest of the proof is identical to the proofs of Theorems 6 and 7. □

Note that unlike the irreducible case, the matrix G is substochastic and if we denote by N_B the total number of visits to the subset B until absorption, the events $\{N_B > k\}$ and $\{S_{B,k+1} > 0\}$ are equal for every $k \geq 0$. It follows that

$$\mathbb{P}\{N_B > k\} = \mathbb{P}\{S_{B,k+1} > 0\} = v^{(1)} G^k \mathbb{1}.$$

In particular, we have

$$\mathbb{E}\{N_B\} = \sum_{k=0}^{\infty} v^{(1)} G^k \mathbb{1} = v^{(1)} (I - G)^{-1} \mathbb{1}.$$

Observe that if we know that the process has visited the set B at least n times, for $n \geq 1$, that is, given that $S_{B,n} > 0$, the evaluation of the accumulated reward during the n th sojourn in B changes. The conditional distribution of $S_{B,n}$ given that $S_{B,n} > 0$ writes, for $n \geq 1$ and $t \geq 0$,

$$\mathbb{P}\{S_{B,n} > t \mid S_{B,n} > 0\} = \frac{\mathbb{P}\{S_{B,n} > t\}}{\mathbb{P}\{S_{B,n} > 0\}} = \frac{v^{(1)} G^{n-1} e^{(R_B)^{-1} A_B t} \mathbb{1}}{v^{(1)} G^{n-1} \mathbb{1}}.$$

The total accumulated reward in subset B until absorption is defined by

$$TS_{B,\infty} = \sum_{n=1}^{\infty} S_{B,n}.$$

Its distribution is given in [3] for semi-Markov reward processes. In the case of Markov reward processes, it becomes

$$\mathbb{P}\{TS_{B,\infty} > t\} = v^{(1)} e^{(Q_1 + Q_2)t} \mathbb{1}.$$

Finally, the distribution of the sojourn times $S_{B,n}$ for semi-Markov reward processes has been obtained in [15].

6 Notes

As stated in the introduction, the analysis of sojourn times of Markov models in subsets of their state spaces, in the dependability area, started apparently in [10], in the irreducible case. In biology, close related work, with differences however, was known before (see [6]). There are other papers related to the analysis of these objects. For instance, [1, 4] provide bounds on reliability metrics by exploiting the fact that in many dependability models, when the system is highly reliable, there is a huge difference in the holding times the chain spends in its states.

Perhaps even closer to this chapter, we can mention [2], where the authors define a “conditional MTTF” in the case of a system subject to very different failures. If the chain has, say, two absorbing states a and a' , representing two different situations where the system has failed, depending on the causes of such a failure, it makes sense to analyze the mean time the system operates given that it will be absorbed in state a , that is, the metric $\mathbb{E}\{T \mid X_\infty = a\}$, where T is the system’s life-time. If, more generally, we are interested in some subset B of transient states, we can look at the n th sojourn time of the process X in B , if it exists.

Using the notation in this chapter, let us briefly outline how to derive the distribution of the n th sojourn time of X in B , given that $X_\infty = a$. As in Sect. 5, we simplify the presentation avoiding the fact that we must define $S_{B,n}$ and $V_{B,n}$ for all n , in particular when the n th sojourn in B does not exist (in that case, $S_{B,n} = 0$ and $V_{B,n}$ is assigned a specific extra state). After doing that and simplifying, we can write

$$\begin{aligned} \mathbb{P}\{S_{B,n} > t \mid X_\infty = a\} &= \sum_{i \in B} \mathbb{P}\{S_{B,n} > t, V_{B,n} = i \mid X_\infty = a\} \\ &= \sum_{i \in B} \mathbb{P}\{S_{B,n} > t \mid V_{B,n} = i, X_\infty = a\} \mathbb{P}\{V_{B,n} = i \mid X_\infty = a\} \\ &= \sum_{i \in B} \mathbb{P}\{S_{B,1} > t \mid X_0 = i\} \mathbb{P}\{V_{B,n} = i \mid X_\infty = a\}. \end{aligned}$$

Then, for every $i, j \in B$,

$$\begin{aligned} \mathbb{P}\{V_{B,n+1} = j \mid V_{B,n} = i, X_\infty = a\} &= \frac{\mathbb{P}\{X_\infty = a \mid V_{B,n+1} = j, V_{B,n} = i\} G_{i,j}}{\mathbb{P}\{X_\infty = a \mid V_{B,n} = i\}} \\ &= \frac{\mathbb{P}\{X_\infty = a \mid V_{B,n+1} = j\} G_{i,j}}{\mathbb{P}\{X_\infty = a \mid V_{B,n} = i\}} \\ &= \frac{\mathbb{P}\{X_\infty = a \mid X_0 = j\} G_{i,j}}{\mathbb{P}\{X_\infty = a \mid X_0 = i\}}. \end{aligned}$$

The quantities $\mathbb{P}\{X_\infty = a \mid X_0 = k\}$ are well-known in Markov chain theory, and the distributions $\mathbb{P}\{S_{B,1} > t \mid X_0 = k\}$ were given in Sect. 2. This type of computation also appears in [7], where bounds of the asymptotic availability, and more generally of the asymptotic reward, are derived. The Markov model considered is irreducible

on a state space partitioned into three classes, say B , C , C' . The object of interest is the sojourn time of the process in B , given that when leaving B the process will visit C next. Concerning Sect. 2, see that we can get more information about sojourn times following the same lines as described there. For instance, assume that we are interested in the last state of B visited by X during its n th visit to that subset. Call it $W_{B,n}$, and call $w^{(n)}$ its distribution: $\mathbb{P}\{W_{B,n} = i\} = w_i^{(n)}$. Following the same path as in Theorem 1, we have for $n \geq 1$,

$$w^{(n)} = w^{(1)} M^{n-1} = v^{(n)} (I - P_B)^{-1},$$

where $M = P_{BB^c} (I - P_{B^c})^{-1} P_{B^c B} (I - P_B)^{-1}$. Note that this is the matrix appearing in Lemma 2, where it is stated that $\pi_B = \pi_B M$.

In [13], we analyze conditions under which the sequence of sojourn times of a Markov chain X in a subset B of states is i.i.d., with applications always in the analysis of dependability properties. A particularly important metric appearing in dependability, more complex to analyze than previously considered ones, is the interval availability over an interval $[0, t]$, which is the random variable

$$IA_t = \frac{1}{t} \int_0^t 1_{\{X_s \in B\}} ds,$$

where B is the set of operational states. In words, IA_t is the fraction of $[0, t]$ during which the system is operational. The first paper where the distribution of this variable is proposed using the uniformization techniques (as we do in this chapter) is [19]. We proposed some improvements to the algorithms (including the possibility of dealing with infinite state spaces) in [14]. Many of the results described here can be extended to semi-Markov processes, and also to the case where the states in the model are weighted by rewards, or costs. Some of these extensions have been presented in Sect. 5. See also [11] or [15]. The monograph [5] also discusses many of these results together with several other related issues.

References

1. Bobbio A, Trivedi KS (1986) An aggregation technique for the transient analysis of stiff Markov chains. *IEEE Trans Comput* 35(9)
2. Choi H, Wang W, Trivedi KS (1998) Analysis of conditional MTTF of fault-tolerant systems. *Microelectronic Reliability* 38(3)
3. Ciardo G, Marie R, Sericola B, Trivedi KS (1990) Performability analysis using semi-Markov reward processes. *IEEE Trans Comput* 39(10)
4. Couto da Silva AP, Rubino G (2006) Bounding the mean cumulated reward up to absorption. In: Langville A, Stewart W (eds) *Markov anniversary meeting*, Boscov Books
5. Csenki A (1994) Dependability for systems with a partitioned state space. *Markov and semi-Markov theory and computational implementation*. Lecture notes in statistics, 90. Springer

6. Hawkes AG, Cui L, Du S (2014) Occupancy times for markov and semi-markov models in systems reliability. In: Chapter 16 reliability applied reliability engineering and risk analysis: probabilistic models and statistical inference. Wiley
7. Mahévas S, Rubino G (2001) Bound computation of dependability and performability measures. *IEEE Trans Comput* 50(5)
8. Muppala J, Fricks R, Trivedi KS (2000) Techniques for system dependability evaluation. In: Grassman W (ed) *Computational probability*. Kluwer Academic Publishers, The Netherlands
9. Reibman A, Smith RM, Trivedi KS (1989) Markov and markov reward models: a survey of numerical approaches. *Eur J Oper Res* 40
10. Rubino G, Sericola B (1989) Sojourn times in Markov processes. *J Appl Probab* 26
11. Rubino G, Sericola B (1989) Distribution of operational times in fault-tolerant systems modeled by semi-Markov processes. In: 12th International Conference on Fault-Tolerant Systems and Diagnostics. Prague, Czech Republic, September 1989
12. Rubino G, Sericola B (1991) A finite characterization of weak lumpable Markov processes. Part I: The discrete time case. *Stoch Proces Appl* 38
13. Rubino G, Sericola B (1992) Interval availability analysis using operational periods. *Perform Eval* 14(3)
14. Rubino G, Sericola B (1993) Interval availability distribution computation. In: Proceedings of the 23rd IEEE international symposium on fault tolerant computing (FTCS'23), Toulouse, France, June 1993
15. Rubino G, Sericola B (1993) Sojourn times in semi-Markov reward processes. Application to fault-tolerant systems modelling. *Reliab Eng Syst Safety* 41
16. Rubino G, Sericola B (1993) A finite characterization of weak lumpable Markov processes. Part II: The continuous time case. *Stoch Process Appl* 45
17. Rubino G, Sericola B (2014) *Markov chains and dependability theory*. Cambridge University Press
18. Sericola B (2013) *Markov Chains: theory, algorithms and applications*. Iste Series, Wiley
19. de Souza e Silva E, Gail HR (1986) Calculating cumulative operational time distributions of repairable computer systems. *IEEE Trans Comput* 35(4)
20. Trivedi KS (2001) *Probability and statistics with reliability, queuing, and computer science applications*. 2nd edn, Prentice Hall

Managed Dependability in Interacting Systems

Poul E. Heegaard, Bjarne E. Helvik, Gianfranco Nencioni
and Jonas Wäfler

Abstract A digital ICT infrastructure must be considered as a system of systems in itself, but also in interaction with other critical infrastructures such as water distributions, transportation (e.g. Intelligent Transport Systems) and Smart Power Grid control. These systems are characterised by self-organisation, autonomous sub-systems, continuous evolution, scalability and sustainability, providing both economic and social value. Services delivered involve a chain of stakeholders that share the responsibility, providing robust and secure services with stable and good performance. One crucial challenge for the different operation/control centres of the stakeholders is to manage dependability during normal operation, which may be characterised by many failures of minor consequence. In seeking to optimise the utilisation of the available resources with respect to dependability, new functionality is added with the intension to help assist in obtaining situational awareness, and for some parts enable autonomous operation. This new functionality adds complexity, such that the complexity of the (sub)systems and their operation will increase. As a consequence of adding a complex system to handle complexity, the frequency and severity of the consequences of such events may increase. Furthermore, as a side-effect of this, the preparedness will be reduced for restoration of services after a major event (that might involve several stakeholders), such as common software breakdown, security attacks, or natural disaster. This chapter addresses the dependability challenges related to the above-mentioned system changes. It is important to understand how *adding complexity to handle complexity* will influence the risks, both with respect to the consequences and the probabilities. In order to increase insight, a dependability

P.E. Heegaard (✉) · B.E. Helvik · G. Nencioni · J. Wäfler
Norwegian University of Science and Technology, Department of Telematics,
O.S. Bragstads Plass 2b, N-7491, Trondheim, Norway
e-mail: poul.heegaard@item.ntnu.no

B.E. Helvik
e-mail: bjarne.helvik@item.ntnu.no

G. Nencioni
e-mail: gianfranco.nencioni@item.ntnu.no

J. Wäfler
e-mail: jonas.waefler@item.ntnu.no

modelling approach is taken, where the goal is to combine and extend the existing modelling approaches in a novel way. The objective is to quantify different strategies for management of dependability in interacting systems. Two comprehensive system examples are used to illustrate the approach. A software-defined networking example addresses the effect of moving control functionality from being distributed and embedded with the primary function, to be separated and (virtually) centralised. To demonstrate and discuss the consequences of adding more functionality both in the distributed entities serving the primary function, and centralised in the control centre, a Smart Grid system example is studied.

1 Introduction

The private and public ICT service-provisioning infrastructure has developed over many years into a complex system and its interactions with other critical infrastructure systems such as water distribution, transportation (e.g. Intelligent Transport Systems) and Smart Power Grid control have created diverse digital ecosystems. Digital ecosystems are characterised by self-organisation, autonomous subsystems, continuous evolution, scalability and sustainability, providing both economic and social value. Services delivered involve a chain of stakeholders that share the responsibility, providing robust and secure services with stable and good performance.

This evolution has been evident for some time. In spite of this, and the crucial role of such systems, not much research is directed toward ensuring the dependability of the services provided by such an ecosystem of systems. The objective of this chapter is to address some of the issues that arise when we seek to manage the dependability of systems.

1.1 Challenges

One crucial challenge for the different operation and control centres of the different systems is to manage the dependability in normal operation with many failures of minor consequence. In seeking to optimise the utilisation of the available resources with respect to dependability [1], the complexity of the (sub)systems and their operation will increase due to increased interconnectedness and complexity.

Some issues to take into consideration include:

- The public ICT services are the result of the cooperation between a huge number of markets actors. The overall system providing these services are not engineered, and there is no aggregate insight into their design and operation.
- There is no coordinated management to deal with issues involving several autonomous systems, in spite of such issues being a likely cause of extensive problems and outages.

- It is necessary to prepare for restoration of service after a major event such as common software breakdown, security attacks, or natural disasters. This preparation must include technical and operational as well as organisational and societal aspects.

An additional challenge is the management of dependability over multiple network domains, with uncoordinated operations in each of the different domains. As a potential side-effect of this, the preparedness for restoration of services after a major event (that might involve several stakeholders) such as common software breakdown, security attacks or a natural disaster will be reduced. In addition, the frequency and consequences of such events may increase. More focus on exercises and use of the improved situational awareness provided by the new operational functionality, will to some extent reduce the negative side effect.

Ensuring the dependability of services based on an interacting relationship between independent stakeholders in the provision is typically agreed upon through Service Level Agreements (SLAs), which give guarantees on the non-functional properties of the services, including dependability aspects such as interval availability. These are important means to ensure the dependability of the services, but are insufficient to prevent and handle dependability problems across providers, as outlined above.

New functionality is added to enhance and improve operation and management of complex digital ecosystems. This is done to rationalise the operation, save money, simplify resource management, and maximise utilisation. It also enables more timely and precise knowledge and information about system state, facilitating timely (proactive) maintenance and reducing the frequency and consequences of failures. The operational cost is reduced by reduction in manual labour through better and quicker detection and diagnostic mechanisms, and more autonomous self-repair. The objective is to shorten the recovery time and to reduce the failure frequency through better proactive maintenance. It should be kept in mind that this functionality targets the frequent (everyday) failures which are anticipated in the system design and normally of low consequence. However, this increased maintainability is achieved by the introduction of new, and partly centralised functionality, that increases the total complexity and creates an interdependent system [8]. These systems not only have additional failures and failure modes [12, 22], but they may also manifest a more fragile behaviour in critical situations [2, 18].

Figure 1 illustrates a risk curve, where the events with high “probability” have low consequence and the events with low “probability” have high consequence. The introduction of ICT-based support system, to operate an ICT system, or a critical infrastructure such as Smart Grid, is expected to reduce the consequences and probability of daily events. Less human resources are needed for the daily operations. However, due to the introduction of another ICT-based system, the complexity and interdependency in a system will increase, with the potential consequence of increased probability of critical events with extensive and long lasting consequences. Such events affect large parts of the system and take a long time to recover from because of lack of understanding of the complexity (“we have not seen this

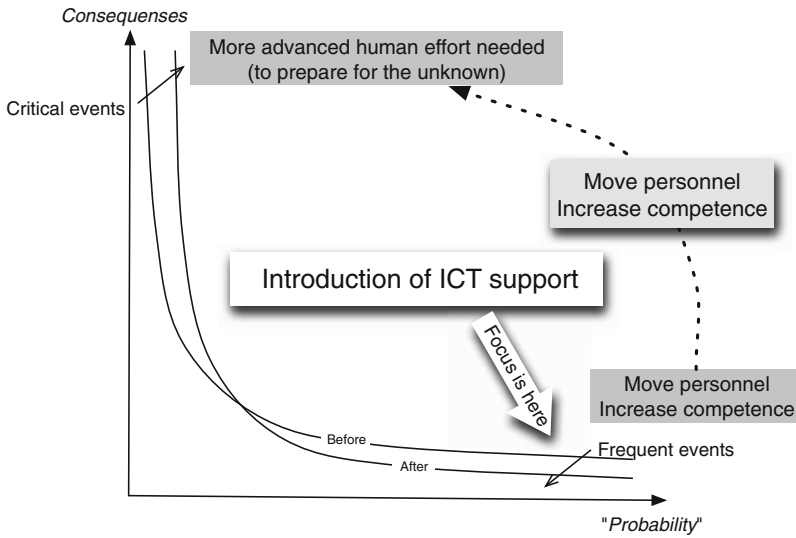


Fig. 1 Introducing ICT support to assist daily operations may increase the overall risk

failure before”), or the lack of maintenance support and coordination between the different subsystems and domains in the digital ecosystem (“who should do what?”). As indicated in the figure, it is not only necessary to increase the focus and manpower on the events with larger consequences, but also increase the competence of the operation personnel.

There is a lack of theoretical foundation to control the societal and per service dependability of ICT infrastructure in the digital ecosystem. No foundation is established for optimisation, consolidated management, and provision of this infrastructure, neither from a public regulatory perspective, nor from the perspective of groups of autonomous (commercially) co-operating providers. A model of an ICT infrastructure must describe the structure and behaviour of the physical and logical information and network infrastructure, and include the services provided. Furthermore, through the modelling phases, it should be described how *resilience engineering* [9] can be applied to manage the robustness and survivability of the ICT infrastructure ecosystem.

1.2 Outline

This chapter describes the above-mentioned challenges and outlines potential approaches to gain more insight into the risks. To increase the understanding and assess the risk (both consequences and probabilities), a holistic modelling approach is taken of service in systems of systems. The goal is to quantify different strategies

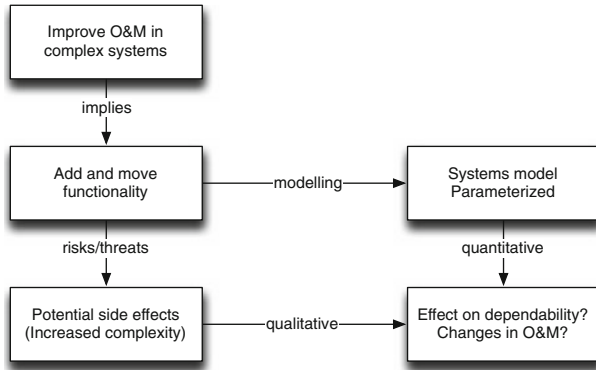


Fig. 2 Understanding the complexity

for management of dependability in interacting systems. This should be addressed by different approaches:

- *System modelling*: Modelling of the functional interaction between embedded technical sub-systems in an ecosystem with multiple actors coordinated via business models only.
- *Management strategies*: Management and provisioning of (digital) ecosystems in a cost-efficient way, considering the trade-off between cost and quality.
- *Quantitative assessment*: Resource allocation optimisation (modelling, measurements, simulations) of robustness/dependability and performance in digital ecosystems.

Figure 2 illustrates that to improve the operation and management (O&M) of complex systems (e.g. in Smart Grids), new control logic and functionality must be added and in some cases also be centralised (e.g. in Software-Defined Networking (SDN), and by the introduction of network function virtualisation NFV in next generation communication networks). This needs to be modelled, and the system models parametrised to quantify the effect on the dependability and to identify potential changes and improvements that can be made in O&M. The reason is that the new and/or moved functionality poses new risks and threats to the systems, and may have potential undesired side-effects that need to be qualitatively assessed to again identify potential changes and improvements that can be made during O&M, and to the O&M systems.

As a step towards gaining this understanding, Sect. 2 discusses how the complexity is changing by adding and moving control logic from being embedded and closely integrated with the functionality to be controlled to being separated and to some extents also centralised. Being able to deal with these issues, the ability to build representative, yet understandable and tractable dependability models are crucial. Seeking to build an entirely new theoretical approach does not seem feasible. Our approach is to extend and combine current approaches in novel manners to reach our objective. Hence, to illustrate this and to exemplify the effect of the changes

in complexity, Sect. 2 includes two simple models with numerical examples. To demonstrate how the complexity might be modelled and assessed, Sect. 3 gives an example of modelling of the increase complexity in SDN, and Sect. 4 provides the same for a Smart Grid example. Finally, our concluding remarks are found in Sect. 5.

2 Complex Digital Ecosystems

As discussed in the previous section, digital ecosystems are complex systems, which are challenging to operate and control. This is due both to their tight integration with other technical systems and the necessity to perform management over multiple system domains where each domain has (partly) uncoordinated operations.

To enhance and improve the operation and maintainability of the complex digital ecosystems, new functionality is *added* and/or *moved and centralised*. As an example, in Software-Defined Networking, the functionality of the control logic is separated from the forwarding functionality in the data plane and *moved* from the distributed control plane residing on the components to be controlled to a virtually centralised control plane. Another example is Smart Grid, where the ICT and power grids are tightly integrated and interdependent. New functionality is *added* both in a distributed manner to enable observability and controllability of the components in the power grid, and centralised in the control centres to implement the control and management.

Adding and moving functionality will contribute to changes in the complexity. The goal is to simplify, or assist handling of complexity. However, adding new hardware and software, or moving the existing, will change the interrelations between functional and logical “entities”/“components”. This means that, even though the total complexity is the same or reduced, the system is less well understood and potentially contains new vulnerabilities and poses new management challenges.

Later in this chapter, two comprehensive system examples are introduced to demonstrate the modelling of this change in complexity. In Sect. 3, a model of Software-Defined Networking is given and in Sect. 4 a Smart Grid example.

2.1 Centralising Distributed Functionality

IP networks are comprised of distributed, coordinated, but autonomous network nodes, where the control logic is embedded and closely integrated with the same forwarding functionality that is to be controlled, as illustrated in Fig. 3a.

In emerging networking technology, the trend is to separate the control and forwarding¹ and to move the control logic from the network nodes to a (virtually) cen-

¹This is similar to how it was done in telephony systems (PSTN) with separate data traffic and signalling traffic using Signalling System 7 (SS7) [10] and in B-ISDN [11].

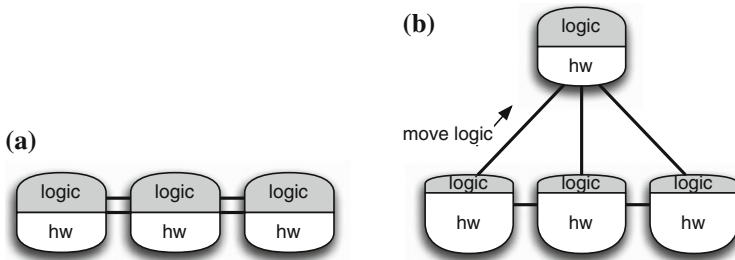


Fig. 3 Moving control logic to enhance the resource utilisation and improve QoS. **a** Distributed logic embedded on the forwarding engine of the network nodes. **b** Logically centralised control logic with simple distributed network nodes

tralised controller. The reduction in the distributed (control logic) functionality and a corresponding increase in the centralised functionality will potentially reduce the complexity in the (partly) autonomous network nodes and increase the complexity of the centralised systems, as illustrated in Fig. 3a.

It is reasonable to assume that a simplification in the functionality will reduce the complexity of the network nodes. If the properties of the hardware platform are unchanged, the network node will then be less error prone. However, if at the same time commodity hardware is used to reduce the node cost then there is a potential risk of decreasing the hardware availability. Then, it is not obvious whether the node availability will improve or not.

The centralisation of the complex functionality should increase the system availability, due to better global overview and coordination. The control logic has comparable (or the same) functionality to the functionality that is moved from the distributed nodes, but additional functionality is needed to coordinate and mitigate the central controllers. Furthermore, centralisation invites new more advanced functionality, for instance consult the motivation for SDN [6, 20, 24]. It is therefore not known what effect the central controllers have on the system availability.

A separation of the forwarding and control functionality does not necessarily mean a separation of the hardware platform and its functionality. A common mistake is to forget that the underlying resources, such as the routing and switching hardware, are typically utilised not only by the primary information handled by the system, such as user packets, but also for the signalling of information exchange necessary to control and manage the very same resources. Such an interdependency has a negative effect on the overall system availability [4].

Whether the system availability is improved or not when centralising complex functionality depends on to what extent the reduced complexity of the functionality will have a positive effect and improve resource utilisation (due to the global system state being availability, which eases resource coordination) compared to the added complexity in the overhead associated with managing the centralised functionality.

Example 1 Availability requirement of the controller. To demonstrate the effect of moving the complexity on availability a very simple example can be considered. Assume that the conventional network in Fig. 3a is modelled as a serial structure

with three network nodes with availability A_{No} . The serial structure of the network nodes is assumed for simplicity and is not regarded as realistic. The new network is a serial structure consisting of the central controller with availability A_C and the three networks nodes with availability A_{Nn} . Since moving the complexity should improve the availability then $A_{No} < A_{Nn}$. The availability requirement of the controller is given by

$$A_C > \left(\frac{A_{No}}{A_{Nn}} \right)^3 \quad (1)$$

If $A_{No} = 0.98$ and $A_{Nn} = 0.99$, then $A_C > 0.97$.

If we have some inherent redundancy in the distributed system, the effect becomes radical. Assuming the elements in the network in Fig. 3a operate in a ideal load-shared mode were on, they can take the entire load. They will then constitute a parallel system and we get $A_C^* \cdot 1 - (1 - A_{Nn})^3 > 1 - (1 - A_{No})^3$, where $A_C^* > 0.999992$.

Later, in Sect. 3, a system model of Software Defined Networking is introduced to address in more detail the effect of moving control functionality from being distributed and embedded with the primary function to be separated and (virtually) centralised.

2.2 Add Distributed and Increase Centralised Functionality

The need for enhanced operation and control in the power grid is an excellent example where the new ICT-based control logic is added to the distributed power grid components. In power distribution grids, the grid components typically contain little or no automated control logic. This means that manual detection and recovery is required, which must be coordinated by the control centre, as illustrated in Fig. 4a.

Figure 4b shows that new functionality must be added to the centralised controller to be able to utilise the new distributed functionality (remote control logic). Centralising functionality to achieve better decisions will provide a single point of failure, performance bottleneck and expose targeted attacks.

The ICT-based control functionality is not only supporting the operations, but needs to be operated in addition to the primary functionality. The technology and functionality will in many cases be new to the organisation and might change the workflows and result in a need for enhanced knowledge and competence in operation.

From a dependability perspective, adding ICT-based control seems to be a bad idea since all the negative side effects pointed out in the previous subsection apply, with functionality added both in the distributed nodes and in the centralised controllers. This produces less-positive effects compared to moving and centralising functionality. However, the new ICT-based control functionality will increase the maintainability through more timely and precise knowledge and information about system state, so timely (proactive) maintenance can be carried out, and hence, the frequency and consequences of the most frequent faults (failures) are reduced. The

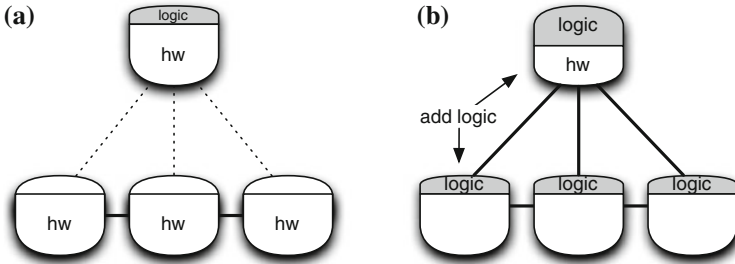


Fig. 4 Adding control logic to enhance the maintainability and improve service reliability. **a** Centralised logic and loosely coupled distributed network nodes. **b** Add logic to both the centralised controller and the distributed network nodes

operational cost is reduced by reduction in manual labour through better and quicker detection mechanisms and more autonomous (self-)repair. The results are reduced recovery times and better proactive maintenance.

It is not guaranteed that the system availability will increase from added (ICT-based) functionality or not. Even though the maintainability is significantly improved, which makes both proactive and reactive maintenance more effective, it is an uncertainty in that the control functionality itself adds complexity that might affect the system availability.

Example 2 Mean component down time. Adding more logic to the components is assumed to reduce the components recovery time, but at the same time increase the component failure intensity. The hardware failure intensity is assumed unchanged, but the added logic might also fail.

To compare the two systems, we should consider the requirements of mean down time (MDT), mean time to failures (MTTF), and availability. In this example, we say that the new system should have the same availability requirement and will then determine the maximum MDT requirement of the component for a given set of failure intensities for the hardware, λ_H , and software, λ_S .

The availability of the original system is:

$$A_{No} = A_{So} \cdot A_H^3 = \frac{\mu_S}{\lambda_S + \mu_S} \cdot \left(\frac{\mu_H}{\lambda_H + \mu_H} \right)^3 \tag{2}$$

while for the modified system with added functionality it is:

$$A_{No} = A_{Sn} \cdot (A_{HS} \cdot A_H)^3 = \frac{\mu_S}{\mu_S + \lambda_{SS}} \cdot \left(\frac{\mu_S \mu_{SS}}{(\lambda_S + \mu_S)(\lambda_H + \mu_{SH})} \right)^3 \tag{3}$$

To retain the same availability level in the new system, the maximum mean down time $MDT = 1/\mu_{HS}$ is determined by $A_{No} < A_{Nn}$. Let the software failure intensity [in minutes^{-1}] for the centralised control logic be $\lambda_{SS} = 0.5\lambda_S$, and $\lambda_H = 1/24$, $\mu_S = 60$, $\lambda_H = 1/168$, $\mu_H = 1$ then $\mu_{HS} > 1.18529$, which means that $MDT < 50.6$ min.

In Sect. 4, a Smart Grid example is introduced to demonstrate and discuss the consequences of adding more functionality, both in the distributed entities serving the primary function and centralised in the control centre.

3 Example: Availability in Software-Defined Networking

The purpose of this section is to present a case study that highlights how the complexity changes by moving the control logic of a system from distributed to centralised. To illustrate this, we extend and combine current approaches in order to model and assess the availability of a new network paradigm. The results show how the management of complex systems is critical from a dependability perspective. In the following, we introduce some details about Software-Defined Networking (SDN) and describe the problem addressed, then we present a two-level hierarchical to evaluate the availability of SDN. Finally, we perform a simple sensitivity analysis on a selected set of parameters that will potentially affect the dependability of SDN.

3.1 Software-Defined Networking

During the recent years, SDN has emerged as a new network paradigm, which mainly consists of a programmable network approach where the forwarding plane is decoupled from the control plane [6, 14]. Despite programmable networks having been studied for decades, SDN is experiencing a growing success because it is expected that the ease of changing protocols and providing support for adding new services and applications will foster future network innovation, which is limited and expensive in today's legacy systems.

A simplified sketch of the SDN architecture from IRFT RFC 7426 [6] without the management plane is depicted in Fig. 5. The control plane and data plane are separated. Here, the control plane is logically centralised in a software-based controller (“network brain”), while the data plane is composed of the network devices (“network arms”) that conducts the packet forwarding.

The control plane has a northbound and a southbound interface. The northbound interface provides an network abstraction to the network applications (e.g. routing protocol, firewall, load balancer, anomaly detection, etc.), while the southbound interface (e.g. OpenFlow) standardises the information exchange between control and data planes.

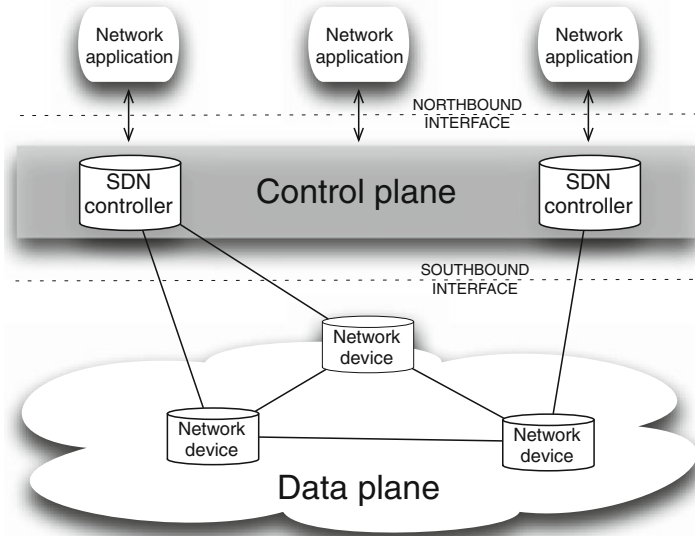


Fig. 5 SDN architecture (exclusive the management plane)

In [20], the following set of potential advantages of SDN were pointed out:

- centralised control;
- simplified algorithms;
- commoditising network hardware;
- eliminating middle-boxes;
- enabling the design and deployment of third-party applications.

However, from a dependability perspective, the SDN poses a set of new vulnerabilities and challenges compared with traditional networking, as discussed in [7]:

- consistency of network information (user plane state information) and controller decisions;
- consistency between the distributed SDN controllers in the control plane;
- increased failure intensities of (commodity) network elements;
- compatibility and interoperability between general purpose, non-standard network elements
- interdependency between path setup in network elements and monitoring of the data plane in the control plane;
- load sharing (to avoid performance bottleneck) and fault tolerance in the control plane have conflicting requirements;

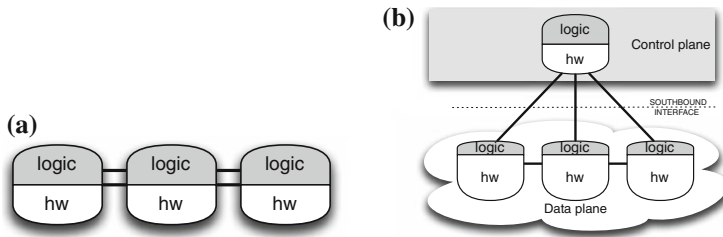


Fig. 6 Software-Defined Networking is an example where the control logic is moved from distributed to virtually centralised (see Fig. 3). **a** Current IP networks: distributed logic embedded on the forwarding engine of the network nodes. **b** SDN: logically centralised control logic combined with simplified network elements

3.2 Problem Description

Traditional IP networks consist of a set of interconnected nodes that include both the data and control planes. Each network node is a complex device that has the functionality of both data forwarding and networking control. To increase the availability and performance of such devices, manufacturers have focused on specialised hardware and software over the past few decades.

As discussed in Sect. 2, SDN has the potential to change the principles of networking and to enhance network flexibility. This implies moving the control logic from the network nodes to a (virtual) centralised controller, and to open up the controllers to a third party via an API (northbound interface), as illustrated in Fig. 6. The transition from a *distributed* network with a focus on establishing and maintaining the connectivity between peering points, to a *centralised* network with a focus on QoS and resource utilisation, will potentially lead to much simpler network nodes with less control logic. The centralised control logic, such as the routing decisions, might be simpler and can even be made more advanced, without making it more complex compared to the distributed solution. The controller has the potential to set up data flows based on a richer set of QoS attributes than in traditional IP networks. However, the coordination and handling of the consistency between the SND controllers, will require new, and complicated logic that will be a critical element to also make SDN a good solution from a dependability perspective.

In the example in this section, we study how the SDN paradigm modifies the overall availability of the network relative to the traditional distributed IP network and analyse which factors dominate in this new scenario.

Although dependability must be regarded as an important issue to make SDN a success, to the best of our knowledge, very limited work on modelling the dependability in SDN availability has been performed. In [17], a model of SDN controllers is developed, while [7] discusses potential dependability challenges with SDN, which is partially illustrated by a small case study with a structural analysis of SDN-enabled network. In this section, we study a comprehensive system model of SDN with respect to dependability.

3.3 Modelling

A two-level hierarchical model is introduced to evaluate the dependability of SDN in a global network. In this example, the dependability is measured in terms of steady-state availability and in the following referred to as availability. The two-level hierarchical modelling approach consists of

- *upper level*: a structural model of the topology of network elements and controllers
- *lower level*: dynamic models (some) of network elements

The approach seeks to avoid the potential uncontrolled growth in model size, by compromising the need for modelling details and at the same time modelling a (very) large-scale network. The detailed modelling is necessary to capture the dependencies that exist between network elements and to describe multiple failure modes that might be found in some of the network elements and in the controllers. The structural model disregards this and assumes independence between the components considered, where a component can be either a single-network element with one failure mode or a set of elements that are interdependent and/or experience several failure modes and an advanced recovery strategy. For the former, we need to use dynamic models such as a Markov model or Stochastic Petri net (e.g. Stochastic Reward Network [3]), and for the latter structural models such as reliability block diagram, fault trees, or structure functions based on minimal cut or path sets.

In the following section, we will demonstrate the use of this approach.

3.3.1 Model Case

In this example, we analyse the availability of a nation-wide backbone network that consists of 10 nodes across 4 cities, and two dual-homed SDN controllers. See Fig. 7 for an illustration of the topology. The nodes are located in the four major cities in Norway, Bergen (BRG), Trondheim (TRD), Stavanger (STV), and Oslo (OSL). Each town has duplicated nodes, except Oslo which has four nodes (OSL1 and OSL2). The duplicated nodes are labelled, X_1 and X_2 , where $X=OSL1, OSL2, BRG, STV,$ and TRD . In addition to the forwarding nodes, there are two dual-homed SDN controllers (SC_1 and SC_2), which are connected to TRD and OSL1.

The objective of the study is to compare the availability of SDN with a traditional IP network with the same topology of network elements (SDN forwarding switched and IP routers). We assume that nodes, links, and controllers in the system may fail. The peering traffic in a city is routed through an access and metro network with a connection to both (all four) nodes in the city. The system is working (up), when all the access and metro networks are connected. Note that for SDN, at least one controller must be reachable from all the nodes along a working path.

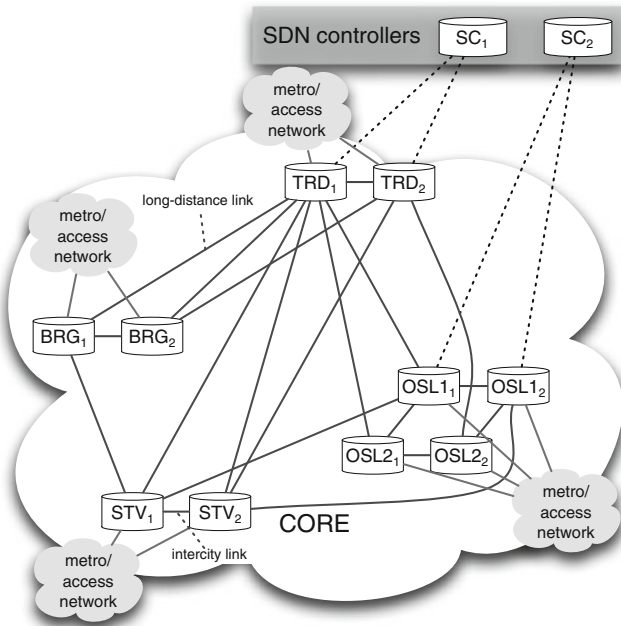


Fig. 7 Case study: nation-wide backbone network

3.3.2 Structural Analysis

The critical parts of the connection between the traffic origins and destinations can be determined using structural analysis based on either *minimal cut sets* S , or *minimal path sets*. The sets are defined as follows:

Definition 1 *Minimal cut set:* The system is failed if and only if all the subsystems in a minimal cut set are failed, given that all the other subsystems that are not in the set are working.

Definition 2 *Minimal path set:* The system is working if and only if all the subsystems in a minimal path set are working, and given that all the subsystems that are not in the set are failed.

We use the *minimum cut sets*, S , to form the basis for a *structure function*, Φ (minimum path sets can also be applied).

Definition 3 *Structure function:* Each max-term of the structure function expressed in a minimal product-of-sums form corresponds to a minimal cut set.

The following connections in SDN must be considered:

- *flow triggering:* a path for the trigger message that should be sent from the source node (at least one node of each city) to at least one SDN controller on arrival of a new flow;

Table 1 Distribution of cardinality of the minimum cut sets for the IP network and SDN

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	Sum
IP network	0	3	8	91	304	360	356	189	70	13				1394
SDN	0	4	15	107	340	520	780	584	302	170	59	31	4	2916

- *network state update and route directives*: a path from the SDN controller to each node;
- *forwarding*: forwarding path from/to each city (6 combinations).

The structural analysis for all the possible connections in the SDN example, shows that the cardinality of the set of minimal cut set S is $\|S\| = 2916$. The cardinality $c_j = \|s_j\|$ of each of the minimal cut sets, $j = 1, \dots, 2916$ is given in Table 1. Each column contains the number of sets that is $C_k = \|\{s_j \in S | c_j = k\}\|, k = 1, \dots, 13$. The table compares the minimal cut sets of SDN with a conventional IP network where the control plane is embedded in the nodes, and hence, no controllers are needed.

The number of minimal cut sets with cardinality one is equal to zero because traffic sources are at least dual-homed and there are two dual-homed control sites.

The number of minimal cut sets C_2 increases from three to four due to the control nodes. Note also that the number of minimal cut sets C_3 almost doubles. This indicates that in this example, a significant increase in vulnerability is observed for the SDN case that is not explained solely by the introduction of a control node, but the fact that a controller must be reachable from every node across the backbone in order for the network to work.

3.3.3 Markov Model of Networks Elements

In order to evaluate the availability of each network element, we develop Markov models of each of the links, traditional routers/switches, SDN routers/switches, and the SDN controllers.

Links

The network model of a link is assumed to be dominated by hardware failures. Therefore, a simple two-state Markov model is used. The links are either up or down due to hardware failure. We use the same model for both traditional networks and SDN. Given failure rate λ_L and repair rate μ_L , the availability of a link is $A_L = \frac{\mu_L}{\lambda_L + \mu_L}$. This model is assumed for each of the components in the structural model.

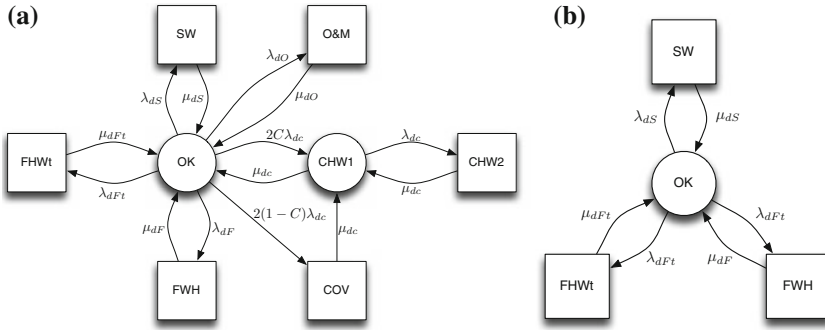


Fig. 8 Markov model of a router/switch. a Traditional network. b SDN

Table 2 State variables for traditional IP router

State	Up/Down	Description
OK	Up	System is fault free
OM	Down	Operation and Maintenance state
CHW1	Up	Hardware failure of one controller
CHW2	Down	Hardware failure both controllers
COV	Down	Coverage state, unsuccessful activation of the stand-by hardware after a failure; manual recovery
FHW	Down	Permanent hardware failure in forwarding plane
FHWt	Down	Transient hardware failure in forwarding plane
SW	Down	Software failure

Routers

The model of a traditional router/switch is depicted in Fig. 8a, where the states are defined in Table 2.

Multiple failures are not included in the model since they are rare and will have an impact significantly smaller than the expected accuracy of the approach.

SDN Forwarding Nodes

Figure 8b shows the model of the forwarding node, i.e. router or switch in an SDN, which corresponds to the traditional IP router. It is significantly simpler. The states related to the control hardware and O&M failures are not contained in this model, since all the control logic is located in the controller. The software is still present but its failure rate will be very low since the functionality is much simpler.

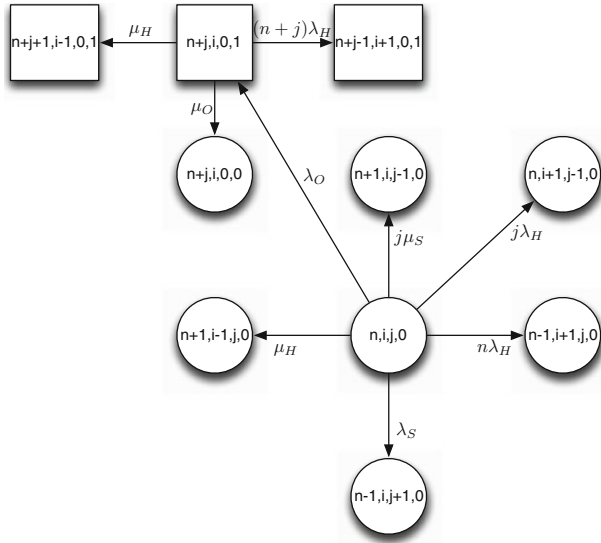


Fig. 9 Generic states of the model of SDN controller

SDN Controller

The model of the SDN controller is composed of two sets of states. One set captures the software and hardware failures. The second set captures the O&M failures in combination with the hardware states of the system. We have assumed that the SDN controller is a cluster of M processors and the system is working, i.e. possesses sufficient capacity if K out of the M processors are active, which means that both software and hardware are working. To represent this scenario, each state is labelled by four numbers $\{n, i, j, k\}$, where n is the number of active processors, i the number of processors down due to hardware failures, j the number of processors down due to software failures and k the state of the O&M functionality ($k = 1$ if O&M mistake, $k = 0$, if not). Figure 9 shows the *outgoing* transitions from a generic state $\{n, i, j, k\}$. The main characteristics of the model are:

- single repairman for a hardware failure;
- load dependency of software failure when the system is working, $\lambda_S(n) = \lambda_S/n$, where the meaning of λ_S is explained in more detail in Sect. 3.4;
- load independence of software failure when the system has failed, $\lambda_S(n) = \lambda_S$;
- when the entire system fails, only processors failed due to hardware failures will be down until the system recovers.

3.3.4 Using Inclusion-Exclusion Principle to Evaluate the System Availability

The inclusion-exclusion principle is a technique to obtain the elements in the union of finite sets. Using the inclusion-exclusion principle on the structure function, we can write the system availability as the probability of the union of all minimal paths:

$$A_S = P\left(\bigcup_{i=1}^n Q_i\right) = \sum_{k=1}^n (-1)^{k-1} \sum_{\substack{\emptyset \neq I \subseteq [n] \\ |I|=k}} P\left(\bigcap_{i \in I} Q_i\right), \quad (4)$$

where $\{Q_1, Q_2, \dots, Q_n\}$ is the set of all minimal paths, and $P(Q_i)$ is the probability of set Q_i .

To compute the probability of the intersection of minimal paths, we need to know the availability of each network element. To this end, we can calculate the element availability by using the proposed Markov models.

3.4 Numerical Evaluation

To evaluate the availability of traditional networks, we consider the typical parameters in Table 3, which are inspired by and taken from several studies [5, 15, 23].

All SDN parameters are expressed relative to the parameters for the traditional network (Table 3). The parameters for the SDN switch you find in Table 4 and for the SDN controller in Table 5. The parameters α_H , α_S and α_O are proportionality factors that are studied in this example.

Using these parameters in the models described in this section, we can compare the (un)availability of traditional IP and SDN networks. Failures with the same cause, have the same intensities in both models. However, we assume that the software on an SDN switch/router will be much less complicated than on a traditional IP router, and we have set the failure rate to zero, for the sake of simplicity. In an SDN controller, all failure rates are N -times larger than in the traditional network, where N is the number of network nodes. This is because we assume that the centralised system needs roughly the same processing capacity and amount of hardware. Therefore, the failure intensity is assumed to be proportional to N , and of the same order of magnitude as the total failure intensity of the traditional distributed IP router system.

The results of a numerical example are given in the plot in Fig. 10. The overall unavailability, i.e. the probability that not all cities in Sect. 3.2 are connected (for SDN this requires also a connection to a controller) is given for different values of α_O . The figure shows that the unavailability increases with about one order of magnitude when α_O changes in the range from 0.1 to 1. The sensitivity of α_H and α_S are far less significant. This indicates that O&M failures are dominant and most critical to the dependability of SDN.

Table 3 Model parameters for the IP network

Intensity	[time]	Description
$1/\lambda_L = 4$	[months]	Expected time to next link failure
$1/\mu_L = 15$	[minutes]	Expected time to link repair
$1/\lambda_{dF} = 6$	[months]	Expected time to next permanent forwarding hardware failure
$1/\mu_{dF} = 12$	[hours]	Expected time to repair permanent forwarding hardware
$1/\lambda_{dFt} = 1$	[week]	Expected time to next transient forwarding hardware failure
$1/\mu_{dFt} = 3$	[minutes]	Expected time to repair transient forwarding hardware
$1/\lambda_{dC} = 6$	[months]	Expected time to next control hardware failure
$1/\mu_{dC} = 12$	[hours]	Expected time to repair control hardware
$1/\lambda_{dS} = 1$	[week]	Expected time to next software failure
$1/\mu_{dS} = 3$	[minutes]	Expected time to software repair
$1/\lambda_{dO} = 1$	[month]	Expected time to next O&M failure
$1/\mu_{dO} = 3$	[hours]	Expected time to O&M repair
$C = 0.97$		Coverage factor

Table 4 Model parameters for SDN switch/router

Intensity	Description
$\lambda_F = \lambda_{dF}$	Intensity of permanent hardware failures
$\mu_F = \mu_{dF}$	Repair intensity of permanent hardware failures
$\lambda_{Ft} = \lambda_{dFt}$	Intensity of transient hardware failures
$\mu_{Ft} = \mu_{dFt}$	Restoration intensity after transient hardware failures
$\lambda_{sS} = 0$	Intensity of software failure

Table 5 Model parameters for SDN controller

Intensity	Description
$\lambda_H = \alpha_H \lambda_{dC} N/K$	Intensity of hardware failures
$\mu_H = \mu_{dC}$	Hardware repair intensity
$\lambda_S = \alpha_S \lambda_{dS} N$	Intensity of software failures
$\mu_S = \mu_{dS}$	Restoration intensity after software failure
$\lambda_O = \alpha_O \lambda_{dO} N$	Intensity of O&M failures
$\mu_O = \mu_O$	Rectification intensity after O&M failures

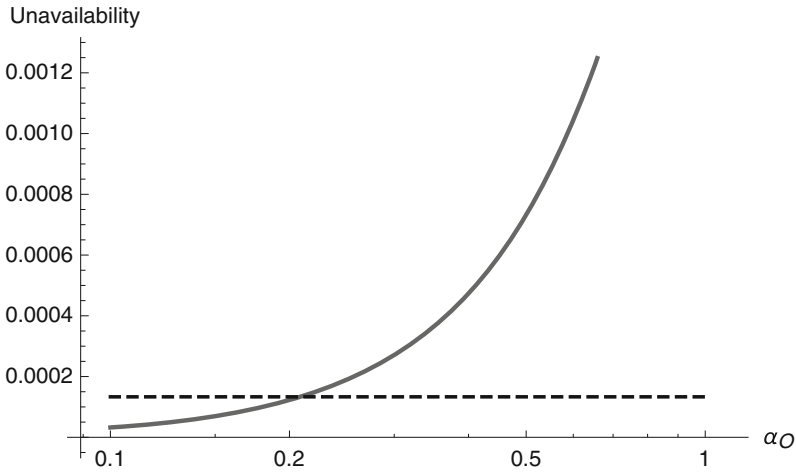


Fig. 10 Unavailability of SDN (solid line) and of traditional network (dashed line) by varying α_O ($\alpha_H = 1$ $\alpha_S = 1$)

As a preliminary conclusion from this study, it seems as the use of commodity hardware and centralised control has a moderate effect on the availability of the overall network. However, the O&M failures and software/logical failures that causes a control cluster to fail are very important in order to improve the dependability when changing from the traditional distributed IP network to SDN.

4 Example: Restoration in Smart Grid

The purpose of this example is to show how the automation of process steps changes the dependability of a system. The system under consideration is a power grid and we focus on the restoration process after a physical failure.

A power grid is a critical infrastructure and its reliability is critical to the smooth operation of a resilient society. Power grids are due to undergo modernisation in the coming years. This next generation power grid is commonly called the smart grid. One of the biggest differences compared to the current grid is additional monitoring information about the current state of the grid and new control abilities throughout the grid. These improvements allow the introduction of more automated processes with the goal of increasing the overall dependability of the system.

This is the starting point of our example. We model the restoration process with and without automation and conduct a dependability analysis. Our results show that the introduction of automation yields benefits like a reduction of down time, but it also extends the system into a compound and more complex system. This system has new failure modes as the automation may malfunction and thus, without taking the appropriate measures, may partially negate benefits.

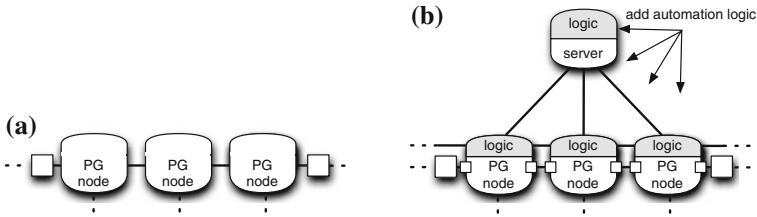


Fig. 11 Schematic view of a protection zone in the current power grid and smart grid. **a** Current power grid: no automated detection and controlling. **b** Smart grid: added logic for automated detection and controlling (distributed and centralised)

4.1 Problem Description

The power grid (PG) has traditionally contained only a few monitoring and controlling devices distributed throughout the grid. Mostly they are deployed in the higher voltage levels. In the lower voltage levels monitoring and controlling devices are, depending on the country, virtually absent. In case of a failure a distributed and autonomously working protection system automatically disconnects a whole protection zone by opening a circuit breaker, causing a power outage to all customers inside this protection zone.

The future power grid, the so called smart grid, will possess monitoring and control systems widely deployed throughout the power grid. These devices detect failures automatically and send failure diagnostics to a central control, operation, and management system. The central system then attempts to isolate the failure by opening other circuit breakers closer to the failure and connecting the rest of the protection zone again to the grid. It is assumed that the power grid at this voltage level has an open ring topology that allows reconnection to the non-isolated parts after a single failure. Figure 11 shows a protection zone in the current PG and in the smart grid, consisting of three PG nodes and two protection devices represented by large squares. The small squares represent new circuit breakers controlled by the centralised control system.

In the following, we study how the introduction of detection and isolation automation changes the characteristics of the restoration process. More precisely, we study the downtime and the energy not supplied (*ENS*), which is the accumulated energy that could not be delivered due to outages, i.e. down time weighted with the load during the outages. Both the lines and the PG nodes can fail, but only larger outages that require a repair crew to go on site are considered.

4.2 Modelling

The restoration process of a power grid failure consists of two stages containing a total of six phases, as shown in Fig. 12. The phases are:

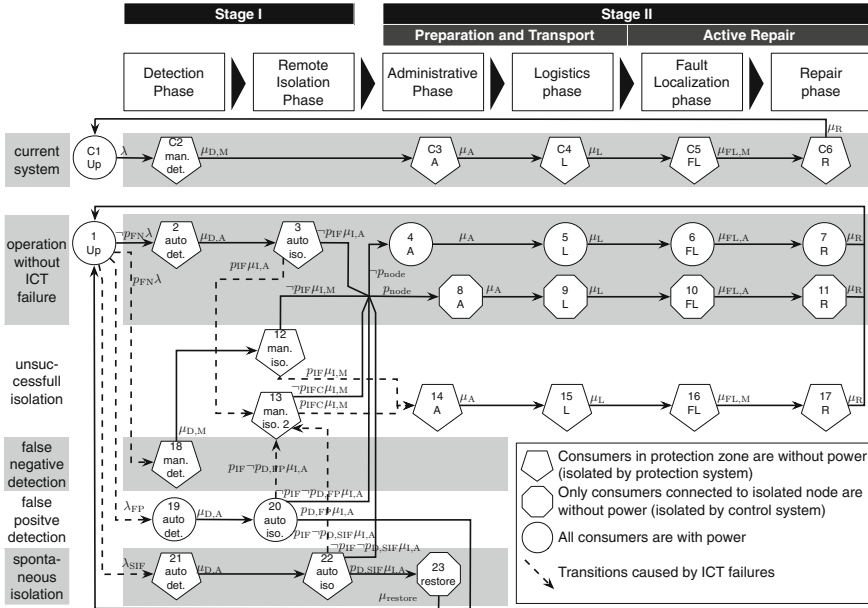


Fig. 12 Phases during the restoration process. For readability reasons, the transitions into state 4 and 8 are displayed in a compact form, it is read as follows; States 3, 12, 13, 20 and 22 each have a transition to 4 and 8. The first is multiplied with $\neg p_{node}$, i.e. $(1 - p_{node})$, the latter with p_{node}

Detection Time period between a failure and its detection in the monitoring system. It is assumed that the protection system disconnects the protection zone containing the failure immediately after the incident. In reality, there is a short delay of several milliseconds. The disconnection leads to a black out in the whole protection zone.

Remote Isolation The failed element is isolated more precisely, either automatically by the central system or manually by a controller at the control centre. The rest of the protection zone is powered up again.

Administrative Failure diagnostics from the monitoring devices are evaluated, the recovery is planned, and a repair crew is assigned.

Logistic Repair crew is equipped with the necessary material and moves to the incident location.

Fault Localization Precise localisation of the failure, both geographically and in the system.

Repair Actual repair, all isolated network elements are restored to normal operation.

The difference between the current power grid and the smart grid lies mainly in *Stage I*. In the current power grid, detection occurs manually, i.e. the failure is detected by a controller or through a call by a consumer. There are no remote isolation

capabilities, so this phase is skipped. Throughout the entire restoration phase, the whole protection zone is without power in the model in Fig. 12. This is denoted by pentagonal states.

In the smart grid, the distributed devices detect the failure automatically and send an alarm together with fault diagnostics to the central system. Now, the failure is isolated automatically and remotely from the central system and *Stage II* begins. If a PG node is affected by the failure, and now isolated, then the system proceeds to state 8. If only a line is isolated then it proceeds to state 4. In the first case, there are still consumers without power. In the latter case, the power supply has been reinstated to all consumers. This difference is indicated in the model by the different shapes of the states. In both cases, the number of consumers affected is smaller than in the current system. An additional difference is the sojourn time of the fault localisation phase. It is shorter for the smart grid, as the detection devices provide fault diagnostics that accelerate this phase.

So far, we have described the process during operation without any failures in the new system. In the following, we consider failures in the information and communication technology (ICT) subsystem used for the automation. It is assumed that all the other systems, e.g. the protection system, work perfectly. The following failures in the detection system are considered:

- **false positive detection failure:** there is no failure, but the detection system reports one.
- **false negative detection failure:** there is a failure but the detection system does not notice it.

A *false positive detection failure* is modelled with a new transition out of state 1 with an additional failure intensity leading to state 19. The failure is detected by the system as before. If the system discovers the false positive failure the restoration process is interrupted and the system goes back to state 1, otherwise it continues.

A *false negative detection failure* is modelled by splitting the transition from state 1 to 2 into two, pointing one to state 18 and weighting the rate by the false negative probability p_{FN} . The new state 18 indicates a manual detection because of the non-detection in the system. The isolation is then done manually by an operator. If the isolation is successful, it proceeds as before either in state 4 or 8 depending on whether a line or a node is affected. If the isolation is not successful, the entire protection zone remains without power for *Stage II* of the restoration process.

In the isolation system, the following failures are considered:

- **isolation failure:** there is a failure, but isolation is unsuccessful because of problems with communication or systems. The whole protection zone remains unpowered.
- **spontaneous isolation failure:** there is no failure, but a network element is falsely isolated by the system.

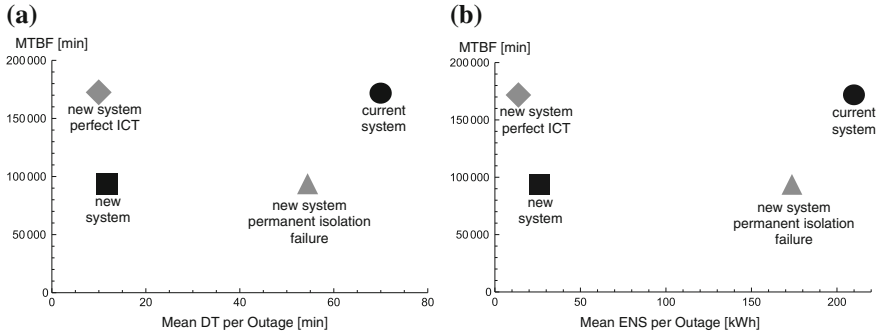


Fig. 13 Mean values per outage. **a** *MTBF* against *MDT* per outage. **b** *MTBF* against *Mean ENS per outage*

An *isolation failure* is modelled in the system by splitting the transitions from the isolation states 3, 12, 13, 20 and 22 into two, and weighting the rate by the probability of an isolation failure p_{IF} , except for the transitions from 13, which uses a higher probability p_{IFC} , because the system already suffered one ICT failure and is in a critical state.

A *spontaneous isolation failure* is modelled with a new transition out of state 1 with an additional failure intensity leading to state 1. The failure is detected by the system as before. If the system discovers that the failure originates from the isolation system and not the power grid, it restores the system (state 23) and goes back to the up state; otherwise, it continues.

4.3 Numerical Example

All event times in the system are assumed to be exponentially distributed with the following expected values. The event times are based on data for longer outages from the Norwegian regulator [21] (Table 6).

First, we compute *MDT* and the mean time between failure (*MTBF*) for the model in Fig. 12. All states in which there is a power outage are considered as down states, i.e. all states but the round states. *MTBF* is computed with

$$MTBF = 1 / \left(\sum_{i \in \Omega_{Up}} \sum_{j \in \Omega_{Down}} \lambda_{ij} p_i \right)$$

where p_i is the steady-state probability of being in state i , λ_{ij} is the transition rate from state i to j and Ω_{Up} and Ω_{Down} are the sets of up and down states, respectively. *MDT* is computed by $MDT = U \cdot MTBF$, where the unavailability U is computed with $U = \sum_{i \in \Omega_{Down}} p_i$.

Table 6 Model parameters for the IP network

Intensity	[time]	Description
$1/\lambda = 4$	[months]	Expected time to next PG failure inside this protection zone
$1/\lambda_{FP} = 6$	[months]	Expected time to next false positive detection failure
$1/\lambda_{SIF} = 12$	[months]	Expected time to next spontaneous isolation failure
$1/\mu_{D,M} = 20$	[minutes]	Expected manual detection time
$1/\mu_{D,A} = 1$	[minutes]	Expected automatic detection time
$1/\mu_{I,M} = 5$	[minutes]	Expected manual isolation time
$1/\mu_{I,A} = 1$	[minutes]	Expected automatic isolation time
$1/\mu_A = 5$	[minutes]	Expected time in administrative state
$1/\mu_L = 15$	[minutes]	Expected time in logistics state
$1/\mu_{FL,M} = 20$	[minutes]	Expected manual fault localisation time, i.e. without fault diagnostics from the detection devices
$1/\mu_{FL,A} = 10$	[minutes]	Expected automatic fault localisation time
$1/\mu_R = 10$	[minutes]	Expected repair time
$1/\mu_{restore} = 10$	[minutes]	Expected restoration time for discovered spontaneous isolation failure
$p_{node} = 0.1$		Probability of failure affecting a node
$p_{FN} = 0.01$		Probability of false negative detection failure
$p_{D,FP} = 0.2$		Probability of discovering a false positive in isolation phase
$p_{D,SIF} = 0.2$		Probability of discovering a spontaneous isolation failure in isolation phase
$p_{IF} = 0.1$		Probability of unsuccessful isolation
$p_{IFC} = 0.5$		Probability of unsuccessful isolation (ICT failure)

The results are presented in Fig. 13a. Four scenarios are computed:

1. current system, which is today's power grid system
2. new system,
3. new system with perfect ICT, i.e. $p_{FN} = 0$, $p_{IF} = 0$, $\lambda_{FP} = 0$, $\lambda_{SIF} = 0$, and
4. new system with a permanent isolation failure, i.e. $p_{IF} = 1$.

The *MDT* of the new system is smaller than the current system, due to the reduced event times. However, when considering the new system with imperfect ICT, the *MTBF* is reduced as well. Hence, the reduction in *MDT* comes at the expense of more frequent failures. In case of a permanent isolation failure, the *MDT* increases significantly but is still shorter than the current system, as the time in the detection phase is reduced.

MDT gives a one-sided picture of the situation, as the down states have different consequences for the system. The consequences are marked in the model with three different shapes. To incorporate this information, we use the concept of Energy Not Supplied (*ENS*). *ENS* is used in outage reports in power engineering and plays a central role in the Norwegian regulation framework [13]. As the name suggests, it indicates the amount of energy that could not be supplied due to an outage. For our example, we assume that each PG node has a constant energy consumption of 1 kWh per minute. In the pentagonal states, three nodes are down. Therefore, the *ENS* is 3 kWh per minute. The octagonal states have an *ENS* of 1 kWh per minute and the round states 0 kWh per minute.

We use a Markov reward model to obtain the instantaneous *ENS* $e(t)$, i.e. the energy that cannot be delivered at time instance t . First, state 1 is defined to be absorbing. When the system is in steady state, a down period starts in state $j \in \Omega_{\text{Down}}$ with probability $p_j(0) = \text{MTBF} \cdot \sum_{i \in \Omega_{\text{Up}}, j \in \Omega_{\text{Down}}} \lambda_{ij} p_i$. Now the instantaneous *ENS* is computed with $e(t) = \sum_{i \in \Omega_{\text{Down}}} p_i(t) \cdot e_i$, where $p_i(t)$ and e_i are the instantaneous state probability and the energy consumption per minute of state i , respectively.

Integrating $e(t)$ over time yields *Mean ENS per outage* $= \int_0^\infty e(t)$, which is plotted in Fig. 13b. The *MTBF* is the same as in Fig. 13a. Compared to *MDT*, the improvement achieved by automation is even larger in this metric because *ENS* weighs the downtime according to the consequences. However, this is not true for the case with a permanent isolation failure because the down states are all pentagonal like in the current system.

Finally, we extend *downtime-frequency curves* [19] to characterise how the total *ENS* per year of all failures in this protection zone depends on the down time. Let us denote the total *ENS* per year with $\text{ENS}_{\text{total}}$. Counting only the *ENS* of those outages that are longer than time t_0 , it becomes time dependent and is computed by:

$$\text{ENS}_{\text{total}}(t_0) = \frac{d(t_0)}{\text{MTBF}} \left(\int_{t_0}^\infty \frac{e(t)}{d(t_0)} dt + e^*(t_0) \right)$$

where $(\text{MTBF})^{-1}$ is the number of failures per year, $d(t)$ the probability that the system is down at time t , computed by $d(t) = \sum_{j \in \Omega_{\text{Down}}} p_j(t)$, and $e^*(t_0)$ is the energy not supplied up to time t_0 given that the system has not yet been restored. In order to compute $e^*(t_0)$, the Markov model is modified so there is no transition out of the subspace formed by Ω_{Down} because no complete restoration takes place before t_0 by definition. The transition rates are defined as

$$\lambda_{ij}^* = \begin{cases} \lambda_{ij} & \text{if } i, j \in \Omega_{\text{Down}} \\ 0 & \text{otherwise} \end{cases}$$

The initial state vector of the system is $\mathbf{p}^*(0) = \mathbf{p}(0)$, as before. Thus, $\mathbf{p}^*(t)$ and $e^*(t)$ are computed in the same way as explained above.

The results for $\text{ENS}_{\text{total}}(t_0)$ are shown in Fig. 14. In the current system, the relation between downtimes and $\text{ENS}_{\text{total}}$ is approximately linear during the first 50 min. In

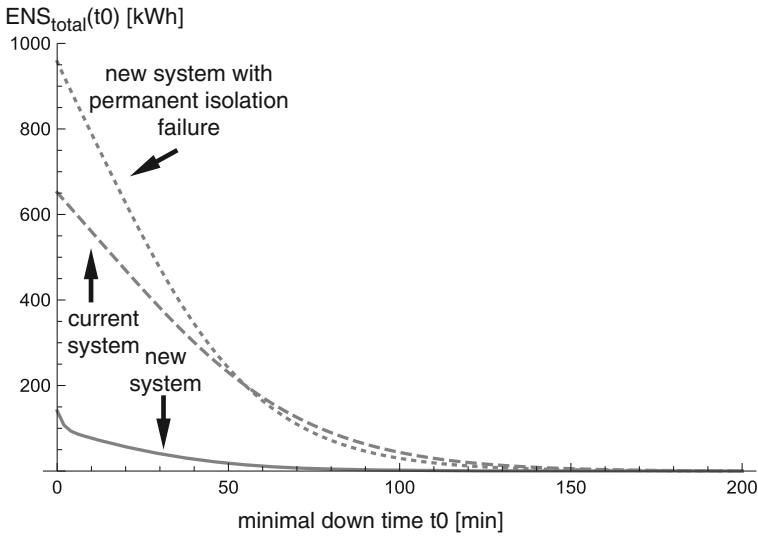


Fig. 14 ENS_{total}(t₀) of failures with downtimes > t₀

the new system, however, there is a drop in the beginning, indicating that short downtimes contribute disproportionately to ENS_{total}. The drop corresponds to *Stage I* of the model. After that, there are either no consumers without power or the system is in the restoration process with the octagonal or pentagonal states and behaves similarly to the current system but at a reduced level. In the case of a permanent isolation failure, ENS_{total}(t₀) is larger than in the current system for t₀ > 55 min, mainly because of the shorter *MTBF*. For larger t₀, this is compensated for by the effect of shorter *MDT* due to automatic detection. The results show that the automation possesses significant potential to reduce ENS_{total}. However, in case of longer failures, this may become a disadvantage.

4.4 Observations from the Example

The automation of the detection and isolation phase is introduced with the goal of reducing *MDT* and *mean ENS per failure*. However, as the new supporting ICT systems may fail as well, the failure characteristics of the system are changed. First, the *MTBF* decreases significantly, i.e. the number of failures per year increases. Second, outages are on average shorter, and short outages become an important factor when the total *ENS* per year is considered. Third, in case of a longer permanent failure in the ICT system, the consequences increase temporarily and, thereby, adversely affect of the benefit of automation.

The introduction of automation should, therefore, be accompanied by two crucial steps. First, additional training is necessary for the staff covering the new failure characteristics and failures, including the scenario of a malfunctioning ICT system [16]. Second, it is necessary to acquire the skills to maintain and quickly restore the new ICT system to assure a high dependability and thus achieve the positive effects for which the automation was originally introduced.

5 Concluding Remarks

The focus of this chapter has been the increasing complexity in digital ecosystems, which are system-of-systems of ICT infrastructures or interact with other critical infrastructures such as water distribution, transportation (e.g. Intelligent Transport Systems) and Smart Power Grid control. There is a lack of theoretical foundation to control the societal and per service dependability of ICT infrastructure in the digital ecosystem. No foundation has been established for optimisation, consolidated management and provision of this infrastructure, neither from a public regulatory perspective, nor from the perspective of groups of autonomous (commercially) co-operating providers.

More ICT-based operation support and control functions are included to manage digital ecosystems, with the objective to reduce the frequency and consequences of daily events. However, it is important to be aware of the potential side effects that might increase the frequency and consequences of critical and catastrophic failure events. The reason is that the added support enables interaction and integration of even more complex and heterogeneous systems, changes workflows in organisations and ICT-based support systems may fail.

To enhance and improve the operation and maintainability of complex digital ecosystems, new functionality is *added* and/or *moved and centralised*. Two examples are considered in this chapter: (i) Software-Defined Networking, which separates the control logic from the forwarding functionality and *moves* the logic from the distributed network elements to a virtual centralised controller, (ii) Smart Grid integrates ICT and power grids which make them more interdependent. Here, new functionality is *added* both in a distributed manner to enable observability and controllability of the components in the power grid and centralised in the control centres to implement the control.

How the changes in complexity affect the overall system dependability is less understood, contains potential vulnerabilities and poses new management challenges. This chapter emphasises the importance of being able to model ICT infrastructures. A model must describe both the structure and behaviour of the physical and logical information and network infrastructure, including the services provided. Furthermore, through the modelling phases, it should be explained how *resilience engineering* can be applied to manage the robustness and survivability of the ICT

infrastructure. This is the research focus of the research lab on *Quantitative modelling of dependability and performance*, NTNU QUAM Lab (<https://www.ntnu.edu/telematics/quam>).

Acknowledgments This work is partly funded by Telenor–NTNU collaboration project *Quality of Experience and Robustness in Telecommunications Networks*, NTNU project *The next generation control centres for Smart Grids* (<https://www.ntnu.edu/ime/smartgrids>), COST Action ACROSS (IC1304) and the research lab on *Quantitative modelling of dependability and performance*, NTNU QUAM Lab (<https://www.ntnu.edu/telematics/quam>).

References

1. Avizienis A, Laprie JC, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans Dependable Secure Comput* 1:11–33
2. Buldyrev SV, Parshani R, Paul G, Stanley HE, Havlin S (2010) Catastrophic cascade of failures in interdependent networks. *Nature* 464(7291):1025–1028
3. Ciardo G, Trivedi KS (1993) A decomposition approach for stochastic reward net models. *Perf Eval* 18:37–59
4. Cristian F, Dancey B, Dehn J (1990) Fault-tolerance in the advanced automation system. In: *Fault-tolerant computing, 1990. FTCS-20. Digest of Papers, 20th International Symposium*, pp 6–17
5. Gonzalez AJ, Helvik BE (2012) Characterization of router and link failure processes in UNINETT’s IP backbone network. *Int J Space-Based Situated Comput*
6. Haleplidis E, Pentikousis K, Denazis S, Salim JH, Meyer D, Koufopavlou O (2015) Software-defined networking (SDN): layers and architecture terminology. In: *Request for comments RFC 7426, Internet Research Task Force (IRTF)*
7. Heegaard PE, Mendiratta VB, Helvik BE (2015) Achieving dependability in software-defined networking—a perspective. In: *7th international workshop on reliable networks design and modeling (RNDM), Munich, Germany*
8. Heller M (2001) Interdependencies in civil infrastructure systems. *Bridge* 31(4) (2001)
9. Hollnagel E, Woods DD, Leveson N (2006) *Resilience engineering: concepts and precepts*. Ashgate
10. ITU-T: Recommendation Q.700: Introduction to signaling system No. 7 (1994)
11. ITU-T: Recommendation I.371: traffic control and congestion control in B-ISDN (1996)
12. Kirschen D, Bouffard F (2009) Keeping the lights on and the information flowing. *IEEE Power Energy Mag* 7(1):50–60. doi:10.1109/MPE.2008.930656
13. Kjølle G, Samdal K, Brekke K (2009) Incorporating short interruptions and time dependency of interruption costs in continuity of supply regulation. In: *CIREN, Prague, Czech Republic*, pp 1–4
14. Kreutz D, Ramos FMV, Veríssimo PJE, Rothenberg CE, Azodolmolky S, Uhlig S (2015) Software-defined networking: a comprehensive survey. *Proc IEEE* 103(1):14–76
15. Kuusela P, Norros I (2010) On/off process modeling of ip network failures. In: *International conference on dependable systems and networks (DSN), 2010 IEEE/IFIP*, pp 585–594. doi:10.1109/DSN.2010.5544427
16. Line MB (2015) Understanding information security incident management practices: a case study in the electric power industry. Ph.D. thesis, Norwegian University of Science and Technology (NTNU)
17. Longo F, Distefano S, Bruneo D, Scarpa M (2015) Dependability modeling of software defined networking. *Comput Netw* 83:280–296
18. Morris RG, Barthelemy M (2013) Interdependent networks: the fragility of control. *Scientific reports* 3. doi:10.1038/srep02764

19. Norros I, Pulkkinen U, Kilpi J (2007) Downtime-frequency curves for availability characterization. In: IEEE/IFIP dependable systems and networks (DSN), pp 398–399
20. Nunes B, Mendonca M, Nguyen XN, Obraczka K, Turletti T (2014) A survey of software-defined networking: past, present, and future of programmable networks. *Commun Surv Tutorials IEEE* 16(3):1617–1634. doi:[10.1109/SURV.2014.012214.00180](https://doi.org/10.1109/SURV.2014.012214.00180)
21. NVE (2014) Norwegian water resources and energy directorate: avbrottsstatistikk. [Outage statistics 2013]
22. Rinaldi S, Peerenboom J, Kelly T (2001) Identifying, understanding, and analyzing critical infrastructure interdependencies. *IEEE Control Syst* 21(6):11–25. doi:[10.1109/37.969131](https://doi.org/10.1109/37.969131)
23. Verbrugge S, Colle D, Demeester P, Huelsermann R, Jaeger M (2005) General availability model for multilayer transport networks. In: Proceedings 5th international workshop on design of reliable communication networks (DRCN 2005), pp 85–92 IEEE
24. Xia W, Wen Y, Foh CH, Niyato D, Xie H (2015) A survey on software-defined networking. *Commun Surv Tutorials IEEE* 17(1):27–51. doi:[10.1109/COMST.2014.2330903](https://doi.org/10.1109/COMST.2014.2330903)

30 Years of GreatSPN

Elvio Gilberto Amparore, Gianfranco Balbo, Marco Beccuti,
Susanna Donatelli and Giuliana Franceschinis

Abstract GreatSPN is a tool for the stochastic analysis of systems modeled as (stochastic) Petri nets. This chapter describes the evolution of the GreatSPN framework over its life span of 30 years, from the first stochastic Petri net analyzer implemented in Pascal, to the current, fancy, graphical interface that supports a number of different model analyzers. This chapter reviews, with the help of a manufacturing system example, how GreatSPN is currently used for an integrated qualitative and quantitative analysis of Petri net systems, ranging from symbolic model checking techniques to a stochastic analysis whose efficiency is boosted by lumpability.

1 Introduction

GreatSPN [24] is a tool that supports model-based (stochastic) analysis of discrete event dynamic systems (DEDS). It has evolved significantly over its 30 years of life and it has been used not only for the evaluation of systems, but also to support research activities, by providing an environment suitable for the development of new methods and techniques, mainly aimed at performance evaluation.

E.G. Amparore (✉) · G. Balbo · M. Beccuti · S. Donatelli
Dipartimento di Informatica, Università di Torino, C.so Svizzera 185, 10149 Turin, Italy
e-mail: amparore@di.unito.it

G. Balbo
e-mail: balbo@di.unito.it

M. Beccuti
e-mail: beccuti@di.unito.it

S. Donatelli
e-mail: donatelli@di.unito.it

G. Franceschinis
Dipartimento di Scienze e Innovazione Tecnologica, Università del Piemonte Orientale,
Viale Teresa Michel 11, 15121 Alessandria, Italy
e-mail: giuliana.franceschinis@uniupo.it

The modeling formalisms of reference in GreatSPN are Generalized Stochastic Petri nets (GSPN) [3] and its colored extension stochastic well-formed nets (SWN) [18]. SWN are based on the high-level Petri net model of well-formed net (WN). WNs have been recasted into symmetric nets (SN) in the Petri net ISO/IEC 15909-2 standard [29], and therefore SWN are sometimes also called Stochastic Symmetric Nets (SSN).

GreatSPN was conceived about 30 years ago as a tool for performance evaluation. To overcome the (at that time) existing limitations in expressing synchronization and resource acquisition, GreatSPN evolved into a tool with a more holistic approach to verification. In this approach, classical performance properties (like resource usage and throughput of transitions) and classical qualitative Petri net properties (like liveness of transition, existence of deadlocks and traps) and, more recently, probabilistic verification properties, work in a synergic manner to establish the property of interest of a DEFS. In the rest of the paper, we shall use the term “stochastic analysis” to refer to the set of analysis activities aimed at establishing the qualitative correctness of the model, using both performance and performability properties.

One of the distinctive features of GreatSPN with respect to other tools is the willingness of its development team to maintain, in the stochastic extension, the basic semantics of transition enabling and firing as well as the relevance of the graphical information. The underlying Petri net formalism is that of place/transition nets with priorities and inhibitor arcs, which have a comprehensive graphical representation of the net behavior. The idea was to try to diverge as little as possible from the underlying Petri net formalism, so as to be able to reuse all possible solution techniques available for classical (nonstochastic) Petri nets. This choice enhanced the analytical power, but certainly decreased the modeling power, since certain modeling features, like queueing policy for places and marking dependencies on arcs, have never been included.

In this chapter, we review 30 years of history of GreatSPN and discuss its current role for model-based analysis: we revisit how the graphical interfaces have evolved over the years, and we also revisit many of the advances in stochastic Petri net analysis and what is the current status in model-based stochastic analysis.

The chapter starts with a review of the GSPN formalism: its roots and its evolution (Sect. 2), followed by the history of the GreatSPN tool in Sect. 3. The rest of the paper shows the current value of GreatSPN for model-based stochastic analysis. The tool, as it is now, is presented in Sect. 4. Section 5 describes how GreatSPN3.0 supports the workflow of a model-based analysis of a target system: from model construction to validation through model checking and evaluation using stochastic model checking and standard performance evaluation techniques. A similar workflow is illustrated for the colored case (Sect. 6). The common reference example is inspired by the various flexible manufacturing system models available in the literature. The chapter ends with a literature survey of tools with similar characteristics (Sect. 7) followed by a summary of the status of GreatSPN3.0 and of its desirable future (Sect. 8).

2 From Petri Nets to GSPN

Petri nets (PN) [34] are a natural, simple, and powerful formalism aimed at the modeling of the information and control logics in systems with asynchronous and concurrent activities. In Stochastic Petri nets (SPN) [32], all the transitions are assumed to fire with a random delay that is exponentially distributed. This feature enriches the analysis of a variety of systems by computing several quantitative indices on their efficiency (performance) and reliability. Usually, this is achieved by automatically constructing a continuous-time Markov chain (CTMC) which reflects the behavior of the system and then applying the analysis methods available for this type of stochastic process.

In models of real systems, it is often the case that a change of state occurs not only because there has been a completion of an activity which takes time, but also because there has been a change of some logical conditions which may depend on the current state of the system in a rather intricate manner. These two types of events may have rather different durations, and modeling these type of systems with SPNs yields CTMCs with quite different transition rates, making the numerical analysis of the model very difficult.

Starting from the practical observations, Generalized Stochastic Petri Nets (GSPN) [3] were proposed. Immediate transitions were introduced to address the need for events that happen in a very short time (actually zero), it was also chosen that immediate transitions have priority over timed ones (the transitions that fire after a non-negligible amount of time). Priorities were introduced to simplify the analysis, by splitting markings in “vanishing” markings (states in which at least one immediate transition is enabled and where therefore the net does not spend any time) and “tangible” markings (states in which only timed transitions are enabled and where the net does spend time). Soon after their introduction, GSPNs became very popular in the performance and reliability evaluation community. The reason for this unexpected success was probably due to three quite different reasons: the simplicity of the formalism, the presence and the role of immediate transitions, and the availability (not much later than the formalism definition) of a design and analysis tool.

GSPNs are based on very few (and simple) primitive constructs that with their precise semantics make the formalism easy to learn and apply to many interesting practical problems. Indeed, researchers with considerably different backgrounds found GSPNs easy to grasp and useful for quickly drawing and analyzing complex probabilistic models that would have been otherwise difficult to construct in a reliable manner. Despite the need for more powerful formalisms for a compact representation of complex real systems, the choice of keeping the formalism simple while delegating to a different modeling language (namely Stochastic Well-Formed nets—SWNs), the burden of dealing with these possible additional complexities allowed many newcomers to become quickly acquainted with GSPNs without scaring them away with less intuitive definitions. On the other hand, researchers already familiar with the features of the basic formalism found quite interesting the possibility of using with little additional effort the more complex high-level extensions, as they

realized that this additional complexity pays off when it is actually needed by the difficulty of the problem at hand.

Immediate transitions were originally included in GSPNs to allow a simple representation of very fast activities and logical choices. The extensive use of the formalism soon made clear that the structure of the underlying untimed (autonomous) net could play an important role in allowing many more results to be derived from the model. Time scale differences captured with timed and immediate transitions were related with the concepts of visible and invisible transitions in Petri net theory. The priority of immediate over timed transitions led to the study of untimed Petri nets with different levels of priority. Drawing on these results, it became clear the danger of specifying “confused models,” and thus the difficulty of constructing “correct models” including sequences of immediate transitions. To help analysts in specifying “well behaving nets,” the concept of extended conflict set was formalized and methods were developed to find this structure at the net level [17, 37]. In the analysis of GSPN models, immediate transitions are thus “preprocessed” to construct a reduced embedded Markov chain defined on tangible markings only.

Even the simplest GSPN models that one can conceive are difficult to describe and analyze without the use of proper software tools. The development and the free distribution of such a tool to academic researchers was indeed a key factor in spreading the knowledge and the use of GSPNs within the performance and reliability research communities.

As mentioned before, the complexity of real systems often requires more powerful formalisms in order to build relatively simple models, where abstraction is the key element to understand highly intricate situations. Colored extensions of (G)SPNs have thus been proposed, allowing a more compact and parametric model representation with a more efficient analysis. Such analysis is based on the strong or exact lumpability conditions on Markov chains. In particular, Stochastic Well-Formed Nets (SWN) [18] have a structured color syntax that enables an automatic discovery of the presence of behavioral symmetries, leading directly to a reduced state space and a corresponding lumped CTMC.

3 The History of GreatSPN

The development of the Generalized Stochastic Petri Net (GSPN) formalism [2] was motivated by the modeling power of SPNs, with their effectiveness and simplicity [4]. A prototype solver [1] was initially jointly developed by members of the Computer Science Department of the University of Torino and of the Electronics Department of the Politecnico of Torino with the simple aim of overcoming the tedious and error-prone task of manually constructing the Markov chains underlying GSPN models. Starting from the insights gained from the experience of using this preliminary tool, it was decided to design and implement a complete framework for the modeling, verification, and solution of GSPN models. The first version of the framework [15], written in Pascal [30], was released in 1985 and targeted three platforms: the VAX

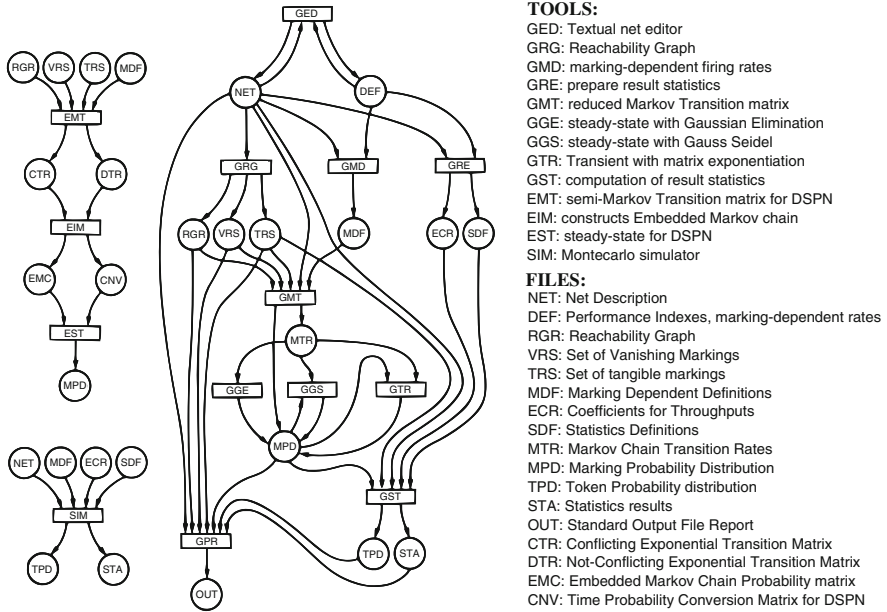


Fig. 1 The structure of the GSPN/DSPN solution toolchain in 1985

11/780 with VMS, the VAX 11/780 with BSD Unix 4.1, and Sun 1 workstation with BSD Unix 4.2. This was the first documented software package for the analysis of GSPN models [23, p. 29].

The structure of the framework was conceived as a collection of interacting tools. Figure 1 (taken from [15], p. 139) shows the structure of this original toolchain. Programs (represented as rectangles) communicate with each other using intermediate files (represented as circles). Each program is designed to solve a specific problem. The purpose of this toolchain was the generation of the reachability graph, computation of steady-state and/or transient solution of the reduced Markov chain, and computation of result statistics. GSPN and DSPN (Deterministic and Stochastic Petri Nets with deterministic transitions) models were supported.

Started in August 1986 and based on the experience of Mike Molloy’s SPAN interface [33] (which was the first graphical editor for SPN), a new GUI was written from scratch for the SunView 3.0 graphical toolkit, and its fusion with the GSPN solution toolchain became the *GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets*, namely GreatSPN 1.0. GreatSPN was the first software package developed to fully integrate within a single user-friendly tool a modeling pipeline that included, among other features, editing Petri net models graphically, inspecting their properties like invariants and bounds, calling solvers, and showing graphically the results. In 1987, version 1.3 was released with all command line programs rewritten in C to increase the portability of the framework. Several “compilation techniques” [16] were introduced in this version to improve the time and space efficiency of the tool.

With these new features, GreatSPN had the merit of joining a powerful graphical interface with a large variety of (both qualitative and quantitative) analysis methods. Models developed with GreatSPN showed for the first time the advantage of making practical the possibility for a performance analyst to first study qualitative properties of systems with methods (s)he was unfamiliar with, and for formal method experts to complete their correctness and validation studies with performance considerations. Subsequently the addition of animation (token game) and discrete-event simulation facilities [10] made GreatSPN beneficial also in management-oriented environments where the intuitive representations of real systems were more important than the powerful analysis methods that could be used for their evaluation. In 1995, the milestone release 1.7 introduced bound computation based on linear algebra techniques, as well as colored Petri nets in the form of *Stochastic Well-formed Nets* (SWN) [18]. The SWN solution method implemented in GreatSPN included both the state-space generation and consequent solution of the associated Markov chain as well as the simulation, using either colored or symbolic markings.

4 GreatSPN Now

After almost 30 years of developments, improvements, and tests, the GreatSPN framework is now a collection of many tools that support Petri net modeling and evaluation in multiple ways. Figure 2 shows a (simplified) schema of the current features of GreatSPN. Tool names are written in bold, and are grouped into logical modules. Tool functions include: numerical solutions, structural analysis, state-space exploration, and model checking for GSPN and SWN, support for Markov Decision Well-formed nets (MDWN), conversions among multiple formalisms, Monte Carlo simulation, support for DSPN definition and solution, and model composition. The graphical editor is the center of GreatSPN as it is used for drawing the models and for defining their properties. It is responsible for the invocation of various command line tools and for the visualization of the results. GreatSPN is now in the transition of replacing the old Motif-based GUI with a new interface developed in Java. For the rest of the chapter, the characteristics of the framework will be shown from the user point of view, i.e., interacting with the new Java GUI. Most of the command line tools comprised in GreatSPN can be called directly from the GUI.

The workflow of GreatSPN was conceived, back in its original design, to consist of three main phases: the user (“modeler”) draws the Petri net in a graphical way; then structural properties are computed (minimal P-/T-semiflows, place bounds, conflict sets, ...) to understand if the model is designed properly and if it can be solved using numerical methods or via simulation; finally, the user specifies the measures of interest directly on the model and calls a command line solver to compute the results. Several solvers are provided for different types of models and with different characteristics. Models are written to the disk in the net/def format, which contains the net description and the evaluation indices to be computed. There are three families

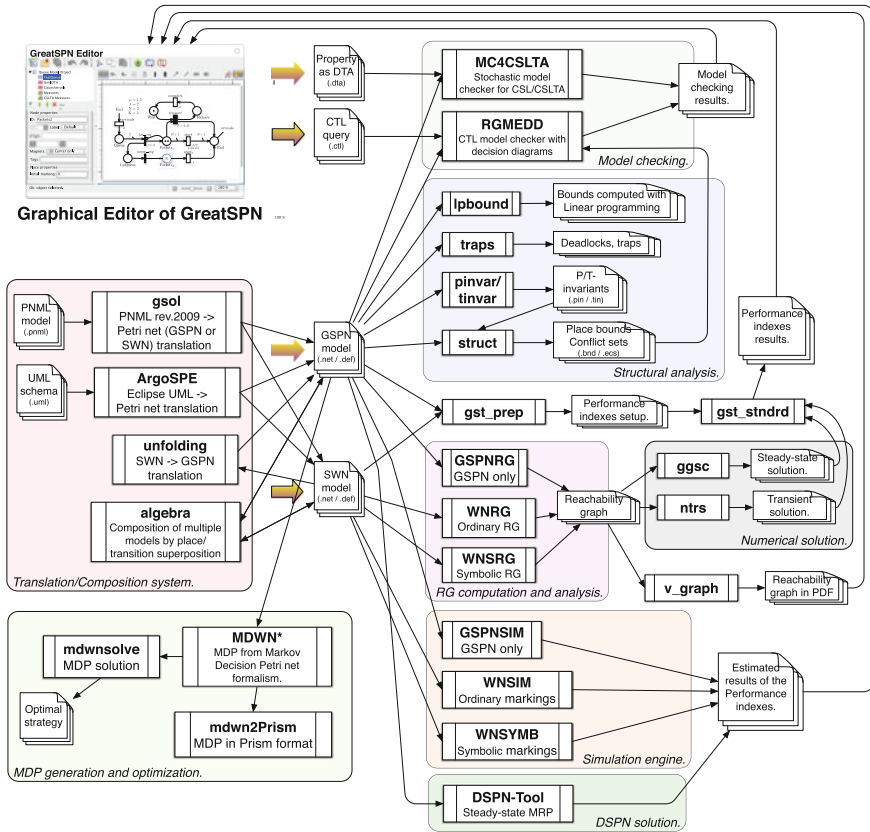


Fig. 2 The structure of the GreatSPN framework today

of models supported by GreatSPN: colored GSPNs, GSPNs with deterministic and/or general transitions, and Markov Decision Petri nets (MDWNs).

The new GUI [6] integrates a modern editing pipeline which supports the entire GreatSPN workflow consisting of the editing phase, the visual inspection of net properties, the evaluation of qualitative and quantitative properties, and visualization of the results. The rest of the chapter describes this implementation discussing a case study represented by a sufficiently complex GSPN model that illustrates the details and the new features of the framework.

A picture of the new GreatSPN GUI is shown in Fig. 3, taken while editing a Petri net model. In the upper left panel, there is the list of open files. The editor supports *multipage* files. In the current version of the editor, pages can be of three types: Petri net models, Deterministic Timed Automaton models (to be discussed later), and tables of measures. New model formalisms can be added to the editor by specifying new types of pages. The property panel is in the lower left corner. It shows the editable properties of the selected objects. It is possible to operate to more than

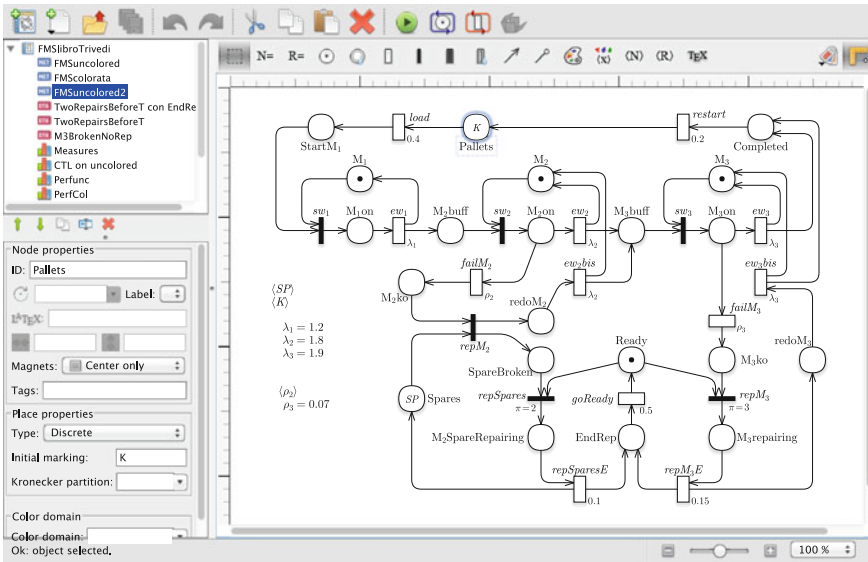


Fig. 3 The GUI with a GSPN model of an FMS with faults and repairs

one object, of the same type, at a time. The central canvas contains the editor of the selected project page, in this case a GSPN model.

Petri nets are drawn with the usual graphical notation. Transitions may be immediate (thin black bars), exponential¹ (white rectangles), or general² (black rectangles). The priority level of immediate transitions is indicated as $\pi = i$ (omitted when $i = 1$). Input and output arcs are arrows and inhibitor arcs are circle-headed arrows. Arcs may be “broken,” meaning that only the beginning and the end of the arrows are shown. The real values (or real-valued parameters) associated with transitions represent either the rate of the exponential distribution associated with timed transitions or the weight used to derive firing probabilities of immediate transitions. Definitions (constants, color classes, color variables) are drawn in textual form. Names, arc multiplicities, transition delays, weights, and priorities are all drawn as small movable labels positioned next to the corresponding Petri net elements.

Unlike the previous graphical interface, the check of the syntax of the colored definition is done while the definition is written. The editor also supports fluid places and fluid transitions (not shown in the example). Places can be partitioned into labeled groups for Kronecker-based solutions [14]. The editing process supports all the common operations of modern interactive editors, like undo/redo of every action, cut/copy/paste of objects, drag selection of objects with the mouse, single and multiple editing of selected objects, etc. Petri net models are drawn entirely using vector graphics, which allows for high-quality visualization and print of the

¹Firing times are random variables with negative exponential distributions.

²Firing times are random variables with general distributions.

net. Object labels may be drawn with an optional L^AT_EX engine. The interface is designed to avoid modal dialog windows as much as possible to streamline the use of the GUI.

5 Model-Based Analysis Through GSPN in GreatSPN3.0

Model-based analysis is supported by GreatSPN3.0 in various ways. The goal is to verify the correct behavior of the modeled system through qualitative analysis and model checking as well as to verify performance properties once it is decided that the model correctly represents the system under study (as in the case of nets automatically generated from system specifications). Once the user is confident that the model represents the system behavior, the analysis workflow concentrates on the probabilistic aspects, through stochastic model checking, and through the computation of classical performance and/or dependability evaluation indices. The GUI of the tool supports the computation and the visualization of the results for a varying set of parameter values. The analysis workflow is illustrated on a rather classical GSPN model of a flexible manufacturing system (FMS). All the figures and the graphs reported are directly produced by GreatSPN3.0, unless otherwise stated.

The GSPN model of our FMS model is depicted in Fig. 3, inside a screenshot of the GUI that has been used for its definition. The net by itself could also be printed as pdf file using the classical printing facilities of the operating system. The system includes three machines M_i (places M_i) and K pallets of parts to be worked (place Pallets). Each part is loaded and then sequentially processed by the three machines until the work is completed, the manufactured part is unloaded and the pallet goes back to place Pallets through transition *restart* waiting for a new raw part to be loaded. For each machine M_i an arriving part is taken (transition sw_i), worked (transition ew_i), and put in the input buffer of the subsequent machine or in the buffer of the completed parts. Machines M_2 and M_3 can fail. In the case of M_2 there are SP spares available, while for M_3 there are no spare parts. Spares in use can fail as well. Both spares and machines are repaired by a repairman (token in place Ready). Since there are no spares for M_3 the repairman is assigned with higher priority to M_3 . This is implemented through the priority of transition $repM_3$ which is higher than that of transition $repSpare$ s. Upon failure of M_2 (firing of transition $failM_2$), if no spare is available, the work to be done waits in place M_2ko , while if a spare is available it is taken (transition $repM_2$), the machine goes into repair, and the piece is worked (transition ew_2bis). Finally, the part is put in the input buffer of M_3 (place M_3buff) and a token goes into the machine place M_2 meaning that M_2 is available again. Upon failure of M_3 , since there is no spare available, the part is blocked until the repair ends (transition $repM_3E$) and the part is worked (transition ew_3bis). Then machine goes back to M_3 and the part goes into the buffer of completed parts.

The modeler can play the token game and observe the flow of tokens by firing the transitions. The token game can be driven by the modeler, which explicitly chooses

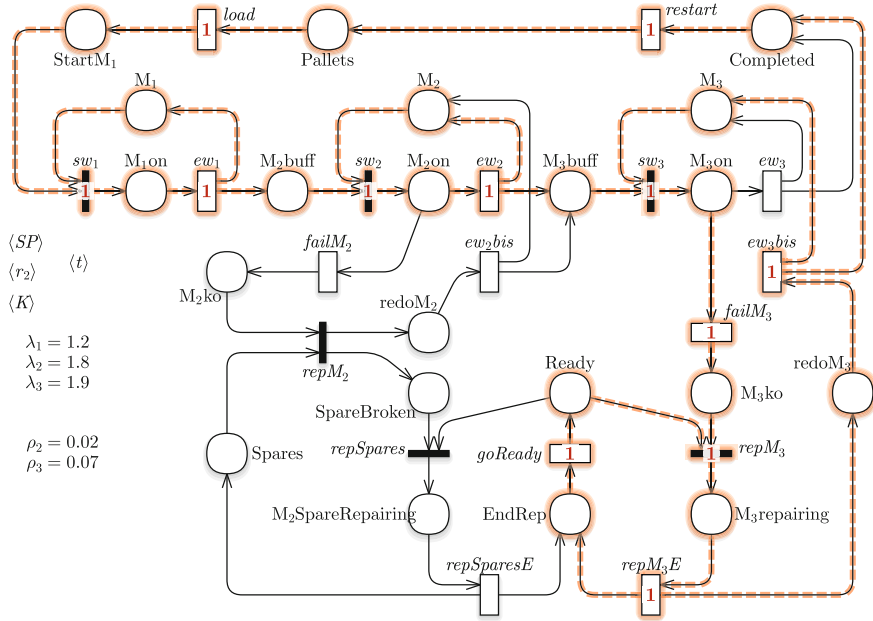


Fig. 4 Visualization of a T-semiflow for the FMS model

which transition to fire among the set of enabled transitions, or can be delegated to the tool (random mode execution).

Step 1: standard qualitative properties

The analysis by P- and T-invariants, which can be activated from the graphical interface, reveals that there are 6 minimal P-semiflows and 4 minimal T-semiflows. The corresponding P-invariants prove that all places are bounded, with bounds equal either to 1, K or SP . Figure 4 shows one of the four T-semiflows as displayed by the GUI directly on the GSPN model. Transitions in the semiflow are marked in red and their weight in the semiflow is displayed inside the transition box (all weights equal to 1 in this semiflow). The T-semiflow of the figure refers to a pallet that goes normally through machines M_1 and M_2 and then experiences a failure at M_3 . For this T-semiflow there is a firing sequence fireable in the initial marking. The T-semiflow shows a scenario in which there is the direct intervention of the repairman to complete the work.

Reachability graph generation and analysis reveal that the state space contains a single connected component, that there are no deadlocks, and that all transitions are live. Reachability graphs can also be displayed by selecting the “measure” RG (TRG for the tangible reachability graph). Figure 5, right, shows the TRG as displayed by GreatSPN, while the left part is a zoom-in of a portion of it. The feature for displaying the reachability graph is very useful for Petri net beginners and for teaching, but it

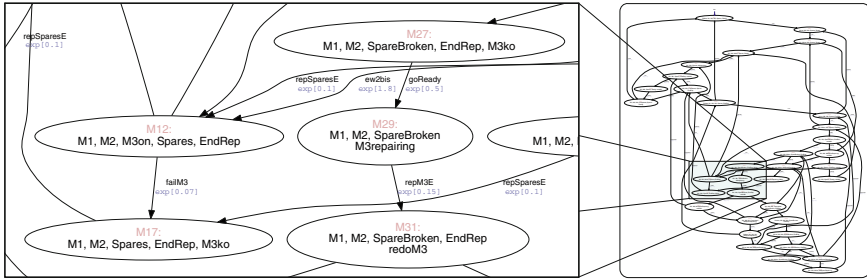


Fig. 5 TRG visualization

is usually not part of the normal workflow of model-based analysis because the size of the graph when modeling realistic systems is most of the time too large to be conveniently visualized.

Step 2: Computational Tree Logic (CTL) model checking

More sophisticated properties of the net can be investigated through the CTL [20] model checker provided by the tool. The user defines one or more “CTL measures” in the measure panel, as shown in Fig. 6. The analysis is performed for $SP = 3, K = 5$ (assigned in the top part of the window), since the GreatSPN model checker [5] is

Target model: Solver:

Template parameters:

Name: Assigned Value:

$\langle SP \rangle$ =

$\langle K \rangle$ =

Solver parameters:

Place bound: bound:

Variable order heuristic: Generate counter-examples/witnesses when possible.

Measures:

Pos:	Measure:	Result	Action
1° <input type="checkbox"/> CTL	AG ndeadlock	✓ = true	<input type="button" value="Compute"/>
2° <input type="checkbox"/> CTL	AG EF en(load)	✓ = true	<input type="button" value="Compute"/>
3° <input type="checkbox"/> CTL	EF #M2ko>0 && #Spares==0	✓ = true	<input type="button" value="Compute"/>
4° <input type="checkbox"/> CTL	AG AF en(goReady)	✓ = false	<input type="button" value="Compute"/>
5° <input type="checkbox"/> CTL	E [#Spares==SP U #Spares==(SP-1)]	✓ = true	<input type="button" value="Compute"/>
6° <input type="checkbox"/> CTL	AG #Spares==SP -> A [#Spares==SP U #Spares==(SP-1)]	✓ = false	<input type="button" value="Compute"/>
7° <input type="checkbox"/> CTL	E [#Spares==SP U #Spares==(SP-2)]	✓ = false	<input type="button" value="Compute"/>

Fig. 6 CTL model checker of GreatSPN

based on decision diagrams, much higher values of the parameters are verifiable. The CTL model checker of GreatSPN assigns a variable to each place. As usual in model checkers based on decision diagrams, a bound on each variable should be known and a variable ordering should be defined. The central part of the window (Fig. 6) is devoted to establish by which methods the bounds and the variable ordering have to be computed. The syntax of the CTL operators is the classical one, with A and E standing for “for all paths” and “there exists a path.” Operators G and F stand for “for all states in the path” and “it exists a state in the path.” The term $\#p$ means “number of token in place p ”, while condition $en(t)$ means “transition t is enabled.” The panel displays the truth value of each formula (computed in the initial state), but a log is available with more detailed information, including counterexamples or witnesses, whenever feasible. The properties listed in Fig. 6 allow to investigate, from the top of the list downwards, more and more detailed aspects of the system behavior.

The first point of the analysis is to check standard Petri net properties like absence of deadlocks and liveness of transitions. Property 1° (AG deadlock) checks that on all states of all paths reachable from the initial marking (the whole RG) it is true that the state is not a deadlock. The panel of Fig. 6 shows that the property is true and therefore the system has no deadlock. Property 2° (AG EF $en(load)$) is an example of liveness check. This property reads as “from all reachable states (AG) it is possible to find a path (EF) that enables $load$ ” ($en(load)$), which is equivalent to the classical definition of transition liveness in Petri nets. The screenshot of Fig. 6 shows that this property is true, so transition $load$ is live.

Property 3° is instead an example of a model-dependent property, aimed at investigating the use of the spares. Do we really need all the SP spares that have been included in the model? Indeed property 3° checks if there is a reachable state in which there is a failure at M_2 and no spare is available. This property reads as follows: there is a state on a path (EF) for which machine M_2 has failed ($M_2ko > 0$) and no spare is available (Spares = 0). The property is true and a system designer may be tempted to add more spares to have a more efficient production process, but of course CTL analysis is not enough to assert how convenient this addition will be. This objective should be addressed by a quantitative (stochastic) analysis.

The fourth property investigates the need for the spares to be repaired (and for the repairman to get into action). This requirement has been translated into a CTL formula (4°) that checks if it is true that on all reachable states (AG), on all paths that start from those states transition $goReady$ is enabled (AF $en(goReady)$), that is to say, repairman goes back to the ready state. This property is false, since in the RG there are loops in which machines never break down; again, only a stochastic analysis can establish how often this happens, but it is well known that, in the long run, the probability of having an execution in which machines never break down goes to zero. This is an instance of the classical fairness problem in CTL, which, when considering all paths, accounts also for the single (or the few), executions that will never happen under a fair schedule. GreatSPN3.0 is not able to check fair CTL, but the modeler can use stochastic analysis to discriminate these situations, as we shall illustrate later in this section through the stochastic property of Fig. 12. A stochastic approach might also be safer than a fair model checker: indeed in fair CTL the modeler explicitly

```

1.1: Spares(3), Ready(1), Pallets(5), M3(1), M2(1), M1(1)
    State 1.1. does not satisfy: (#Spares == 2). Start of loop.

1.2: Spares(3), Ready(1), Pallets(4), StartM1(1), M3(1), M2(1), M1(1)
    State 1.2. does not satisfy: (#Spares == 2).

1.3: Spares(3), Ready(1), Pallets(4), M3(1), M2(1), M1on(1)
    State 1.3. does not satisfy: (#Spares == 2).

1.4: Spares(3), Ready(1), Pallets(4), M2buff(1), M3(1), M2(1), M1(1)
    State 1.4. does not satisfy: (#Spares == 2).

1.5: Spares(3), Ready(1), Pallets(4), M2on(1), M3(1), M1(1)
    State 1.5. does not satisfy: (#Spares == 2).

1.6: Spares(3), Ready(1), Pallets(4), M3buff(1), M3(1), M2(1), M1(1)
    State 1.6. does not satisfy: (#Spares == 2).

1.7: Spares(3), Ready(1), Pallets(4), M3on(1), M2(1), M1(1)
    State 1.7. does not satisfy: (#Spares == 2).

1.8: Spares(3), Ready(1), Pallets(4), Completed(1), M3(1), M2(1), M1(1)
    State 1.8. does not satisfy: (#Spares == 2).

1.9: loop back to state 1.1.

```

Fig. 7 Log with a counterexample for property 6°

indicates that the model checker should consider only the paths that verify certain conditions. If the modeler identifies the wrong conditions the whole analysis process can be severely impaired.

The last three properties (5° – 7°) verify how the system uses the spares (for example, if spares are always taken one by one). 5° is an existential until property that investigates whether there is a path from the initial marking in which the spares remain untouched ($\#Spares == SP$) until the count of spares is diminished by one. This property is true, but its value is limited. Indeed it does not say that there are no other paths in which a different behavior is possible, not even on that same path there could be a different behavior. Property 5° is more informative since it checks that for all reachable states (AG) in which the number of spares is equal to SP all the paths stemming from that state keep SP spares until they get to $SP - 1$. This property is false, and the tool provides a counterexample in the log file. A portion of this log is shown in Fig. 7, which lists the states of a cyclic execution that starts with 3 spares and, passing through a number of reachable states, all with 3 spares, comes back to the initial state of the loop. Again, this is due to the presence of an infinite path in which machine M_2 never breaks down. Another line of investigation could be to check whether there exists a path, from the initial marking, in which all spares are available until two of them are taken in one step. This is formalized in property 7° , which is false.

Step 3: Classical performance evaluation

Once the user is confident that the model faithfully represents the system (at the desired level of abstraction), he/she can proceed to the standard performance index

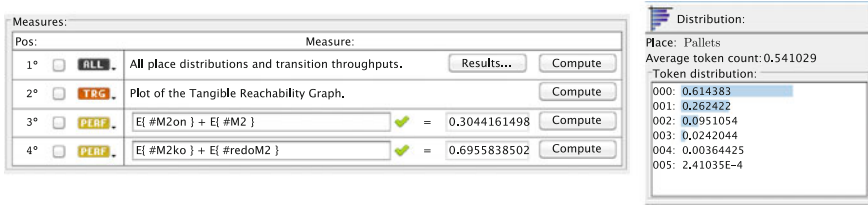


Fig. 8 GUI for performance indices of the FMS and plot of the token distribution in place Pallets

computation (throughput of transitions, average number of tokens in places and distribution of tokens in places, for various settings of the transition parameters and/or of the initial marking).

Figure 8 depicts two windows of the GreatSPN3.0 GUI. The measure panel (left) and a plot of the token distribution in place Pallets (right) show that most of the time the place is empty. The measure panel is the same panel of Fig. 6, where the target measures are performance indices instead of CTL properties. Measure ALL computes the (steady-state) token distributions for all the places and the throughput of all transitions, while specific performance indices (measure of type PERF) can be defined with an appropriate grammar. Measure 3° defines the sum of the average number of tokens in the two places that represent machine M_2 correctly working, or of the two places that represents machine M_2 broken (measure 4°).

The tool also supports dependability measures like “how long machine M_2 will survive if no repair on M_2 can take place,” for example, for a varying number of parts circulating in the system. The condition that machine M_2 cannot be repaired is implemented in the model by simply removing transition *repSpare*s (repair of a spare) so that machine M_2 and its spares are not repaired any longer. Figure 9 shows on the left the specified performance index (1°) and the range of variability of the parameters. In this case the analysis is conducted for 3 spares and a number of parts equal to 3, 4, and 5. The selected solution is the transient one at time t , where t ranges in the interval [4 . . . 40] with Step 4. Measure 1° is the probability of finding all SP



Fig. 9 Distribution of the time for consuming all spares of machine M_2 for a varying number of parts

sparers in place SpareBroken. Results are plotted on the right side of Fig. 9. The plot represents the distribution of the time needed to break all spares, for the three values of parameter K . The plot has been obtained using Excel on data exported through a specific GUI facility. Distribution of tokens in places and transition throughputs may also be shown and exported in Excel.

GreatSPN plots directly inside the GUI the distribution of the number of tokens in places (as in the case of the token distribution of Fig. 8) and the throughput of transitions directly on the net elements in the net window of the GUI.

Step4: A “less classical” approach: performance and dependability through CSL^{TA}.

Standard and nonstandard dependability/performance measures can be computed using the stochastic model checker for the CSL^{TA}[21] logic which is also part of the GreatSPN. A CSL^{TA} formula has the form $\Phi = Prob_{\bowtie\alpha}(\mathcal{A})$, where $\bowtie \in \{\leq, <, =\}$, α a probability value, and \mathcal{A} is a timed automaton with a single clock x that accepts/rejects timed paths of a GSPN. People familiar with simulation can think of the timed path of a GSPN as a simulation trace, with states, events, and time at which events happen. A formula Φ is true in a marking m if the probability of the set of the GSPN executions that starts in m and that are accepted by \mathcal{A} is $\bowtie \alpha$. Model checking algorithms for CSL^{TA} exist that return true if the property is true for the initial marking. The CSL^{TA} model checker of GreatSPN [7] computes the truth value for the initial marking as well as, for each reachable marking, the probability of the set of accepted executions stemming from that marking. The check of a CSL^{TA} property has a cost which, in the worst case, is equivalent to the cost of solving in steady state a Markov Regenerative Process.

Figure 10 (left) shows the automaton that accepts the GSPN executions in which the repairman completes two repairs before time t . The automaton has four locations, including l_0 , the initial one, and l_2 , the accepting one. The timed automaton of a CSL^{TA} formula can have more than one initial and more than one accepting locations. There is a single clock x that starts from zero and increases linearly unless it is reset to zero. A condition $x \leq t$ on an arc implies that the arc can be taken only when the value of the clock x is $\leq t$. Arcs are labeled with GSPN transition names (where *Act* stands for “any transition”) and location have an associated property that is evaluated over the GSPN marking. If the clock x is attached to the arc, taking the arc implies that x is set to zero. When the GSPN moves from marking m to marking m' for the firing of t , this move is accepted by the automaton only if there is an arc labeled t

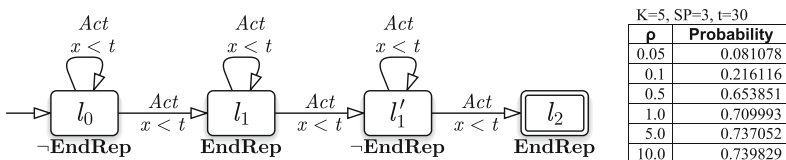


Fig. 10 Probability of two repairs, any machine, before time $t = 30$ for a varying failure rate ρ

out of the current location, the clock guard is satisfied, and the arc leads to a location whose atomic proposition is satisfied by marking m' . The only other way in which an automaton can move is because of a condition $x = \beta$ associated to an arc from the current location of the automaton. When the clock reaches value β the automaton “takes the arc” and changes location.

For a GSPN execution to be accepted by the automaton of Fig. 10, it must start in a \neg EndRep marking, then move, in one of more transition firings, to an EndRep marking, then, back to a \neg EndRep marking. The execution is finally accepted when the GSPN goes back to an EndRep marking. Whenever the clock reaches t , the path is discarded. When the atomic proposition EndRep is instantiated to the GSPN condition “#EndRep = 1” and the value of t is instantiated to 30, the CSL^{TA} model checking algorithms compute the probability of the accepted paths. This probability is listed in table reported in Fig. 10 (right). Parameter ρ is the rate of failure of both M_2 and M_3 .

Figure 11 shows an example of a timed automaton used to compute the distribution of a completion time. The automaton accepts all executions that take less than t_{\max} from the first failure at M_3 (firing of $failM_3$) to complete the part that underwent the machine failure (firing of ew_3bis). The analysis plotted in Fig. 11 for a varying value of t_{\max} is from 5 to 50.

Note that the automaton in Figure 10 accepts paths depending only on the visited markings. Since there are no specific requirements for actions associated to arcs, the automaton of Fig. 11 only accepts a path based on the transitions that occur along that path (the atomic proposition associated to location is always the clause “True”).

There is another less common use of probabilistic verification that can be very useful. As previously discussed, there are certain CTL formulas which are false due to the presence of some anomalous behavior, like infinite executions in which a machine never breaks down. Typically, these executions are not realistic. Indeed the property “AGAF $en(goReady)$ ” was shown to be false. We can define a similar property in CSL^{TA} through the automaton of Fig. 12, which accepts all paths in which transition $goReady$ fires at least once. The CSL^{TA} model checker reveals that, for all states, the set of paths accepted by the timed automaton has probability 1, clearly indicating that in a probabilistic setting the path(s) that makes the CTL formula false are negligible. This is indeed an example on the importance of having in a single tool both qualitative and probabilistic model checking.

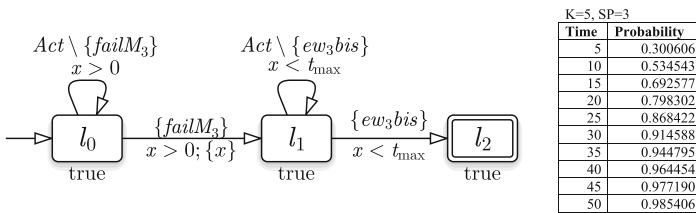
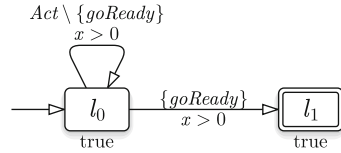


Fig. 11 Less than t_{\max} time units from first M_3 failure to completion of the part that underwent the machine failure

Fig. 12 Probability of executing *GoReady* at least once



6 Model-Based Analysis Through Colored GSPN in GreatSPN3.0

GreatSPN supports also a formalism in the class of high-level Petri nets, namely Stochastic Symmetric Nets³ (SSN) [18]. The new GreatSPN GUI supports the design of SSN models as well as the interactive simulation with the *colored token game*. It also includes the *unfolding* function that generates a GSPN model whose behavior is equivalent to that of the SSN.

Figure 13 shows a variation of the FMS model, enriched with *colors*, and expressed through the SSN formalism. The model definition includes a set of finite basic color classes, and set of place color domains which result from the Cartesian product of basic color classes and define the possible *colors* of the tokens in each place. Transition color domains define the possible *color instances* of each transition. The arc functions define the multiset of tokens withdrawn from or added to the input and output places by a given transition instance. Colors may be useful in different situations, like when there is a need to identify a specific token among a set of tokens residing in the same places (e.g., to compute first passage time distributions [11]). This differentiates the qualitative behavior of some entities (hence making the marking evolution to depend on colors) or the stochastic delays of the activities involving specific entities (hence defining color-dependent transition rates). In general colored models are more compact and can highlight symmetries in the model.

The SSN model in Fig. 13 has two color classes, C (identifiers of parts being processed by the FMS), partitioned into three static subclasses: C_1 , C_2 , and C_3 , and CM (faulty machine identifiers: m_2 and m_3 corresponding to machines M_2 and M_3) containing two static subclasses, one for each of the machine that may breakdown and be repaired. The components in subclasses C_2 and C_3 skip the processing on machine m_2 . This is modeled by the two guards associated with transitions t_0 and t_1 , namely $[(x \in C_2) \vee (x \in C_3)]$ and $[(x \in C_1)]$. The repair procedure for the two machines in this model is the same, and is based on the availability of spares. We shall consider two configurations: an asymmetric one where we have three spares for machine m_2 and only one for machine m_3 , and a symmetric one, where there are two spares for each machine. There is also a repairman who replaces the broken spares of both machines. In the model we can observe that the broken spares of machine m_3 are repaired with priority over those of machine m_2 (indicated by the labels $\pi = 3$ and $\pi = 2$ next to transitions $repSp3$ and $repSP2$).

³The formalism was first introduced with the name of Well-Formed Nets, but recently it has been replaced by the new name Symmetric Nets, better emphasizing its specific features.

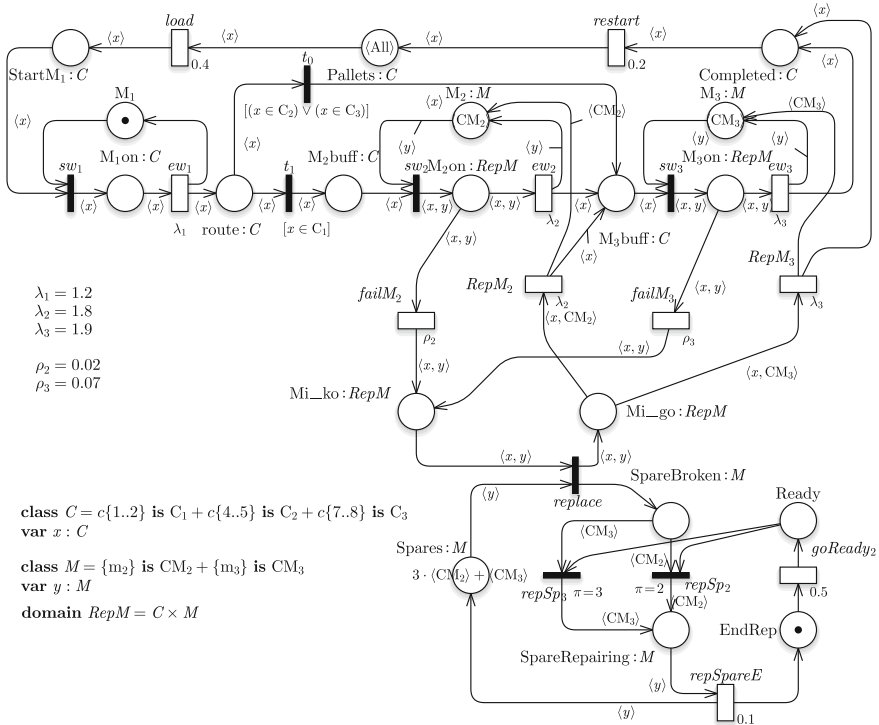


Fig. 13 A colored model of an FMS with faults and repairs

For an early check of the qualitative behavior of the colored model in the design phase, it is possible to use the colored token game feature of the GUI, which in any given marking highlights all the transitions that have at least one enabled instance. Clicking on one such transition a list of enabled instances pops up so that the modeler can choose which one to fire, leading to a new colored marking. The trace of markings reached along the interactive simulation is shown, and the user can return to one of the visited markings by clicking on it to try another path originating in that marking. During this initial phase the transition timing can be taken into account allowing the GUI to generate random delays to be associated with transitions enabled in a given marking.

The analysis of colored models can proceed in two directions: (1) through the *unfolding* of the colored net (now available with one click through the new GUI) into a (usually much larger) equivalent net without colors that can be processed using the solvers illustrated earlier, or (2) by direct analysis of the colored model using SSN specific solvers.

Table 1 shows the number of places and transitions in the unfolded version of model depicted in Fig. 13 for different sizes of color class C . The table reports also the sizes of the ordinary reachability graph (RG), and that of the symbolic reachability

Table 1 Symbolic/Ordinary RG size of the FMS colored model and size of its unfolding for different configurations Configuration with asymmetric number of spares and color independent/dependent transition rates in ew_3 and $RepM3$

C size	SRG			RG			ESRG (Tan+Van)			Unfolded	
	Tangible	Vanishing	Tangible	Vanishing	Tangible	Vanishing	ESRG	Strong	Exact	P , T	
$ C_1 , C_2 , C_3 $											
2,2+2	40,985	116,663	965,596	3,061,896	3,061,896	3,061,896	41,471	54,608	155,171	103,131	
2,2,2 deprates	159,786	484,140	965,596	3,061,896	3,061,896	3,061,896	41,471	61,057	617,185		
							41,471	76,423	617,185		
3,2+2	118,225	367,670	7,046,972	24,442,193	24,442,193	24,442,193	79,052	96,500	476,190	118,152	
3,2,2 deprates	455,230	1,491,364	7,046,972	24,442,193	24,442,193	24,442,193	79,052	96,500	1,858,652		
							79,052	110,218	1,858,652		
3,3,2 deprates	1,019,868	3,546,614	36,225,840	136,062,633	136,062,633	136,062,633	123,330	154,172	4,344,685	133,173	
							123,330	175,403	4,344,685		
Configuration with symmetric number of spares and color independent/dependent transition rates in ew_3 and $RepM3$											
C size	SRG			RG			ESRG (Tan+Van)			Unfolded	
	Tangible	Vanishing	Tangible	Vanishing	Tangible	Vanishing	ESRG	Strong	Exact	DSRG	
$ C_1 , C_2 , C_3 $											
2,2+2	44,915	131,482	1,055,314	3,433,358	3,433,358	3,433,358	43,307	52,509	173,689	46,896	
2,2,2 deprates	174,654	543,900	1,055,314	3,433,358	3,433,358	3,433,358	43,307	52,509	689,260	46,896	
							43,307	67,618	689,260	57,147	
3,2+2	129,895	413,617	7,736,126	27,382,639	27,382,639	27,382,639	81,815	101,982	532,883	89,360	
3,2,2 deprates	499,144	1,673,558	7,736,126	27,382,639	27,382,639	27,382,639	81,815	101,982	2,076,213	89,360	
							81,815	143,348	2,076,213	99,611	
3,3,2 deprates	1,116,480	3,969,892	39,705,528	151,977,288	151,977,288	151,977,288	126,798	161,896	4,843,938	139,219	
							126,798	269,711	4,843,938	162,469	

graph (SRG and ESRG). In a symbolic RG, multiple markings are lumped together into a single symbolic marking. SRG and ESRG have two different criteria for this marking aggregation.

GreatSPN can generate both the *ordinary* RG, equivalent to the RG of the unfolded model, and a more compact *symbolic* RG for any SN model. The latter exploits model symmetries and aggregates equivalent states. A CTMC can be derived both from the RG and from the SRG, on which both transient and steady-state analyses can be performed. Extended and Dynamic SRG [9] (ESRG and DSRG) are also available, to efficiently deal with partially symmetric systems. Finally, a simulator allows to compute estimates of steady-state performance measures (with confidence intervals). The simulator supports both the plain-colored marking representation and the symbolic one (which can improve the efficiency of future event list handling [12, 22]).

In Table 1 it is possible to compare the size of the RG, SRG, ESRG, and DSRG for the test cases. The number of states in the RG is derived directly from the information contained in the SRG, but it can also be derived by direct generation from the model, at the price of higher cost in both time and space, or from the unfolded model. The state-space reduction becomes relevant as the cardinality of class C grows. The size of the SRG is 17 % of the RG size for the case $|C1| = |C2| = |C3| = 2$, 6 % for the case $|C1| = 3, |C2| = |C3| = 2$, and 3 % for the case $|C1| = |C2| = 3, |C3| = 2$. Observe that a coarser partition into static subclasses gives better results in terms of state-space reduction. Indeed when transition rates do not depend on the color, the two static subclasses $C2$ and $C3$ can be merged and the SRG shrinks further: in the cases $|C1| = 2, |C2| = 4$ and $|C1| = 3, |C2| = 4$, where two static subclasses of cardinality two have been merged, the SRG size are, respectively, 4 and 1.5 % of the corresponding RG.

In systems where there are (partial) symmetries which cannot be exploited by SRG, it is still possible to take advantage of these symmetries by means of two solvers: The Extended SRG and the Dynamic SRG. The FMS model of Fig. 13 has three static subclasses. The elements in the first subclass are routed to machine m_2 after leaving m_1 , while the others are routed to m_3 . The subclasses C_2 and C_3 are needed only in some configuration where the rates of certain transitions (in our example ew_3 and $RepM3$) are different for elements in classes C_2 and elements in the other classes. By executing the ESRG module on the FMS model the algorithm automatically detects and exploits partial symmetries. The generation of the CTMC according to the ESRG method comprises two steps. First, a graph is built that overaggregates the states. Then a refinement step derives a lumped CTMC based either on *strong lumpability* or *exact lumpability* criteria [9]. The choice of the type of refinement depends on the performance indices the user wants to compute. Exact lumpability allows to retrieve the probability of detailed states, because it ensures equiprobability of the states in the same aggregate. In Table 1 the number of states generated with the ESRG algorithm are shown for some configurations. In particular observe that the size of the structure generated in the first phase of the ESRG derivation is much smaller than the size of the SRG for the same model (the number of states of the ESRG structure includes both tangible and vanishing markings). However, the refinement

with the exact lumpability condition results in a size that is close at the SRG one. When the refinement is performed using the strong lumpability condition, the size of the refined lumped CTMC is only slightly larger than that unrefined one. In the strong lumpability case however only some color-dependent performance indices can be computed.

Another possibility is to apply the DSRG solver. This aggregates the state space yielding a lumped CTMC satisfying the *exact lumpability* condition. The model specification in this case must be completely symmetric (no static subclasses partition of color classes, no guards, symmetric initial marking) while the asymmetries can be expressed in a separate file where it is possible to indicate, for each transition, restrictions on the colors that can be bound to each transition variable. In Table 1 the DSRG size for the model with equal number of spares for both machines and with or without color-dependent transition rates is shown. The type of lumpability condition makes it possible to compute color-dependent performance indices based on the information contained in the DSRG structure.

Measures can be defined for colored nets in the same way as for the uncolored ones: their definition can be independent of the color (average number of tokens in a place regardless of their color, or overall throughput of a transition corresponding to the sum of the throughput of any instance of that transition) or be color dependent. An interesting feature of the SRG is that, despite the relevant state-space aggregation, it allows one to derive the same performance indices that could be computed on the much larger RG. The same is true for the ESRG with exact lumpability refinement and for the DSRG (which also ensures exact lumpability). The model checking facilities of GreatSPN currently require that colored models have to be first unfolded for the analysis to be performed.

Table 2 shows some measures of interest computed using the GreatSPN solvers exploiting the model (partial) symmetries. These measures include system throughput (partitioned on the static subclasses of C), machines utilization, and probability for machines m_2 and m_3 to be unavailable due to a breakdown. Applying Little's formula we also derive the average time spent in m_2 queue (obtained as the ratio between the average number of customers in queue and the throughput of machine m_2). The measures are shown for the configurations (a) 2,2,2, (b) 3,2,2, and (c) 3,3,2. Each of these three configurations are tested in four different scenarios, that have/have not the same number of spares for each machine, as well as in case of uniform or color-dependent rates of the ew_3 and $RepM3$ transitions.

The analysis of the model for an increasing number of elements in class C is limited by the state-space size that grows considerably despite the application of techniques that able to exploit the model behavioral symmetries. It is however possible to estimate the measures of interest through the SSN simulator.

Table 3 reports system throughput values, utilization of machine m_1 , and probability of the repairman being in waiting status ($\#Ready == 1$, i.e., one token in place *Ready*) or working ($\#SpareRepairing == 1$, i.e., one token in place *SpareRepairing*). Each of these four measures are computed for six different sizes of the color classes, reported as triplets in the first column. The simulator can work

Table 2 Performance indices of the FMS colored model

Throughput load				Utilization						Pr(Down)		T queue
C1	C2	C3		M1	M2	M3	C1	C2	C3	M2	M3	M2
Different number of spares, uniform ew_3 rate												
0.1906	0.2024	0.2024	0.1588	0.1686	0.1686	0.1059	0.1003	0.1065	0.1065	3.4e-7	0.0418	0.0310
0.2768	0.1963	0.1963	0.2306	0.1636	0.1636	0.1537	0.1457	0.1033	0.1033	2.1e-6	0.0538	0.0640
0.2678	0.2845	0.1896	0.2232	0.2371	0.1580	0.1488	0.1839	0.1952	0.1952	2.1e-6	0.0658	0.0624
Different number of spares, color-dependent ew_3 rate												
0.1899	0.2005	0.2016	0.1583	0.1671	0.1680	0.1043	0.0999	0.1179	0.1061	3.5e-7	0.0444	0.0310
0.2757	0.1945	0.1955	0.2298	0.1621	0.1629	0.1532	0.1451	0.1144	0.1029	2.1e-6	0.0565	0.0639
0.2661	0.2812	0.1884	0.2218	0.2343	0.1570	0.1478	0.1401	0.1654	0.0992	2.2e-6	0.0701	0.0622
Equal number of spares, uniform ew_3 rate												
0.1953	0.2075	0.2075	0.1628	0.1729	0.1729	0.1085	0.1028	0.1092	0.1092	2.8e-5	0.0092	0.0314
0.2853	0.2025	0.2025	0.2378	0.1688	0.1688	0.1521	0.1502	0.1066	0.1066	1.1e-4	0.0136	0.0652
0.2776	0.2951	0.1967	0.2313	0.2459	0.1639	0.1542	0.1461	0.1553	0.1035	1.3e-4	0.0188	0.0638
Equal number of spares, color-dependent ew_3 rate												
0.1949	0.2059	0.2070	0.1624	0.1716	0.1725	0.1083	0.1026	0.1211	0.1089	2.9e-5	0.0101	0.0313
0.2846	0.2009	0.2020	0.2372	0.1674	0.1683	0.1581	0.1498	0.1182	0.1063	1.2e-4	0.0148	0.0651
0.2764	0.2923	0.1959	0.2303	0.2436	0.1632	0.1536	0.1455	0.1719	0.1031	1.3e-4	0.0208	0.0637

Results grouped in set of three lines for workloads 2,2,2, 3,2,2 and 3,3,2

Table 3 Performance indices obtained with the GreatSPN SSN simulator (confidence level 95 % accuracy 2 %)

Color class size	Performance indices	Point estimate	Confidence interval	Time (s)	Average event list size
3,3,2	X(load)	0.7355	0.7306, 0.7406	31	4.3783
	U(M1)	0.6134	0.6082, 0.6186		
	E(#Ready)	0.6398	0.6273, 0.6524		
	E(#SpareRep)	0.3003	0.2888, 0.3116		
3,8,2	X(load)	0.9683	0.9580, 0.9788	165	4.9284
	U(M1)	0.8108	0.8017, 0.8200		
	E(#Ready)	0.5509	0.5351, 0.5667		
	E(#SpareRep)	0.3751	0.3606, 0.3895		
3,8,8	X(load)	1.0794	1.0671, 1.0920	172	5.1236
	U(M1)	0.8970	0.8879, 0.9064		
	E(#Ready)	0.5006	0.4836, 0.5177		
	E(#SpareRep)	0.4158	0.4001, 0.4312		
5,8,8	X(load)	1.0725	1.0623, 1.0850	180	5.2520
	U(M1)	0.9012	0.8923, 0.9103		
	E(#Ready)	0.4811	0.4659, 0.4964		
	E(#SpareRep)	0.4304	0.4167, 0.4439		
8,8,8	X(load)	1.0957	1.0827, 1.1089	283	5.4395
	U(M1)	0.9219	0.9129, 0.9311		
	E(#Ready)	0.4765	0.4582, 0.4950		
	E(#SpareRep)	0.4340	0.4174, 0.45031		
10,10,10	X(load)	1.1096	1.0975, 1.1218	291	5.4807
	U(M1)	0.9226	0.9142, 0.9310		
	E(#Ready)	0.4383	0.4217, 0.4550		
	E(#SpareRep)	0.4687	0.4534, 0.4838		

with the same symbolic marking representation used for the SRG, so that the future event list size (shown in the last column of Table 3) does not grow significantly when the number of colors in the color classes increases.

7 Literature Review

GreatSPN3.0 has many features that can find in similar software packages for the analysis of Petri net-based models. A full comparison of these tools would require a chapter on its own. In this section, we only provide a brief overview of a few other tools that share some of the features of GreatSPN3.0 with hints on similarities and differences. Two characteristics that are unique to GreatSPN and that will not be

listed explicitly as differences are the availability of a CSL^{TA} model checker and of solution techniques based on symmetries for colored Petri nets. Vice versa, some of the tools listed below support compositionality through hierarchical models: this is a feature that is not present in GreatSPN.

SPNP The software package SPNP ([28]) was developed in the 90s at Duke University, by the group of Trivedi, and it has evolved over the years to account for new research results in the field. SPNP basic formalism is that of SRNs, which incorporate several structural extensions to GSPNs such as marking dependencies (as marking dependent arc cardinalities and guards) and allow reward rates to be associated with each marking. Type of measures that can be computed are steady state, transient, cumulative transient, time-averaged, and up-to-absorption. A discrete-event simulator is available for both SRN and its non-Markovian extension. Limited support is provided for qualitative analysis of models, which is partially due to the choice of using a powerful text-based modeling language, for which a smaller number of qualitative analysis techniques are available. SRN definition was done in a C-like language, but in 2000 a Tcl/Tk-based graphical interface was added to reduce the need for the modeler to express the model in a purely textual form. SPNP graphical interface can also be used to draw and simulate fluid Petri nets.

Möbius The tool Möbius [19] is an extensible dependability, security, and performance modeling environment for studying large-scale discrete-event systems. It supports multiple model formalisms which allow the modeler to represent each part of a system in the formalism that is most appropriate for it. Among the available formalisms it supports Stochastic Activity Networks (SANs) [35] a superclass of GSPNs in which the primitive *Input* and *Output* gates allow one to specify complex transition behaviors by general functions written in a C-like language. Like GreatSPN, Möbius provides multiple solution techniques. It supports time and space efficient discrete-event simulation as well as numerical solutions based on compact MDD representation of the state space. Möbius is probably the most mature tool for Petri net (a commercial version is available as well). With respect to GreatSPN3.0 it has better features for stochastic simulation, but it has a limited support to structural and qualitative analyses.

TimeNET The software package TimeNET [38], now at version 4.3, was developed starting back in 1995 at the Technische Universität of Ilmenau, as a successor of the software DSPNexpress, which was partly inspired by GreatSPN. The main focus of TimeNET is an efficient unified solution of DSPN and GSPN nets. Steady-state and transient analysis techniques include either exact numerical solutions, approximate solutions, or simulations. Firing delays of nonexponential transitions may have an arbitrary distribution. The graphical user interface, initially developed in Motif and then rewritten in Java, supports colored stochastic Petri nets as well as Markov chains, and is designed to be extensible to graph-like modeling formalisms. TimeNET provides more support for general distribution than GreatSPN3.0, but it does not include a complete qualitative analysis as provided by the CTL model checker of GreatSPN.

Snoopy–Marcie–Charlie The University of Cottbus has developed a suite of tools for the analysis of qualitative and stochastic properties of Petri nets which shares many of the objectives of GreatSPN3.0. Snoopy [25] is a java-based graphical interface which includes token animation. Marcie [26] is a set of analysis algorithms for various forms of stochastic Petri nets. These algorithms implement CTL and CSL model checking as well as CTMC solution and stochastic simulation of the net. CTL model checker is very efficient, based on a specific class of decision diagrams called IDD which are particularly efficient for Petri nets. Marcie is a command line tool, but it can be called through Charlie [27], which is an “extensible” interface for solvers. Charlie computes standard structural and behavioral properties of Petri nets, complemented by nonsymbolic CTL and LTL model checking. Through the use of plug-in Charlie can work as interface for the command line solvers included in Marcie.

The suite of tools has been developed over numerous years and it is now very rich. It is not always easy to find the right solver to use and how to use it, so the wide choice of available solvers is not very easy to use. We believe that the Cottbus suite is facing a situation similar to that we experienced in GreatSPN before deciding to rewrite the GUI and to link from the GUI all the available solvers. The Cottbus suite puts emphasis on structural analysis and on the richness of Petri net extension the tool is able to deal with. Great attention is devoted to stochastic simulation of Petri nets of biological system. Features available in GreatSPN and not in the Cottbus suite include CSL^{TA} model checking and efficient solution of colored models.

Smart Smart [36] or *Stochastic Model checking Analyzer for Reliability and Timing* is a software package providing command line environment for logic and probabilistic analysis of complex systems. Its main input formalism are Stochastic Petri nets and both discrete-time and continuous-time Markov chains. For the analysis of logical behavior, both explicit and symbolic state-space generation techniques are available. In the new release, currently under development, all the symbolic algorithms will be based on Meddly library as in GreatSPN. For the study of stochastic and timing behavior, sparse storage, symbolic and Kronecker numerical solution approaches are available when the underlying process is a Markov chain. Discrete-event simulation is also provided.

APNNtoolbox The APNNtoolbox [8] has been developed at the University of Dortmund as an open toolset around a common exchange interface denoted as the Abstract Petri Net Notation (APNN). The tool provides support for stochastic Petri net, including support for hierarchies and for a limited form of colors. The solvers in the APNN toolbox are focused on state-space-based analysis methods, where state-space explosion is dealt with through Kronecker representation. The toolbox also includes a graphical user interface (APNNed) to draw the net and call the solvers.

QPME QPME [31] is a tool developed initially at the University of Darmstadt, and currently maintained at the University of Würzburg. It is devoted to Stochastic Petri nets with the extension of queueing places for which the tool provides discrete-event simulation.

Oris The Oris tool [13] has been developed at the University of Florence to deal with timed and stochastic Petri nets. The delay associated with transitions can either be

a nondeterministic value, between a pair of min–max boundaries, or stochastic over a finite/infinite interval, thus subsuming also classical stochastic Petri nets. The tool is equipped with a graphical interface for net specification and for the display of the analysis results. Oris is oriented at the analysis of non-Markovian system, for which it provides the most advanced solvers currently available.

8 Future Work

A future work list for GreatSPN strictly depends on what will be the research results in the performance evaluation field in the coming years, given the willingness of keeping GreatSPN always at pace with the most useful research advances. Such a list is difficult to write, but there are nevertheless a few features of the current graphical interface and associated solvers that are already planned for the (hopefully near) future. The first enhancement is to develop a CTL model checker for colored models to improve the analysis capabilities of GreatSPN3.0 already made unique by the use of the CSL^{TA} model checker. It could be interesting if such model checker could work directly on the symbolic reachability graph. Another approach to solve this problem is to unfold a colored Petri net (like a SWN) into its uncolored (GSPN) equivalent and then perform an (uncolored) model checking. Another point that deserves more attention in the tool is the definition of the color-dependent performance indices that would complement in an extremely useful manner the efficient solution techniques based on symmetries for colored Petri nets already implemented in GreatSPN3.0. Finally, compositionality of Petri net models is clearly a desired feature that would make it easier to draw complex hierarchical models by separating the logic into multiple nets, supporting both top-down and bottom-up approaches. In GreatSPN3.0 net composition can be performed through a command line program called *algebra*, yet to be integrated in the GUI, which implements a parallel operator similar to the parallel operator in process algebra (in CSP style) based on transition superposition; additionally, the *algebra* program also implements place superposition. Upgrading this feature to the level of the invocation of the other analysis capabilities of the tool through the new GUI is obviously an enhancement that we would like to develop as soon as possible.

References

1. Ajmone Marsan M, Balbo G, Ciardo G, Conte G (1984) A software tool for the automatic analysis of generalized stochastic Petri net models. In: Proceedings of the 1st international conference on modeling techniques and tools for performance analysis, INRIA, Paris, France, pp 243–258
2. Ajmone Marsan M, Conte G, Balbo G (1984) A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans Comput Syst* 2:93–122

3. Ajmone Marsan M, Balbo G, Conte G, Donatelli S, Franceschinis G (1995) Modeling with generalized stochastic Petri nets. Wiley, New York. <http://www.di.unito.it/~greatspn>
4. Ajmone Marsan M, Balbo G, Conte G (2000) The early days of GSPNs. In: Performance evaluation: origins and directions. Springer, London, pp 505–512
5. Amparore E, Beccuti M, Donatelli S (2014) (stochastic) Model checking in GreatSPN. Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 8489, LNCS, pp 354–363
6. Amparore EG (2015) Reengineering the editor of the GreatSPN framework. In: Moldt D, Rölke H, Störle H (eds) Petri nets and software engineering. International workshop, PNSE'15, Brussels, Belgium, 22–23 June 2015. Proceedings, CEUR-WS.org, CEUR workshop proceedings, vol 1372, pp 153–170
7. Amparore EG, Donatelli S (2010) MC4CSL^{TA}: an efficient model checking tool for CSL^{TA}. In: International conference on quantitative evaluation of systems. IEEE Computer Society, Los Alamitos, CA, USA, pp 153–154
8. APNNtoolbox (2015) APNNtoolbox webpage. <http://ls4-www.cs.tu-dortmund.de/APNN-TOOLBOX/>
9. Baarir S, Beccuti M, Dutheillet C, Franceschinis G, Haddad S (2011) Lumping partially symmetrical stochastic models. Perform Eval 68(1):21–44
10. Balbo G, Chiola G (1989) Stochastic Petri net simulation. In: Proceedings of the 21st conference on winter simulation. ACM, New York, NY, USA, WSC '89, pp 266–276
11. Balbo G, Beccuti M, De Pierro M, Franceschini G (2011) Computing first passage time distributions in stochastic well-formed nets. SIGSOFT Softw Eng Notes 36(5):7–18
12. Beccuti M, Franceschinis G (2012) Efficient simulation of stochastic well-formed nets through symmetry exploitation. In: Proceedings—winter simulation conference
13. Bucci G, Carnevali L, Ridi L, Vicario E (2010) Oris: a tool for modeling, verification and evaluation of real-time systems. Int J Softw Tools Technol Transfer 12(5):391–403
14. Buchholz P, Ciardo G, Donatelli S, Kemper P (2000) Complexity of memory-efficient Kroncker operations with applications to the solution of Markov models. INFORMS J Comput 12(3):203–222
15. Chiola G (1985) A Software package for the analysis of Generalized Stochastic Petri Net models. In: International workshop on timed Petri Nets. IEEE Computer Society, Washington, DC, USA, pp 136–143
16. Chiola G (1988) Compiling techniques for the analysis of stochastic Petri nets. In: Proceedings of the 4th international conference on modeling techniques and tools for computer performance evaluation, AFCET, pp 11–24
17. Chiola G, Ajmone Marsan M, Balbo G, Conte G (1993) Generalized stochastic Petri nets: a definition at the net level and its implications. IEEE Trans Softw Eng 19(2):89–107
18. Chiola G, Dutheillet C, Franceschinis G, Haddad S (1993) Stochastic well-formed colored nets and symmetric modeling applications. IEEE Trans Comput 42(11):1343–1360
19. Clark G, Courtney T, Daly D, Deavours D, Derisavi S, Doyle J, Sanders W, Webster P (2001) The mobius modeling tool. In: Proceedings of the 9th international workshop on Petri nets and performance models, pp 241–250
20. Clarke EM, Emerson EA (1982) Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen D (ed) Logics of programs, workshop, Yorktown Heights, New York, May 1981. Lecture notes in computer science, vol 131. Springer, pp 52–71
21. Donatelli S, Haddad S, Sproston J (2009) Model checking timed and stochastic properties with CSL^{TA}. IEEE Trans Softw Eng 35(2):224–240
22. Gaeta R (1996) Efficient discrete-event simulation of colored Petri nets. IEEE Trans Softw Eng 22(9):629–639
23. German R (2000) Performance analysis of communication systems with non-Markovian stochastic Petri nets. Wiley, New York
24. GreatSPN (2015) GreatSPN webpage. <http://www.di.unito.it/~greatspn/index.html>
25. Heiner M, Herajy M, Liu F, Rohr C, Schwarick M (2012) Snoopy—a unifying Petri net tool. In: Application and theory of Petri nets—33rd international conference, PETRI NETS 2012, Hamburg, Germany, 25–29 June 2012. Proceedings, pp 398–407

26. Heiner M, Rohr C, Schwarick M (2013) MARCIE—model checking and reachability analysis done efficiently. In: Colom JM, Desel J (eds) Application and theory of Petri nets and concurrency—34th international conference, PETRI NETS 2013, Milan, Italy, 24–28 June 2013. Proceedings, Lecture notes in computer science, vol 7927. Springer, pp 389–399
27. Heiner M, Schwarick M, Wegener J (2015) Charlie—an extensible petri net analysis tool. In: Devillers RR, Valmari A (eds) Application and theory of Petri nets and concurrency—36th international conference, PETRI NETS 2015, Brussels, Belgium, 21–26 June 2015. Proceedings, Lecture notes in computer science, vol 9115. Springer, pp 200–211
28. Hirel C, Tuffin B, Trivedi KS (2000) Spnp: stochastic Petri nets, version 6.0. In: Proceedings of the 11th international conference on computer performance evaluation: modelling techniques and tools. Springer, London, UK, UK, TOOLS '00, pp 354–357
29. ISO/IEC (2011) Standard ISO/IEC 15909–2:2011, Systems and software engineering—high-level Petri nets. http://www.iso.org/iso/catalogue_detail?csnumber=43538. Accessed 05 Aug 2015
30. Jensen K, Wirth N (1975) PASCAL user manual and report. Springer, New York
31. Kounev S, Spinner S, Meier P (2010) Qpme 2.0—a tool for stochastic modeling and analysis using queueing Petri nets. In: Sachs K, Petrov I, Guerrero P (eds) From active data management to event-based systems and more. Lecture notes in computer science, vol 6462. Springer, Berlin, pp 293–311
32. Molloy M (1982) Performance analysis using stochastic Petri nets. *IEEE Trans Comput C-31(9):913–917*
33. Molloy MK, Riddle P (1986) The Stochastic Petri Net Analyzer system design tool for bit-mapped workstations. University of Texas at Austin, Computer Science Department
34. Petri C (1962) Kommunikation mit Automaten. PhD thesis, Schriften des Institutes fr Instrumentelle Mathematik, Bonn
35. Sanders W, Meyer J (2001) Stochastic activity networks: formal definitions and concepts. In: Brinksma E, Hermanns H, Katoen JP (eds) Lectures on formal methods and performance analysis. Lecture notes in computer science, vol 2090. Springer, Berlin, pp 315–343
36. Smart (2015) SMART webpage. <http://www.cs.ucr.edu/~ciardo/SMART>
37. Teruel E, Franceschinis G, De Pierro M (2003) Well-defined generalized stochastic Petri nets: a net-level method to specify priorities. *IEEE Trans Softw Eng 29(11):962–973*
38. Zimmermann A (2012) Modeling and evaluation of stochastic Petri nets with timenet 4.1. In: Gaujal B, Jean-Marie A, Jorswieck EA, Seuret A (eds) 6th international ICST conference on performance evaluation methodologies and tools, Cargese, Corsica, France, 9–12 October 2012. ICST/IEEE, pp 54–63

WebSPN: A Flexible Tool for the Analysis of Non-Markovian Stochastic Petri Nets

Francesco Longo, Marco Scarpa and Antonio Puliafito

Abstract This chapter describes WebSPN, a modeling tool for the analysis of non-Markovian stochastic Petri nets (NMSPNs). WebSPN is a flexible tool, providing different solution techniques to deal with the complexity of the stochastic process underlying a NMSPN. The first solution technique that was developed within WebSPN is based on a discrete-time approximation of the stochastic behavior of the marking process which enables the analysis of a broad class of NMSPN models with *preemptive repeat different (prd)*, *preemptive resume (prs)*, and *preemptive repeat identical (pri)* concurrently enabled generally distributed transitions. One of the main drawbacks of the discrete state space expansion approach is the state space explosion that limits the tractability of complex models. For such a reason, a new solution technique has been implemented in the WebSPN tool, which is based on the use of multiterminal multi-valued decision diagram (MTMDD) and Kronecker matrices to store the expanded process. Such a solution works in the continuous time domain and enables the analysis of much more complex NMSPNs with *prd* and *prs* concurrently enabled generally distributed transitions. Finally, WebSPN also implements a simulative solution, thus providing a complete and powerful tool for modeling and analysis of real complex systems.

1 Introduction

Some of the available Petri net tools (ESP [1], GreatSPN [2], SPNP [3], DSPNExpress [4], TimeNet [5], UltraSAN [6]) implemented the possibility of some non-Markovian features, thus extending the range of applicability of Petri nets (PNs).

F. Longo (✉) · M. Scarpa · A. Puliafito
Dipartimento di Ingegneria, Università degli Studi di Messina,
C.da di Dio, 98166 Messina, Italy
e-mail: flongo@unime.it

M. Scarpa
e-mail: mscarpa@unime.it

A. Puliafito
e-mail: apuliafito@unime.it

© Springer International Publishing Switzerland 2016
L. Fiondella and A. Puliafito (eds.), *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering,
DOI 10.1007/978-3-319-30599-8_10

Their main limitations include the kind and number of generally distributed (GEN) transitions and their associated preemption policy. A very limited number of simultaneously enabled GEN transitions is allowed and usually restricted to only one. Furthermore, the *preemptive repeat different* (*prd*) policy is the only one assumed. The *preemptive resume* (*prs*) and the more recently proposed *preemptive repeat identical* (*pri*) policies [7], although very powerful, are not yet implemented. The first restriction can be relaxed by the analytical results available for the analysis of *PN* with nonoverlapping *prs* general transitions [8], and there is an active research to identify the proper way to analyze *PN* with concurrently active general transitions [4, 9].

Some recently presented analytical results for the analysis of non-Markovian PNs make use of Markov regenerative theory, but, as far as we know, an automatic procedure based on this approach has not yet been developed. The only possible approach for the analysis of *PN* models with *prs* and *prd* general transitions is the phase-type (*PH*) approximation. With this technique, the marking process of the non-Markovian *SPN* is approximated by an expanded Markov chain. The main drawback of the *PH* approximation is the very large state space of the expanded Markov chain, primarily if the random firing times have a low coefficient of variation. The *pri* policy cannot be captured with the (*PH*) approximation.

In this chapter, we present a new modeling tool for the analysis of non-Markovian stochastic Petri nets that relax some of the restrictions present in currently available modeling packages. This tool, called *WebSPN*,¹ provides a discrete-time approximation of the stochastic behavior of the marking process which results in the ability to analyze (both in transient and steady states) a wider class of Petri net models with *prd*, *prs*, and *pri* concurrently enabled generally distributed transitions.

To solve the state space explosion problem, *WebSPN* also provides a more advanced solution technique based on the use of multiterminal multi-valued decision diagram (MTMDD) [10] and Kronecker algebra [11, 12]. Under the hypothesis that continuous PH (CPH) distributions are associated to the firing times of the net transitions, a new method for efficiently computing and storing the derived expanded reachability graphs is implemented. The main characteristic of such an approach is that two layers of symbolic representation are used. The lower layer stores the microstates in terms of the evolution of the expanded process within each macrostate by providing a set of Kronecker expressions involving the matrices that represent the CPH distributed events. The higher layer represents the reachability graph of the untimed system annotated with information about active events for all system states which is necessary in order to correctly manage event memory. This is done through the use of a MTMDD. This higher layer is based on one of the classical symbolic techniques used to generate the model state space, appropriately modified to collect and store the information about active but not enabled events for all system states. This information is necessary to algorithmically evaluate the expanded process on-the-fly, as we do. The continuous time-domain solution enables the analysis of much

¹The *WebSPN* tool can be downloaded from <http://webspn.unime.it> after a simple registration procedure.

more complex NMSPNs with *prd* and *prs* concurrently enabled generally distributed transitions [13–24]. In the unlikely case where an NMSPN model cannot be managed by the two techniques mentioned above, WebSPN also provides a discrete-event simulator.

In this chapter, we present the WebSPN tool, introducing the reader to the theory on which the solution techniques mentioned above are based and their algorithmic implementation. The chapter is organized as follows. Section 2 provides details about the discrete-time state space expansion approach, showing how the expanded state space approximating the stochastic process underlying an NMSPN model can be constructed. Section 3 shows the continuous-time approach, presenting the two-level symbolic representation that is used to store the expanded process in a more efficient way. Section 4 presents a simple example to highlight the main characteristics of the tool. Finally, Sect. 5 concludes the chapter with some final remarks.

2 Discrete-Time Expansion Approach

2.1 Introduction to Petri Nets and Preemption Policies

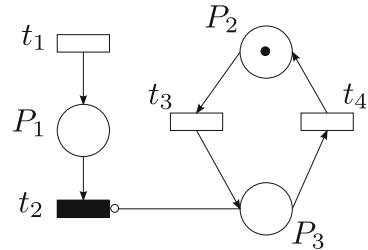
Here we give only a brief definition of timed Petri nets with generally distributed transitions and an intuitive explanation on the behavior of timed transitions in the presence of different memory policies. A timed Petri net is a tuple $PN = (\mathcal{P}, \mathcal{T}, \mathcal{G}, \mathcal{A}, \mathcal{I}, \mathcal{O}, \mathcal{H}, M_0)$ where \mathcal{P} is the set of places; \mathcal{T} is the set of transitions; \mathcal{G} is the set of random variables γ_g associated with transitions; \mathcal{A} is the set of age variables a_g associated with transitions; \mathcal{I} , \mathcal{O} and \mathcal{H} are, respectively, the set of input, output, and inhibitor functions ($\mathcal{I} \subset \mathcal{P} \times \mathcal{T}$, $\mathcal{O} \subset \mathcal{T} \times \mathcal{P}$, $\mathcal{H} \subset \mathcal{P} \times \mathcal{T}$), providing their multiplicity; and M_0 is the initial marking.²

A transition $t \in \mathcal{T}$ is enabled when the number of tokens in each input place is greater than multiplicity of input arcs and the number of the tokens in each inhibitor input place is less than the multiplicity of the inhibitor arcs. The firing of an enabled transition removes as many tokens as the multiplicity of the input arcs from the input places, and adds as many tokens as the multiplicity of the output arcs to the output places. The firing of an enabled transition, in a given marking M_i , generates another marking M_j , which is said directly reachable from M_i ($M_i \rightarrow M_j$). Starting from the initial marking M_0 , the transitive closure of \rightarrow generates the reachability graph $RG(M_0)$, which is the set of all markings reachable from M_0 .

A consistent way to introduce memory into a *SPN* is provided in [25] and extended in [8]. Each timed transition t_g is assigned a general random firing time γ_g with a cumulative distribution function $G_g(t)$. A clock, associated to each individual

²A marking M_i is a tuple, whose cardinality is $||\mathcal{P}||$, recording the number of tokens in each place.

Fig. 1 Petri net model of one server



transition, counts the time for which the transition has been enabled. An *age variable* a_g , associated to the timed transition t_g , keeps track of the clock count. A timed transition fires as soon as the memory variable a_g reaches the value of the firing time γ_g .

Let us consider an example. The Petri net in Fig. 1 models a server with exponential arrivals (transition t_1) and general service time (transition t_2). Waiting customers are represented by the tokens in place P_1 . The server is randomly preempted by higher priority jobs (transition t_3) for an exponentially distributed amount of time (transition t_4), as shown by the inhibitor arc from place P_3 to transition t_2 .

When a customer arrives at a server, a specific service requirement γ_g must be completed. The amount of computation required is sampled from the service time distribution function $F_g(t)$. The optimal case is when the server is able to complete the job before an interruption occurs. However, the server may be interrupted after having processed only a portion of the job submitted. In this case, the behavior is strongly affected by the preemption policy and the performance will depend on the strategy adopted to deal with the preempted job, as described in the following:

- The server drops the customer it was dealing with before the interruption, meaning that it loses the portion of work a_g already completed. The server then starts with a new customer possessing a new work requirement, i.e., a new sample from the same distribution is taken. Of course, the server starts serving this new customer from the beginning.
- The server goes back to the preempted customer with work requirement γ_g . The server does not lose the portion of work a_g already executed, resuming the work from the point at which it was interrupted.
- The server also returns to the same customer with work requirement γ_g . However, this server loses the portion of work a_g already completed and starts the service for this same customer from the beginning.

According to [26], the previous policies are referred to as *preemptive repeat different (prd)*, *preemptive resume (prs)*, and *preemptive repeat identical (pri)*, respectively. From the previous discussion, it is clear that the main difficulty in the analysis of stochastic Petri nets with general transitions is related to the fact that the underlying discrete state marking process is no longer a CTMC because its future evolution depends on the past history.

Based on this memory concept, at any time instant the marking and the individual memory associated with the GEN transitions of a *NMSPN* uniquely determine the future stochastic behavior of the *NMSPN*, which means that the marking process together with the memory process of the GEN transitions is a Markov process. The main idea behind our proposed discrete-time approach is to discretize the continuous memory process and the time to obtain a discrete-time Markov chain (DTMC) that approximates the stochastic behavior of the compound Markov process. The time axis is divided into equal intervals of length δ , while we use the concept of discrete phase-type distributions (*DPH*) to discretize the memory process when possible [27].

2.2 Discrete-Time Approach

Here, we present the discrete-time approach to analyze *NMSPN* that is implemented in WebSPN.

In order to explain how to approximate the stochastic behavior of continuous-time *SPNs*, the *SPN* shown in Fig. 2 is considered as an example. This *SPN* models a system that alternates between two conditions: a fully operative state (token in place P_2), where useful work can be performed, and a failure state (token in place P_1), where the system does not perform any work. The EXP transitions t_1 and t_2 describe the changes in the system state from operational to failed and vice versa. Transition t_3 models the duration of the work to be performed and is assumed to be non-exponentially distributed. In this example, the *DPH* [28] distribution, depicted in Fig. 3a, with generator $P = \{P_{ij}\}$ and initial probability $\alpha = \{1, 0, 0, \dots\}$ is used to approximate the firing time of t_3 . Based on this *DPH* structure, transition t_3 can fire when the *DPH* is either in phase 2 or in phase 4 because those phases possess arcs to absorbing phase 5 of *DPH*. Similarly, Fig. 3b depicts the *DPH* we have adopted to approximate the firing of an exponentially distributed transition, where λ is the firing rate and δ is the approximation step.

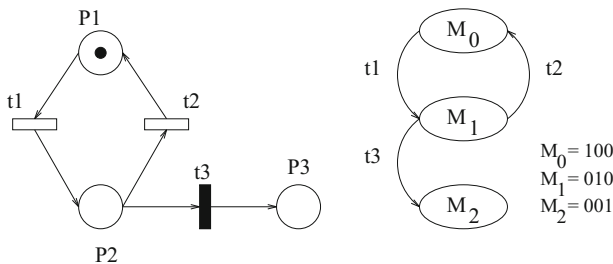


Fig. 2 *SPN* with one GEN transition

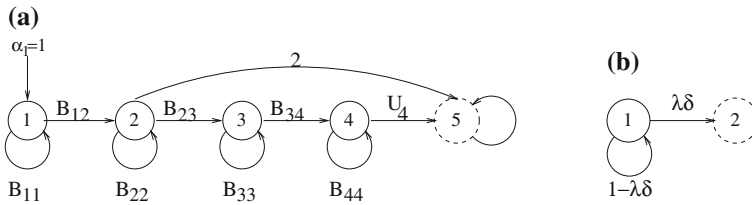


Fig. 3 DPH approximation of the firing time of t_3

2.2.1 SPN with one generally distributed *prd* transition

Let us suppose that the GEN transition t_3 is associated with a *prd* memory policy. Using DPH distributions, the state of the expanded DTMC is defined as a pair of nonnegative integers (i, u) , where i is the index of a marking ($M_i \in RG(M_0)$), and u is a phase of the DPH associated with the GEN transition. Thus, u is used to capture the “memory” that is necessary to model the GEN transitions. $u = \diamond$ denotes that the process is in a state where the general transition has no influence (i.e., it has no memory). If $1 \leq u \leq \nu$ the GEN transition is enabled. The pair (i, u) will be called a *descriptor* and identifies the state of the expanded DTMC.

Figure 4 shows the DTMC constructed to approximate the stochastic behavior of the Petri net depicted in Fig. 2. The chain is derived from the reachability graph. All states in the reachability graph are examined, and the DTMC is generated depending on the transitions enabled in each of them. Each marking in the original continuous process produces a set of states of the expanded DTMC characterized from the same index i in the descriptor (i, u) . All the states with the same marking index in its descriptor constitute a *macrostate*. Of course, the expanded process has as many macrostates as the number of markings of the continuous process. In Fig. 4, the three macrostates are outlined by ellipses with the name of the marking depicted nearby.

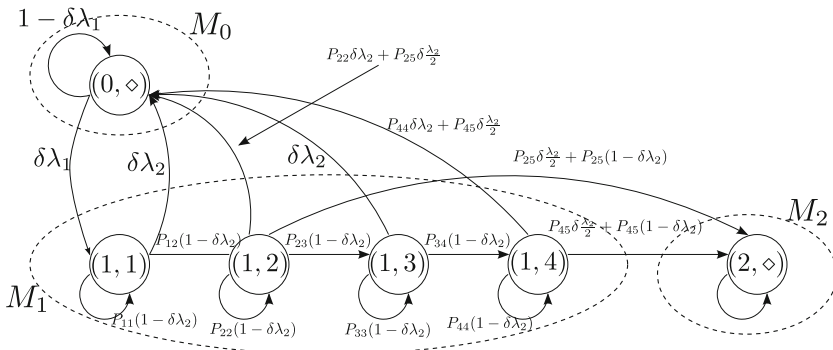


Fig. 4 DTMC approximation of the SPN in Fig. 2 with *prd* transition

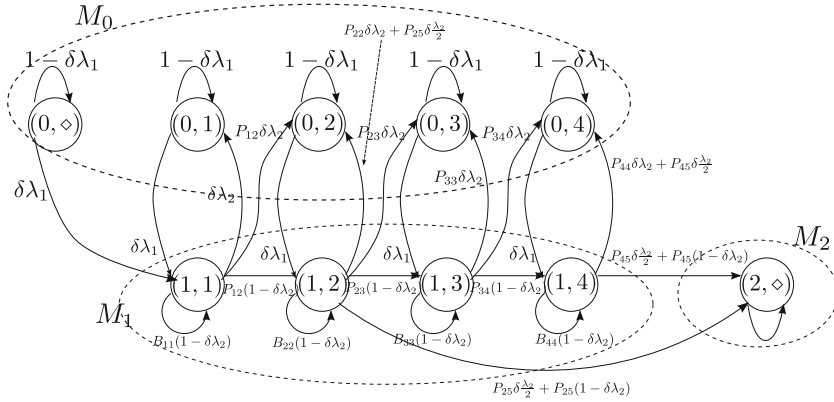


Fig. 5 DTMC approximation of the SPN with *prs* transition

2.2.2 SPN with One Generally Distributed *prs* Transition

If the GEN transition is associated with a *prs* policy, the DTMC structure must be organized in order to keep track of the amount of time the *prs* transition was enabled before being preempted. This is because the transition has to restart with the same age memory value once it becomes enabled again. For this purpose, a different expanded DTMC is needed. Figure 5 shows the DTMC that approximates the stochastic behavior of the SPN depicted in Fig. 2 when t_3 possesses a *prs* memory policy.

The only difference with respect to the *prd* case is the macrostate related to the marking M_0 . With a *prs* policy, four states with descriptors $(0, u)$, $1 \leq u \leq 4$, are added to the macrostate. These descriptors remember the value of the age memory of transition t_3 when it is disabled by the firing of the EXP transition t_2 .

2.2.3 SPN with One Generally Distributed *pri* Transition

If a *pri* policy is assumed for the GEN transition t_3 , an interrupted job must be repeated with an identical work requirement. To capture the stochastic behavior of this case, a different expanded DTMC is constructed. The stochastic behavior of an enabled *pri*-type transition is described by two continuous variables: the actual sample of the firing time and the remaining firing time, or alternatively, the actual sample of the firing time and the amount of time during which the transition has been enabled. In the proposed expansion method, the descriptor (i, u, w) , with $u \leq w$, describes the state of the process, where i indicates the marking, u the duration of time while the transition is enabled (measured in integer time slots δ), and w the sampled value (measured in integer time slots δ). The descriptor $(i, 0, w)$ indicates that the *pri* transition is disabled but has not fired so the sampled firing time w is

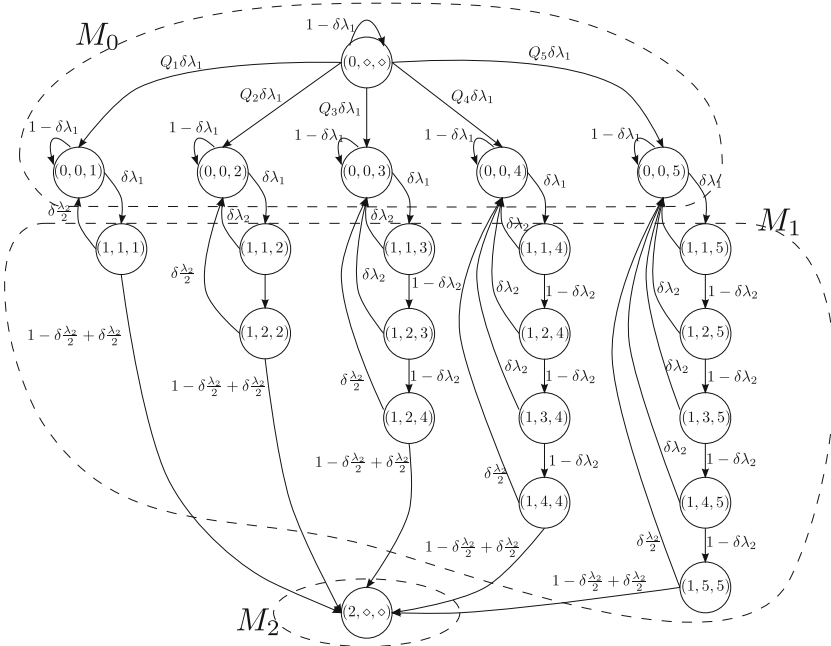


Fig. 6 DTMC approximation of the SPN with *pri* transition

maintained. After becoming enabled again, the process enters state $(i, 1, w)$. The descriptor (i, \diamond, \diamond) is used for states where the process has no memory. In other words, the marking itself completely determines the state of the process.

The evolution of the GEN transition t_k with *pri* memory policy in isolation can be described by q^k columns. The w -th column consists of w states with descriptors (i, u, w) , where $1 \leq u \leq w$. Recalling that w is the sampled firing time, when the discrete process enters a state with descriptor $(1, 1, w)$, w slots of time must pass before firing. This is exactly the time spent transitioning among the states of the column.

Figure 6 shows the DTMC that approximates the behavior of the SPN shown in Fig. 2. In this case, the macrostate corresponding to the marking M_1 is composed of the states approximating the GEN transition t_3 , as described before.

2.2.4 General Solution

In the last three subsections, we have described a method to build a DTMC to approximate the stochastic behavior of Petri nets containing only one GEN transition. Using a similar approach, this section shows how to derive the underlying DTMC for SPNs with more than one simultaneously enabled GEN transition. Similar reasoning can

be followed in the case where more EXP transitions are simultaneously enabled in the same time slot δ .

The following notation has to be introduced:

- N^D , N^S , and N^I are the number of *prd*, *prs*, and *pri* transitions in the *SPN*, respectively.
- $\mathcal{A}^D(i)$, $\mathcal{A}^S(i)$, and $\mathcal{A}^I(i)$ are the set of enabled *prd*, *prs*, and *pri* GEN transitions in marking M_i , respectively.
- $P_{i,j}^k$ is the probability of moving from phase i to phase j in the *DPH* structure of the transition t_k , which describe how a *prd* or *prs* GEN transition changes its phase.
- Q_i^k is the approximated probability that the *pri* GEN transition t_k fires in the i th δ interval.
- L_k is the number of phases in the *DPH* structure of the *prd* or *prs* transition t_k .

As discussed earlier, one variable is needed to handle transitions with *prd* and *prs* policy (to store the current phase of the expanded DTMC), and two variables to handle transitions with *pri* policy (one to store the age of the transition and the other to store the sampled value of the firing time).

When a *pri* transition is enabled, the associated random variable is sampled and the age variable is set to 1.³ If the *pri* transition is preempted in the next state the age variable is reset to 0 and the associated sampled value remains the same.

Thus a generic state of the DTMC will be $Z^r = (j, D^r, S^r, I^r, X^r)$, where

- j is the index of marking M_j of the *SPN*.
- D^r is a vector of length $||T||$, the number of the transitions in the *SPN*, storing the possible phases of a *prd* transition. In particular, its k th element (D_k^r) is the phase of transition t_k when the DTMC is in the state Z^r . The sign \diamond in the k th position indicates that the *prd* GEN transition t_k has no memory (it is not enabled).
- S^r is the same as D^r but for *prs* GEN transitions; $S_k^r = \diamond$ means that the *prs* transition t_k is not active, and thus it has no memory.
- I^r is a vector of length $||T||$. The k th element of I^r (I_k^r) is the age of the *pri* GEN transition t_k when the DTMC is in the state Z^r . Similar to the case of *prs* transitions, $I_k^r = \diamond$ indicates that transition t_k is not active.
- X^r is a vector whose k th element (X_k^r) is the sampled value of the *pri* GEN transition t_k when the DTMC is in the state Z^r .

Given a state $Z^r = (i, D^r, S^r, I^r, X^r)$ of the DTMC, it is possible to build the adjacent states both in the case when none of the enabled transitions fires in a time slot δ as well as the more complex cases when one or more firings occur. The interested reader can find such details in [29].

³Note that as time increases by δ , the total elapsed time at step i is $i * \delta$. This explains why only the index indicating the time interval needs to be recorded.

2.3 The Algorithm

The algorithm uses discretization of the continuous random variables to approximate the continuous process. The phase-type distributions, used in case of *prd* and *prs* GEN transitions, are given by the users, whereas the probabilities Q_i^k are directly computed from the cdf associated with t_k .

The main steps of the solution method implemented are

1. generation of the reachability graph (with tangible and vanishing states) and reduction of the reachability graph to tangible states only;
2. generation and analysis of the expanded DTMC;
3. evaluation of the final measures at the net level, based on the solution of the expanded DTMC.

Based on the results shown in the previous sections, given the reachability graph and the discrete phase-type distributions associated to the GEN transitions, step 2 of the approximation method is as follows:

- **Initialization Step**

Initialization creates the set of states originating in the initial marking M_0 . Moreover, it is necessary to compute the initial state probability vector on the generated states. Note that if no *pri* transition is enabled in M_0 only one state is built in this step of the algorithm. The states created are put in a list of states to expand (`list_expand`).

- **Iteration Step**

It performs

1. a state Z^r to be expanded is extracted from `list_expand`;
2. new expanded states $Z^{r'}$ are computed in case of no firing events;
3. the transition state probabilities from Z^r to $Z^{r'}$ are computed and stored;
4. all the states $Z^{r'}$, not previously created, are stored in `list_expand`;
5. sets $\overline{\mathcal{F}}_p$, with $p = 1, \dots, 2^{\|\mathcal{F}^r\|} - 1$, are computed; based on these sets, other reachable markings M_j are computed, and expanded states $Z^{r'}$ are built (these states are the states associated with the firing of one or more transitions);
6. the transition probabilities from Z^r to $Z^{r'}$ are computed and stored;
7. all states $Z^{r'}$, not previously created, are stored in `list_expand`; the state Z^r is stored in another list named `expanded`;
8. if `list_expand` is not empty the algorithm proceeds with step 1, otherwise it terminates.

Similar to [30], the system behavior is approximated by a discrete-time Markov chain (DTMC) over an expanded state space determined by the cross product of the system states (the markings of the Petri net) and the discretized values of the associated age variables. This approach is also closely related with the *DPH* expansion method proposed by Cumani in [1]. The main difference is that, in this case, the

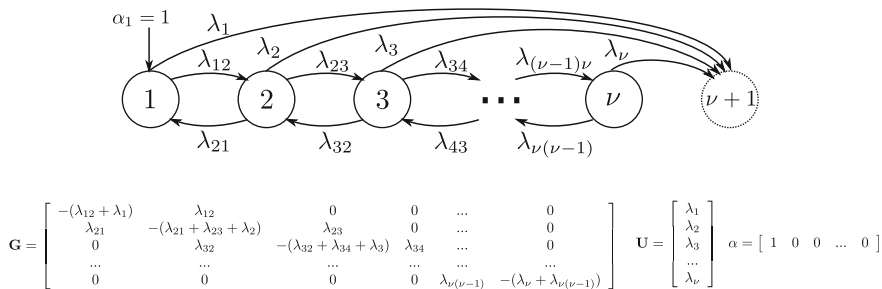


Fig. 7 Example of CPH with its matrix representation

system behavior is approximated by an expanded DTMC, while PH approximation obtains an expanded CTMC. The present approach also inherits some similarities from the supplementary variable approach [31], since the supplementary (age) variables are constrained to assume values in a discretized set.

WebSPN also implements a solution technique which is based on a CTMC approximation. In fact, the technique described in this section is subject to the state space explosion problem which strongly limits its applicability to real system models. To solve this issue, we apply symbolic representation techniques as described in the following section.

3 Two-Layer Symbolic Representation for CPH-Models

3.1 Model Definition and Basic Concepts

In this section, we assume that the firing time of each transition is described by a CPH distribution [32]. An example of a CPH distribution is reported in Fig. 7. We will refer to such a model as a CPH-model. It can be formally defined as follows.

Definition 1 A CPH-model is a tuple CPHM = (S, S^{init}, E, N, C, P), where

- S (of cardinality ||S||) is the model state space;
- S^{init} ∈ S (of cardinality ||S^{init}||) is the set of initial states;
- E (of cardinality ||E||) is the set of the possible events;
- N : S → 2^S is the next-state function, specifying the states that can be reached from a given state in a single step;
- C (of cardinality ||E||) is the set of CPH distributions associated with the model events; the CPH distribution assigned to event e is of order ν_e with representation (α_e, G_e) and α_{eν_e+1} = 0.
- P : E → {prd, prs} is a function that assigns a preemption memory policy to each event.

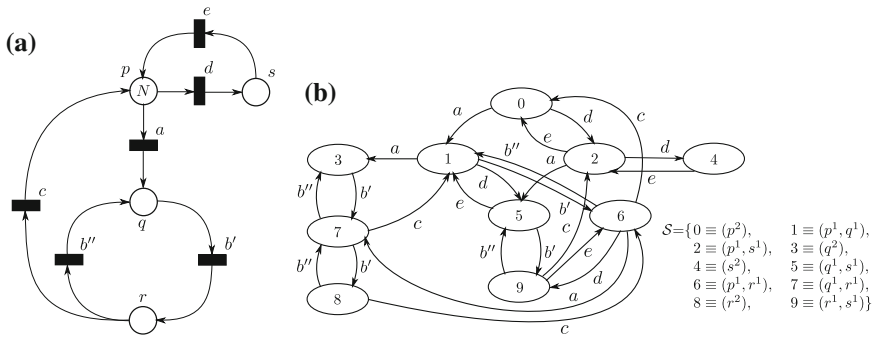


Fig. 8 Example of NMSPN model

Given the model *state space* S , $RG(S^{init})$ is the model *reachability graph* whose nodes are represented by system states and edges are labeled with system events. $RG(S^{init})$ represents the possible evolution among the states of the system. When the timing associated with each system event is considered, the reachability graph represents the stochastic process underlying the model.

A CPH-model can be described using several different modeling formalisms. The only required restriction, due to Definition 1, is that all events have timers characterized by CPH distributions defined over support $[0, b]$ (with $b > 0$) and that no immediate events are possible (distributions have no probability mass at $t = 0$). The latter hypothesis can be relaxed if more complex symbolic techniques and data structures are exploited, e.g., as done in [33]. In Fig. 8a, a *non-Markovian stochastic Petri net* (NMSPN) [34] is depicted. In the following, we will refer to it as a running example under the hypothesis that the CPH distribution depicted in Fig. 7 is associated with each of the NMSPN transitions. The reachability graph of the NMSPN is depicted in Fig. 8b, assuming that $N = 2$ tokens are present in place p in the initial marking of the net. The state space is characterized by $\|S\| = 10$ states, indexed with $s = 0, \dots, 9$. The correspondence between state indices and token distributions is also shown. The number of tokens in each place (if not null) is reported as superscript of the place name.

As mentioned above, in order for a CPH-model to be correctly represented and managed, we have to consider that the memoryless property does not hold and, in general, it is necessary to keep track of the time during which an event has been enabled while moving among states. For this purpose, an *age variable* a_e is usually assigned to each event $e \in \mathcal{E}$, behaving as a local clock. The way in which a_e is managed during state transitions determines the different *preemptive memory policies*, according to the following definition:

Definition 2 If an event $e \in \mathcal{E}$ possesses a *prd memory policy*, its corresponding age variable a_e is reset to zero each time e either fires or is disabled in a new state. In case of a *prs memory policy*, a_e is reset to zero only when e fires, otherwise it maintains its value.

The *prs* memory policy allows an event to have an associated age variable greater than zero over the process time evolution, including states in which the event itself is not enabled. Thus a state classification can be performed with respect to event $e \in \mathcal{E}$ and to the possible values of the associated a_e . Specifically, we define the *active* events in a state as follows:

Definition 3 An event e is said to be *active* in a state $s \in S$ if and only if its age variable a_e can be strictly greater than 0 when the model is in state s .

Note that if an event e is enabled in a state s_0 then it is also active in that state, given that its associated age variable a_e increases its value over time. If such an event is disabled while moving from s_0 to a new state s_1 and if it is associated with a *prs* policy, then the value of its age variable is not reset to 0, according to Definition 2, and the event will be active in s_1 even if it is not enabled. Also, note that even though Definition 3 is widely used in the context of non-Markovian models [7, 8, 35] it is slightly different from the definition of active events given in [8] because we allow an event e to be active even if its associated age variable a_e is equal to 0. According to this definition, the property for an event to be active is primarily related to the potential for the age variable to increase than the actual value it assumes. This is due to the fact that we want to classify the states independently from particular time instants during the evolution of the process. Definition 3, in fact, refers to the range of values a_e could assume during the evolution of the process.

Given a state $s \in S$, according to Definitions 2 and 3, we identify two sets of events $T_e^{(s)}$ and $T_a^{(s)}$ as the set of *enabled* events and *active but not enabled* events in state $s \in S$, respectively. Due to their definitions, $T_e^{(s)} \cap T_a^{(s)} = \emptyset$.

As a consequence of such definitions, we introduce the following theorem for an event to be active but not enabled in a system state of a CPH-model:

Theorem 1 Given a CPH-model \mathcal{M} , a state $s_0 \in S$ and an event $\bar{e} \in \mathcal{E}$ with a *prs* memory policy associated, $\bar{e} \in T_a^{(s_0)}$ iff $\bar{e} \notin T_e^{(s_0)}$ and one of the following statements holds:

1. $\exists s_1 \in S, \exists e_1 \in \mathcal{E}, s_1 \neq s_0, e_1 \neq \bar{e} \mid s_0 \in \mathcal{N}_{e_1}(s_1) \wedge \bar{e} \in T_e^{(s_1)}$
2. $\exists s_1 \in S, s_1 \neq s_0 \mid s_0 \in \mathcal{N}(s_1) \wedge \bar{e} \in T_a^{(s_1)}$

where \mathcal{N}_{e_1} is the next-state function associated with event e_1 .

The theorem states that an event $e \in \mathcal{E}$ belongs to set $T_a^{(s)}$ if it is not enabled in state $s \in S$ and if, for some path within the reachability graph $RG(S^{init})$, it is possible for its age memory variable a_e to be greater than 0 in s . In fact, if event e is enabled in state \bar{s} and it is disabled by the firing of another conflicting event \bar{e} then, according to Definition 2, it will retain its memory level after reaching state s if it has a *prs* memory policy associated (statement 1 of Theorem 2). Moreover, if event e has memory in \bar{s} and it is not enabled then it will retain its memory level in each state s reached by the process until it becomes enabled again (statement 2 of Theorem 1).

Table 1 Sets $T_e^{(s)}$ and $T_a^{(s)}$ for each of the states in the running example

s	$T_e^{(s)}$	$T_a^{(s)}$
0	$\{a, d\}$	$\{b''\}$
1	$\{a, b', d\}$	$\{b'', c\}$
2	$\{a, d, e\}$	$\{b''\}$
3	$\{b'\}$	$\{b'', c, d\}$
4	$\{e\}$	$\{a, b''\}$
5	$\{b', e\}$	$\{a, b'', c, d\}$
6	$\{a, b'', c, d\}$	$\{\}$
7	$\{b', b'', c\}$	$\{d\}$
8	$\{b'', c\}$	$\{d\}$
9	$\{b'', c, e\}$	$\{a, d\}$

Considering our running example, event a is enabled in state 1 and is disabled in state 5 by the firing of event d . Thus, event a is active in state 5 and, as a consequence, it is active also in all the states reachable from it, e.g., state 9. Table 1 reports sets $T_e^{(s)}$ and $T_a^{(s)}$ for each state $s \in \mathcal{S}$ of the running example, under the hypothesis that all the events possess a *prs* memory policy.

In the following, we will show that computation of sets $T_e^{(s)}$ and $T_a^{(s)}$ for each system state $s \in \mathcal{S}$ is a crucial requirement in order to be able to symbolically represent the expanded CTMC (ECTMC) underlying a CPH-model.

3.2 Lower Layer of Symbolical Representation: The Q matrix

In the continuous time domain, the technique that we use to account for the firing time of system events in a CPH-model consists of adding all the dynamics of the CPHs associated with model events into all the states where they are active [34, 36], producing a Markov model whose reachability graph is larger than the original. Similar to the discussion in the context of the discrete-time approach, depending on the dimension of the CPH-model's reachability graph and the order of the CPH distributions associated to system events, the state space explosion can make this explicit representation too large for a computer's memory. The use of symbolic techniques to represent the infinitesimal generator matrix associated with the expanded reachability graph can alleviate the problem. Before providing details about the symbolic approach implemented in WebSPN, we briefly recall some basic concepts on the expansion technique which are helpful to understand this discussion.

3.2.1 Continuous-Time State Space Expansion

The reachability graph $RG(S^{init})$ of a CPH-model \mathcal{M} (without considering any temporization) can be explicitly represented in the form of a square matrix \mathbf{R} (of dimension $\|S\|$) in which each entry \mathbf{R}_{ij} is the set of events whose firing changes state $s_i \in S$ into state $s_j \in S$. For example, the matrix representation of the reachability graph of the running example is as follows:

$$\mathbf{R} = \begin{bmatrix} 0 & a & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a & 0 & d & b' & 0 & 0 & 0 \\ e & 0 & 0 & 0 & d & a & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & b' & 0 & 0 \\ 0 & 0 & e & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b' \\ c & b'' & 0 & 0 & 0 & 0 & 0 & a & 0 & d \\ 0 & c & 0 & b'' & 0 & 0 & 0 & 0 & b' & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & c & b'' & 0 & 0 \\ 0 & 0 & c & 0 & 0 & b'' & e & 0 & 0 & 0 \end{bmatrix}$$

According to the state space expansion approach [34], the evolution of the stochastic process underlying a CPH-model can be represented by an ECTMC $\{\mathcal{Z}(t) : t \geq 0\}$ whose states are defined over the pairs (s, A) where $s \in S$ is a reachable system state and A is a vector containing the values of the age variables assigned to system events. In such a vector, the generic entry a_e ($1 \leq a_e \leq \nu_e$) represents the firing stage currently reached within the CPH distribution associated with event $e \in \mathcal{E}$. The basic idea is to combine the phases of the CPHs of the events that are active in a system state according to the structure of the associated CTMCs. Thus, the state space \mathcal{Z} of \mathcal{Z} possesses a cardinality much greater than $\|S\|$, in which we call the elements of \mathcal{Z} *microstates*. All the microstates with the same value of s represent the same state in \mathcal{M} with different evolution levels of the active events. For this reason, we define a *macrostate* as a set of microstates characterized by the same value of s . Of course, the number of macrostates is $\|S\|$. In WebSPN, we use a set of relations able to represent the infinitesimal generator matrix of \mathcal{Z} exploiting Kronecker algebra [37, 38].

The relations we will build are based on the following features:

- let \mathbf{Q} be the *infinitesimal generator matrix* of the ECTMC $\mathcal{Z}(t)$, then \mathbf{Q} can be represented as an $\|S\| \times \|S\|$ block matrix;
- non-null blocks in \mathbf{Q} correspond to non-null elements in \mathbf{R} ;
- the generic diagonal block \mathbf{Q}_{ii} ($i = 0, \dots, \|S\| - 1$) is a square matrix that describes the evolution of $\mathcal{Z}(t)$ inside the macrostate related to state $s_i \in S$.
- the generic off-diagonal block \mathbf{Q}_{ij} ($i, j = 0, \dots, \|S\| - 1, i \neq j$) describes the transition from the macrostate related to state $s_i \in S$ to the macrostate related to state $s_j \in S$.

In classical or explicit approaches, \mathbf{Q} would be represented and stored in memory as a sparse matrix. In WebSPN, an ad-hoc method is implemented based on two layers of symbolic representation. The approach is inspired by several works already present in the literature. However, at the best of our knowledge, WebSPN is the first tool in which a two-layer symbolic representation technique is implemented.

The first layer deals with storing microstates, i.e., the storage of blocks \mathbf{Q}_{ij} ($i, j = 0, \dots, \|S\| - 1$) through the use of Kronecker algebra operators [37–39] based on the knowledge of the system reachability graph, the CPH representation, and the memory policy of each event in the model.

Kronecker algebra is based on two main operators [40], the Kronecker product (\otimes) and Kronecker sum (\oplus). Given two rectangular matrices \mathbf{A} and \mathbf{B} of dimensions $m_1 \times m_2$ and $n_1 \times n_2$, respectively, their Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is a matrix of dimensions $m_1 n_1 \times m_2 n_2$. More specifically, it is a $m_1 \times m_2$ block matrix in which the generic block i, j has dimension $n_1 \times n_2$ and is given by $a_{i,j} \cdot \mathbf{B}$. On the other hand, if \mathbf{A} and \mathbf{B} are square matrices of dimensions $m \times m$ and $n \times n$, respectively, their Kronecker sum $\mathbf{A} \oplus \mathbf{B}$ is the matrix of dimensions $mn \times mn$ written as $\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_n + \mathbf{I}_m \otimes \mathbf{B}$, where \mathbf{I}_n and \mathbf{I}_m are the identity matrices of order n and m , respectively.

In this context, operators \oplus and \otimes assume particular interpretation. Let us consider two CTMCs \mathcal{X}_1 and \mathcal{X}_2 with infinitesimal generator matrices \mathbf{Q}_1 and \mathbf{Q}_2 . The Kronecker sum of such matrices ($\mathbf{Q}_1 \oplus \mathbf{Q}_2$) is usually interpreted as the infinitesimal generator matrix of the CTMC that models the concurrent evolution of \mathcal{X}_1 and \mathcal{X}_2 . Figure 9a shows a graphical representation of this situation. It is important to note that the states of the resulting CTMC are obtained from the combination of all the states in the two initial CTMCs.

On the other hand, let us consider a CTMC \mathcal{X}_1 with a single absorbing state and let \mathbf{U} be the column vector containing the transition rates to such a state. Moreover, let us consider a CTMC \mathcal{X}_2 and let \mathbf{P} be the matrix containing the probabilities to enter

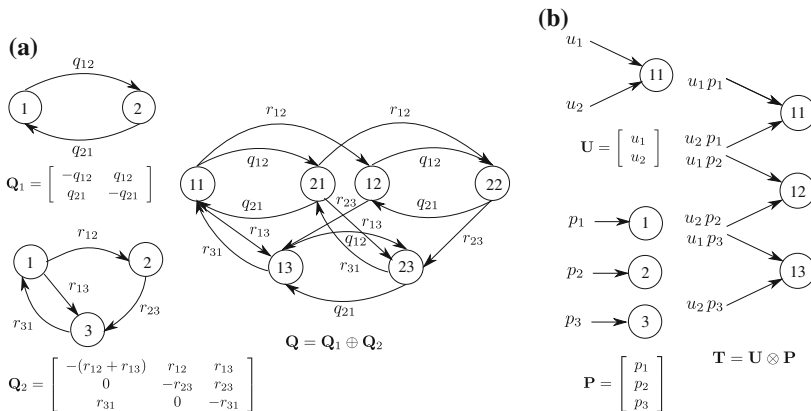


Fig. 9 Graphical representation of the semantic of operator \oplus (a) and \otimes (b)

states in \mathcal{X}_2 . The Kronecker product of \mathbf{U} and \mathbf{P} ($\mathbf{U} \otimes \mathbf{P}$) is usually interpreted as the matrix containing the transition rates to the CTMC resulting from the combination of the absorbing state of \mathcal{X}_1 with all the states of \mathcal{X}_2 . Figure 9b graphically depicts this situation.

Here, we do not provide the details of the Kronecker representation of the infinitesimal generator matrix \mathbf{Q} of a CPH-model implemented in WebSPN. The interested reader is referred to [11] for additional details.

3.3 Higher Layer of Symbolic Representation: Annotated Reachability Graph

The Kronecker representation of microstates is the first step to overcome the state space explosion problem encountered in CPH-models. However, in order to implement a complete solution technique, it is still necessary to explicitly store the system reachability graph $RG(S^{init})$ in some form, e.g., as matrix \mathbf{R} . Moreover, in order to use the previously introduced Kronecker algebra relations on-the-fly, the information about sets $T_e^{(s)}$ and $T_a^{(s)}$ for all the system states must be stored.

Sets $T_e^{(s)}$ can be easily obtained by querying the high-level model, given that the conditions for an event to be enabled only depend on the model structure. For this reason, the information about sets $T_e^{(s)}$ can be obtained on-the-fly each time it is needed by exploiting numerical solution method with little overhead. On the contrary, sets $T_a^{(s)}$ need to be computed by exploiting Theorem 1 whose statements require explorations of the reachability graph of the untimed system and they are strictly related to its structure. Computing sets $T_a^{(s)}$ on-the-fly would require exploration of the system reachability graph multiple times, incurring in non-negligible overhead. As a consequence, it is convenient to compute sets $T_a^{(s)}$ only once, annotating the model reachability graph with such an information.

In practical cases, the explicit representation of such an annotated reachability graph can become unmanageably large preventing it from fitting in a computer's memory. Thus, it is necessary to employ technique to efficiently store both matrix \mathbf{R} and the sets $T_a^{(s)}$. A higher layer of symbolic representation is needed. Before describing such a layer, we recall some fundamental symbolic representation techniques upon which our approach is based.

3.3.1 Background on Symbolic Techniques

Symbolic techniques [41] generate a compact representation of huge state spaces by exploiting a model's structure and regularity. A model has a structure when it is composed of K sub-models, for some $K \in \mathbb{N}$. In this case, a global system state can be represented as a K -tuple (s^1, \dots, s^K) , where s^k is the local state of sub-model k (having some finite size n^k).

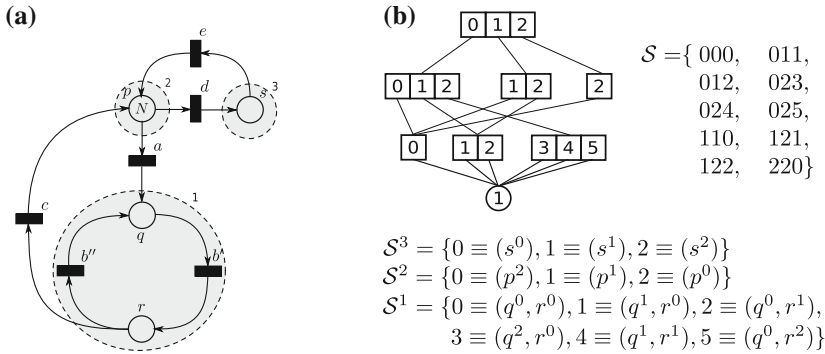


Fig. 10 Running example with a partitioning of its places (a) and the MDD associated to it in the case of $N = 2$ (b)

The use of multi-valued decision diagrams (MDDs) [42] to encode model state spaces was introduced by Miner and Ciardo in [43]. MDDs are rooted, directed, acyclic graphs associated with a finite ordered set of integer variables. When used to encode a state space, an MDD has the following structure:

- nodes are organized into $K + 1$ levels, where K is the number of sub-models;
- level K contains only a single nonterminal node, the root, whereas levels $K - 1$ through 1 contain one or more nonterminal nodes;
- a nonterminal node at level k possesses n^k arcs pointing to nodes at level $k - 1$;

A state $s = (s^1, \dots, s^K)$ belongs to S if and only if a path exists from the root node to the terminal node 1, such that at each node the arc corresponding to the local state s^k is followed.

Figure 10a shows our running example with a partitioning of its places into three sub-nets. Figure 10b shows the local state space of each sub-net with the corresponding integer encoding for each sub-marking, the global state space in terms of sub-markings, and its encoding through an MDD where $N = 2$. Only paths to terminal node 1 are shown.

In [44], and then in [45], Ciardo et al. proposed the *Saturation* algorithm for the generation of reachability graphs using MDDs. This iteration strategy improves both memory and execution-time efficiency. While the first version of *Saturation* proposed in [44] needs to know a priori the local state spaces of the sub-models, *Saturation Unbound* [45] produces an MDD representation of the final state space and a separate representation of the minimal local state spaces without such a knowledge. An efficient encoding of the reachability graph is built in the form of a set of Kronecker matrices $\mathbf{W}_{e,k}$ with $e \in \mathcal{E}$ and $k = 1, \dots, K$. $\mathbf{W}_{e,k}[i_k, j_k] = 1$ if state j_k of sub-model k is reachable from state i_k due to event e . According to such a definition, the next-state function of the model can be encoded as the incidence matrix given by the Boolean sum of Kronecker products $\sum_{e \in \mathcal{E}} \otimes_{K \geq k \geq 1} \mathbf{W}_{e,k}$. As a consequence, the matrix representation \mathbf{R} of the reachability graph of the model can be obtained by

filtering the rows and columns of the matrix corresponding to the reachable global states (our macrostates) encoded in the MDD and replacing each non-null element with the labels of the events that cause the corresponding state transition.

Returning to the running example, the Kronecker matrices associated with event a produced by Saturation Unbound are

$$\mathbf{W}_{a,1} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_{a,2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_{a,3} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Performing the Kronecker product of such matrices and filtering the rows and columns corresponding to the reachable states encoded by the MDD produce the following matrix:

$$\mathbf{R}_a = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

which represents the portion of matrix \mathbf{R} related to event a if the non-null elements are replaced by the label of such an event.

3.3.2 Encoding Sets $T_a^{(s)}$ Through MTMDDs

The combination of the MDD and $K \cdot \|\mathcal{E}\|$ Kronecker matrices produced by Saturation Unbound is a very effective way to represent the reachability graph of the model analyzed. Since $RG(S^{init})$ is the data input to algorithmically generate the infinitesimal generator matrix \mathbf{Q} of the ECTMC, we will use Saturation Unbound to generate it.⁴ The use of Saturation together with the Kronecker representation of macrostate presented in Sect. 3.2 further improves memory consumption compared to the approach presented in [46].

⁴The use of Saturation is incidental. The methodology we propose is independent of the algorithm used to generate the reachability graph of a model. We use Saturation Unbound due to its well-known efficiency.

Knowledge of the reachability graph of the untimed system as produced by Saturation is not enough to manage the matrix \mathbf{Q} on-the-fly according to the lower layer of symbolic representation. Considering that the information about the events enabled for all the system states is contained in the high-level description of the model and it can be evaluated on-the-fly when needed with a negligible overhead, the only additional information needed is the knowledge of the sets $T_a^{(s)}$. In this case, using Saturation to evaluate the reachability graph requires a further analysis step to compute this information and more advanced data structure to store it. We use MTMDDs [10] for this purpose.

The main differences with respect to MDDs are that (1) more than two terminal nodes are present in an MTMDD, and (2) such nodes can be labeled with arbitrary integer values, rather than just 0 and 1. An MTMDD can efficiently store both the system state space S and the sets $T_a^{(s)}$ of active but not enabled events for all $s \in S$ that are necessary in our approach to evaluate non-null blocks of matrix \mathbf{Q} . In fact, while an MDD is only able to encode a state space, an MTMDD is also able to associate an integer to each state. Next, the encoding of sets $T_a^{(s)}$ can be done, associating to each possible set of events a unique integer.

In WebSPN, we exploit a simple code. Let us associate to each event a unique index n such that $1 \leq n \leq \|\mathcal{E}\|$. The integer value associated to one of the possible sets $T_a^{(s)}$ is computed starting from the indexes associated with the system events that belong to it in the following way:

$$b_{\|\mathcal{E}\|} \cdot 2^{\|\mathcal{E}\|} + \dots + b_n \cdot 2^n + \dots + b_1 \cdot 2^1 + 1 = \sum_{i=1}^{\|\mathcal{E}\|} b_i 2^i + 1$$

where

$$b_i = \begin{cases} 1, & \text{if event } e_i \in T_a^{(s)} \\ 0, & \text{otherwise} \end{cases}$$

Figure 11 shows the MTMDD associated to the running example and the sets $T_a^{(s)}$ for each model state in the case where all the events in the NMSPN possess a *prs* policy.

3.3.3 Computation of Sets $T_a^{(s)}$

Using Theorem 1, it is possible to compute the sets $T_a^{(s)}$ for all system states by traversing the reachability graph along all the possible paths and storing this information in an explicit data structure as it is discovered. Such a computation can even be performed on-the-fly during state space generation if classical approaches are used given that they usually discover new states by traversing the reachability graph in a breadth-first-search (BFS) or in a depth-first-search (DFS) order. Since symbolic algorithms work in a highly localized manner, adding states out of the BFS or DFS

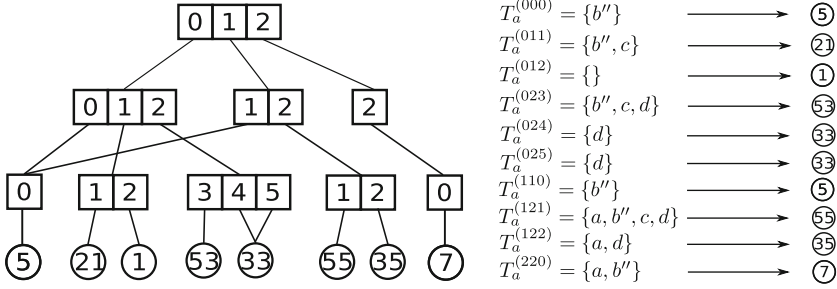


Fig. 11 The MTMDD associated to the running example in the case of $N = 2$

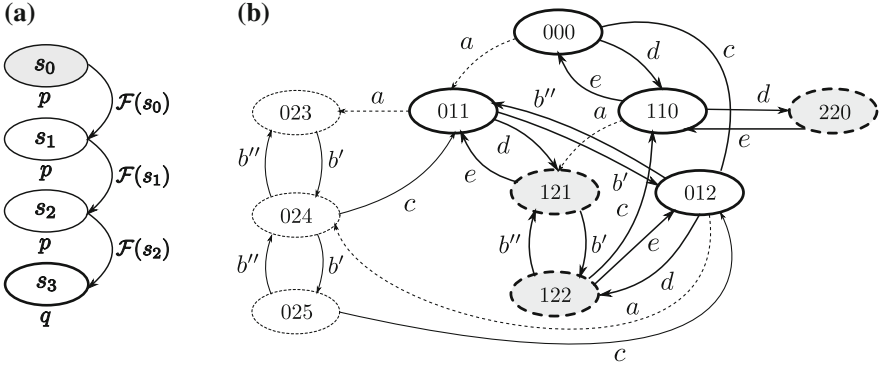


Fig. 12 A graphical example of (a) $s_0 \models E_{\mathcal{F}}[pUq]$ and (b) the algorithm of Table 2

order, the computation of all the sets $T_a^{(s)}$ by directly applying Theorem 1 becomes inefficient or, in most of the cases, impossible.

In [47], Ciardo et al. applied Saturation to efficiently evaluate computation tree logic (CTL) formulas. In [11], we demonstrated how the CTL operator EU can be used to compute the sets $T_a^{(s)}$ and we defined an algorithm that takes advantage of the efficiency of the Saturation algorithm to store such an information in a MTMDD.

Figure 12a shows an example of a path of states satisfying the formula $E_{\mathcal{F}}[pUq]$: state s_0 satisfies the formula $E_{\mathcal{F}}[pUq]$ because there exists a path starting from state s_0 to state s_3 , where logical condition p holds on s_0, s_1, s_2 , q holds on s_3 , and each state in the path is reached according to the next-state function $\mathcal{F}(s)$.

The theorem upon which our algorithm is built on is as follows:

Theorem 2 *An event $\bar{e} \in \mathcal{E}$, with an age memory policy associated, belongs to $T_a^{(s_0)}$, with $s_0 \in \mathcal{S}$, iff $s_0 \models E_{\mathcal{F}}[pUq]$ over a path at least one step in length, where p and q are the statements “ \bar{e} is not enabled” and “ \bar{e} is enabled,” respectively, and $\mathcal{F}(s) = \mathcal{N}^{-1}(s) \setminus \mathcal{N}_{\bar{e}}^{-1}(s)$.*

Theorem 2 states that it is possible to compute the states in which event \bar{e} is active but not enabled simply evaluating CTL formula $E_{\mathcal{F}}[pUq]$ over the system state space considering condition p as “ \bar{e} is not enabled” and condition q as “ \bar{e} is enabled.” Moreover, the next-state function $\mathcal{F}(s)$ considered is the inverse of the system next-state function $\mathcal{N}(s)$ but dropping the paths due to event \bar{e} . Note that Theorem 1 is not applicable if BFS or DFS order is not used to explore symbolic data structures of the reachability graph. However, Theorem 2 is applicable both when traditional and symbolic data structures and algorithms are exploited even when BFS and DFS are not used. Thus, the approach takes advantage of the efficiency of symbolic approaches in the field of state space expansion, representing one of the principal contributions implemented in the WebSPN tool.

3.3.4 The Algorithm

The traditional computation of the set of states satisfying $E[pUq]$ uses a least fixed point algorithm that starts with the set Q of states satisfying q and iteratively adds all the states that reach them on paths where property p holds. The set of states in which property p holds is called \mathcal{P} . In [47], Ciardo et al. proposed a new approach to compute the EU operator based on Saturation called EU_{sat} . It can be easily demonstrated [11] that the set of all system states in which event \bar{e} is active but not enabled can be obtained by applying just one forward Saturation step to the set of states in which event \bar{e} is enabled, neglecting \bar{e} itself. This behavior offers great advantages in terms of execution time and memory occupation compared to traditional explicit expansion approaches.

Figure 12b shows the computation of the set of states in which event a is active in our running example. States in double non-dashed line have transition a enabled and are used as the starting point of the computation. States in double dashed line are obtained through a forward Saturation step, performed without considering the firing of a as a possible event. Transition a is active but not enabled in these states. States in normal dashed line belong to the state space, but transition a is neither enabled nor active in them. These states cannot be obtained through Saturation because we ignore event a .

From Theorem 2 and the considerations above, it is possible to state the algorithm presented in Table 2 in the form of pseudocode.

The algorithm described above includes all the steps needed to generate our higher layer symbolic representation of the annotated reachability graph. In order to better clarify how such steps interact with each other and with the lower layer symbolic representation, we depict the overall procedure in Fig. 13 in graphical form. In this figure, the most relevant data structures involved in the procedure are also reported. Four main steps are represented:

Table 2 Pseudocode for the algorithm

<i>GenerateMTMDD</i> (in $m : model, p : partitioning, S_i : set\ of\ state, \mathcal{E}_m : set\ of\ event$) : <i>mtmdd</i>
Build the MTMDD encoding the state space S and the sets $T_a^{(s)}$ for model m considering the set of initial states S_i , partitioning p , and the set \mathcal{E}_m of events with age memory policy associated.
<ol style="list-style-type: none"> 1. declare $a, t : mdd$; 2. declare $u : mtmdd$; 3. $a \leftarrow InitializeMDD(m, p, S_i)$; 4. <i>ExplorerSaturationUnbound</i>(p, a, m); 5. $u \leftarrow a$; 6. foreach $e \in \mathcal{E}_m$ do 7. $t \leftarrow ExtractMDD(m, p, e, a)$; 8. <i>ExplorerSaturationClassical</i>(m, p, e, t); 9. <i>Sum</i>(t, e, u); 10. return u;
<i>InitializeMDD</i> (in $m : model, p : partitioning, S_i : set\ of\ state$) : <i>mdd</i>
Build the MDD encoding the set of initial states S_i for model m considering partitioning p .
<i>ExplorerSaturationUnbound</i> (in $p : partitioning, inout a : mdd, m : model$)
Recursively explore the MDD a saturating its node and generating the state space S for model m considering the partitioning p . Saturation Unbound is used. Information about the local state spaces and the next-state function are stored in m .
<i>ExtractMDD</i> (in $m : model, p : partitioning, e : event, a : mdd$) : <i>mdd</i>
Extract from MDD a a new MDD encoding the set of states enabling event e for model m considering partitioning p .
<i>ExplorerSaturationClassical</i> (in $m : model, p : partitioning, e : event, inout a : mdd$)
Recursively explore the MDD a saturating its node and generating the set of states reachable from states initially encoded by a considering all the event in \mathcal{E}_m except event e . The first version of Saturation is used.
<i>Sum</i> (in $t : mdd, e : event, inout u : mtmdd$)
For every state s encoded by MDD t if s doesn't enable event e then add 2^n to the value associated to s in MTMDD u where n is the integer associated to event e . If s enables event e then skip to the following state.

- *State space generation*: this step specifies the high-level model through some formalism (e.g., Petri nets, queuing network, process algebra) together with a partitioning of the model and runs the *InitializeMDD()* and *ExplorerSaturationUnbound()* procedures in order to generate the model reachability graph $RG(S^{init})$ stored in the form of an MDD and a set of Kronecker matrices $\mathbf{W}_{e,k}$.
- *Active events search*: the data structures generated are the input for the second step that produces an MTMDD encoding all the information about the active but not enabled events which is needed by the lower layer representation in order to generate all the Kronecker expressions representing the expanded reachability graph.
- *Kronecker expression generation*: based on the information stored into the MTMDD and on the high-level model, the set of Kronecker expressions can be computed, thus generating a compact representation for the expanded

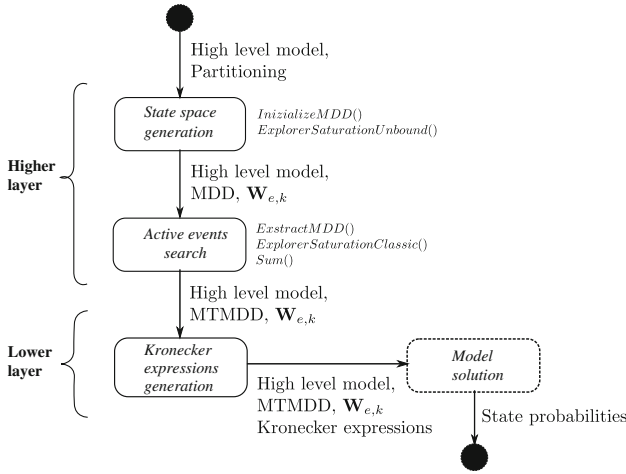


Fig. 13 Activity diagram of the two-layer representation algorithm

reachability graph. In an efficient implementation of the overall method, these expressions would not be explicitly stored but computed on-the-fly when needed, while evaluating the state probabilities.

- *Model solution*: the state probabilities of the expanded process are numerically evaluated.

The *Model solution* phase has been depicted in dashed line because it is not discussed in this chapter. This step has been reported to show how the proposed method works in the context of the overall problem to evaluate the process state probabilities.

4 A Numerical Example

In this section, we develop and solve a nontrivial Petri net model with several concurrently enabled general transitions. The example studies the completion time of a set of applications running in a time-sharing system. The application’s architecture follows the producer–consumer scheme. It is relevant because the Petri net model is characterized by several generally distributed transitions enabled in the same marking, and different kinds of preemption policies can be used according to the different behaviors of the application. This model can be used as a basis to analyze different types of systems such as transactional databases, manufacturing systems, and client/server communication systems.

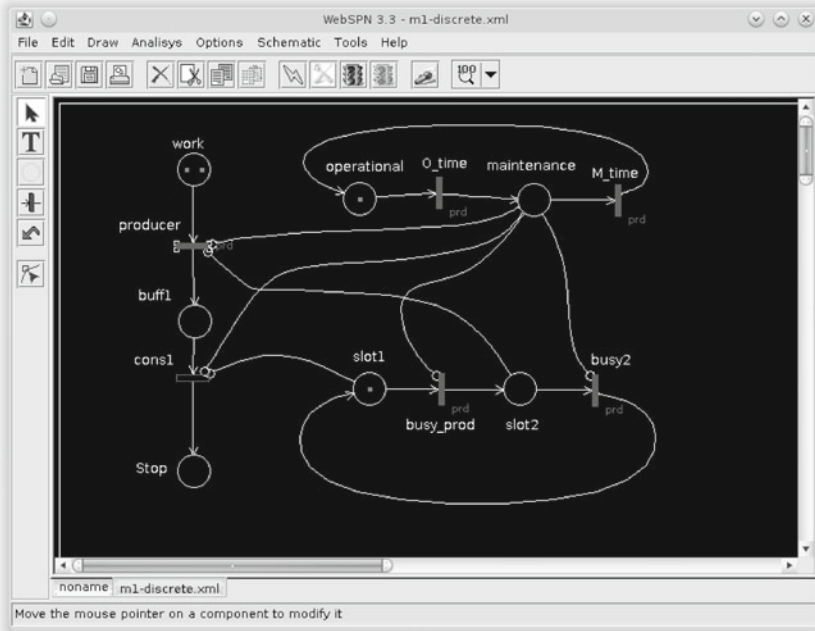


Fig. 14 Petri net model of producer–consumer applications

4.1 Model Description

The system moves between an operational phase, during which useful work is produced, and a maintenance phase in which the processing capacity is temporarily interrupted and no useful work is produced. The task of the system in the operational state is to process a certain number of jobs. The execution of each job consists of two sequential phases: the first one preprocesses the job, while the second one performs the processing. These two phases are executed in fixed time slots. We study the system in order to obtain the probability distribution of the time required to complete a fixed number of jobs.

The *Petri net* model we developed is shown in Fig. 14 directly in the WebSPN GUI, which represents the model of the system that, according to the description, consists of three functional blocks generically referred to as *Block1*, *Block2*, and *Block3*. *Block1* models the alternation of the system between the operational and maintenance phases. *Block2* models the two sequential phases of a job processing. Finally, *Block3* models the system turnover between the preprocessing and the processing job phases. The two phases are performed in time-sharing fashion.

Within *Block1*, the two operating states where the system can be are represented by places *operational* and *maintenance* and by transitions *O_time* and *M_time*. A token in place *operational* denotes the operational state, while a token in place *maintenance* denotes the maintenance state. The time duration of the operative phase is denoted by transition *O_time*, while the maintenance phase is denoted by transition *M_time*.

Block2 models the job processing. In particular, the number of jobs to be processed is denoted by the number of tokens contained in place *work*, while the time pre-processing job is represented by transition *producer*. Preprocessed jobs are therefore queued in a buffer (place *buff1*) where they wait for the second phase of processing (transition *cons1*).

In *Block3*, the alternation between the preprocessing phase and the job processing is represented through places *slot1* and *slot2* and transitions *busy_prod*, *busy2*, *idle_prod*, *idle2*. A token in place *slot1* denotes the system is executing the preprocessing of a job, while a token in place *slot2* denotes the execution of a phase of processing. An inhibitor arc between *slot1* and *cons1* deactivates the phase of processing when preprocessing is active. In the same way, the inhibitor arc between *slot2* and *producer* deactivates the phase of preprocessing when processing is active. The times that the system spends in these two activities are represented by transitions *busy_prod* and *busy2*.

The measure to evaluate from the analysis of the model is the probability distribution of the time required to complete the set of jobs assigned to the system at the beginning. It corresponds to the probability distribution of having a number of tokens equal to the number of jobs in place *Stop*.

For the firing time distributions of assigned to timed transitions, let us assume that transitions *O_time*, *M_time*, *busy_prod*, *busy2* are deterministic. Furthermore, we assume that firing time of transition *producer* follows a Weibull distribution and *cons1* is exponentially distributed.

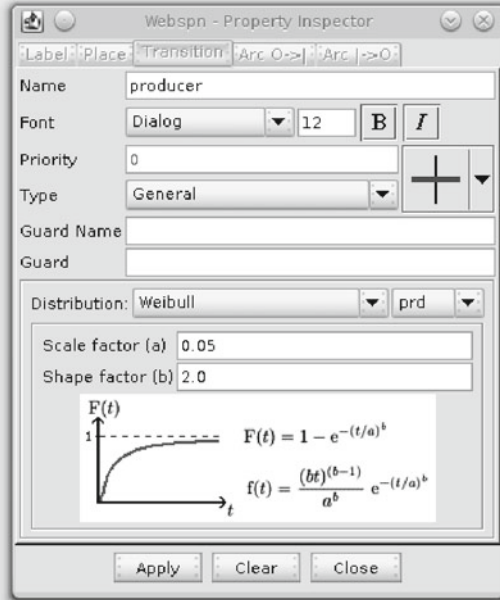
4.2 Numerical Results

The following parameters have been fixed:

- transition *producer* is distributed according to a Weibull distribution with scale factor $\lambda = 0.05$ and shape factor $k = 2$;
- transitions *O_time* and *M_time* are deterministic with a firing time equal to 1.0;
- transitions *busy_prod* and *busy2* are deterministic with a firing time equal to 0.1;
- transition *cons1* is distributed exponentially with a firing rate $\mu = 1.0$;
- the total number of jobs to be processed is 2.

Since the solution methods are based on approximating generally distributed firing times with DPH or CPH distributions, we provided the two solution runs with the appropriate phase-type distributions. In particular, in the case of discrete-time-based method, it is sufficient to set the transition properties with the parameter of the

Fig. 15 Transition properties window in WebSPN



selected distribution through the transition properties’ window (Fig. 15). The associated DPH is automatically computed using the discretization step and sampling the distributions with the same discretization step as described in [34]. When the continuous-time approach is used, in the actual implementation, the CPHs must be given by the modeler. Some files are used to store their infinitesimal generator matrix and the initial probabilities’ vector. We used the PhFit software tool [48] to compute the CPH approximating the Weibull distribution associated with the *producer* transition, generating eight CPH phases. Deterministic events have been modeled through Erlang distributions with 25 stages.

The results obtained are shown in Fig. 16. The completion time distributions computed using the two different methods implemented in WebSPN are denoted with DPH and CPH, respectively. As it can be noted, the two curves are consistent. The differences are due to the approximation of distributions with different kinds of phase-type distributions. In fact, it is a well-known result [49] that DPHs unlike CPHs can precisely capture behaviors characterized by very small, or also null, coefficient of variation. Due to this limit, the continuous-time method results into a less accurate result with respect to the one given by the discrete-time method. In this specific case, the absence of steps in the CPH curve is given by the use of Erlang distributions to approximate deterministic behaviors.

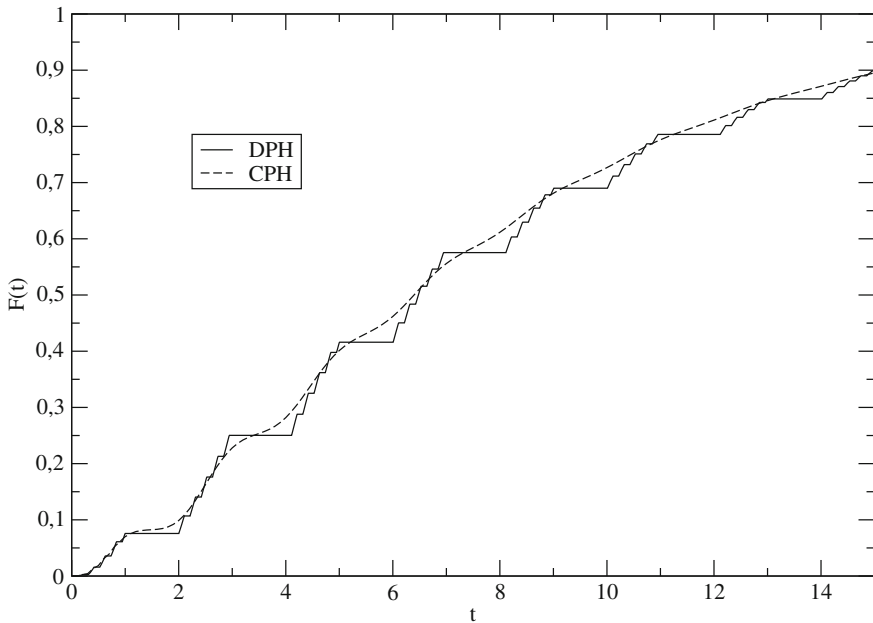


Fig. 16 Distribution of the completion time when transition *producer* has a *prd* memory policy

5 Conclusions

In this chapter, we presented WebSPN, a flexible tool for the analysis of NMSPNs. WebSPN implements a discrete-time expansion approach algorithm, providing an approximation of the stochastic process underlying an NMSPN in the case of multiple concurrently enabled generally distributed transitions with *prd*, *prs*, and *pri* memory policies. To overcome the explosion of the state space that might affect this approach, WebSPN also implements a continuous-time expansion, where two layers of symbolic representation are used to store the underlying reachability graph of the considered stochastic process in an efficient way. An example was provided to show the effectiveness of the WebSPN tool.

References

1. Cumani A (1985) Esp—A package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In: Proceedings international workshop timed Petri nets, Torino (Italy): IEEE Computer Society Press no. 674, pp 144–151
2. Chiola G (1992) GreatSPN 1.5 software architecture. In: Balbo G, Serazzi G (eds) Computer performance evaluation. Elsevier Science Publishers, pp 121–136

3. Ciardo G, Muppala J, Trivedi K (1998) SPNP: stochastic Petri net package. In: Proceedings international workshop on petri nets and performance models—PNPM89. IEEE Computer Society, pp 142–151
4. Lindemann C (1995) DSPNexpress: a software package for the efficient solution of deterministic and stochastic Petri nets. *Perform Eval* 22:3–21
5. German R, Kelling C, Zimmermann A, Hommel G (1994) TimeNET—a toolkit for evaluating non-markovian stochastic Petri nets. Report No. 19 - Technische Universität Berlin
6. Couvillon J, Freire R, Johnson R, Obal W, Qureshi M, Rai M, Sanders W, Tvedt J (1991) Performability modeling with UltrasAN. *IEEE Softw* 8:69–80
7. Bobbio A, Kulkarni V, Puliafito A, Telek M, Trivedi K (1995) Preemptive repeat identical transitions in markov regenerative stochastic Petri nets. In: 6-th International conference on Petri nets and performance models—PNPM95. IEEE Computer Society, pp 113–122
8. Bobbio A, Telek M (1995) Markov regenerative SPN with non-overlapping activity cycles. In: International computer performance and dependability symposium—IPDS95. IEEE CS Press, pp 124–133
9. Puliafito A, Scarpa M, Trivedi K (1998) Petri nets with k simultaneously enabled generally distributed timed transitions. *Perform Eval* 32(1) (February 1998)
10. Miner A, Parker D (2004) Symbolic representations and analysis of large state spaces. In: Validation of stochastic systems, ser. LNCS 2925. Springer, Dagstuhl, Germany, pp 296–338
11. Longo F, Scarpa M (2015) Two-layer symbolic representation for stochastic models with phase-type distributed events. *Intern J Syst Sci* 46(9):1540–1571 (Jul. 2015). <http://dx.doi.org/10.1080/00207721.2013.822940>
12. Longo F, Scarpa M (2009) Applying symbolic techniques to the representation of non-markovian models with continuous ph distributions. In: Bradley J (ed) Computer performance engineering, ser. Lecture notes in computer science, vol 5652. Springer, Berlin Heidelberg, pp 44–58. http://dx.doi.org/10.1007/978-3-642-02924-0_4
13. Bruneo D, Distefano S, Longo F, Puliafito A, Scarpa M (2010) Reliability assessment of wireless sensor nodes with non-linear battery discharge. In: Wireless days (WD), IFIP, Oct 2010, pp 1–5
14. Distefano S, Longo F, Scarpa M (2010) Availability assessment of ha standby redundant clusters. In: 29th IEEE symposium on reliable distributed systems, Oct 2010, pp 265–274
15. Distefano S, Longo F, Scarpa M (2010) Symbolic representation techniques in dynamic reliability evaluation. In: IEEE 12th International Symposium on High-assurance systems engineering (HASE), Nov 2010, pp 45–53
16. Bruneo D, Longo F, Puliafito A, Scarpa M, Distefano S (2012) Software rejuvenation in the cloud. In: Proceedings of the 5th international ICST conference on simulation tools and techniques, ser. SIMUTOOLS '12. ICST, Brussels, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp 8–16. <http://dl.acm.org/citation.cfm?id=2263019.2263022>
17. Bruneo D, Distefano S, Longo F, Puliafito A, Scarpa M (2012) Evaluating wireless sensor node longevity through markovian techniques. *Comput netw* 56(2):521–532. <http://www.sciencedirect.com/science/article/pii/S1389128611003690>
18. Bruneo D, Distefano S, Longo F, Puliafito A, Scarpa M (2013) Workload-based software rejuvenation in cloud systems. *IEEE Trans Comput* 62(6):1072–1085. <http://dx.doi.org/10.1109/TC.2013.30>
19. Distefano S, Bruneo D, Longo F, Scarpa M (2013) Quantitative dependability assessment of distributed systems subject to variable conditions. In: Pathan M, Wei G, Fortino G (eds) Internet and distributed computing systems, ser. Lecture notes in computer science, vol 8223. Springer, Berlin Heidelberg, pp 385–398. http://dx.doi.org/10.1007/978-3-642-41428-2_31
20. Distefano S, Longo F, Scarpa M (2013) Investigating mobile crowdsensing application performance. In: Proceedings of the third ACM international symposium on design and analysis of intelligent vehicular networks and applications, ser. DIVANet '13, New York, NY, USA: ACM, 2013, pp 77–84. <http://doi.acm.org/10.1145/2512921.2512931>

21. Distefano S, Longo F, Scarpa M, Trivedi K (2014) Non-markovian modeling of a blade-center chassis midplane. In: Horvath A, Wolter K (eds) *Computer performance engineering*, ser. *Lecture notes in computer science*, vol 8721. Springer International Publishing, pp 255–269. http://dx.doi.org/10.1007/978-3-319-10885-8_18
22. Longo F, Bruneo D, Distefano S, Scarpa M (2015) Variable operating conditions in distributed systems: modeling and evaluation. *Concurr Comput Pract Exp* 27(10):2506–2530. <http://dx.doi.org/10.1002/cpe.3419>
23. Longo F, Distefano S, Bruneo D, Scarpa M (2015) Dependability modeling of software defined networking. *Comput Netw* 83:280–296. <http://www.sciencedirect.com/science/article/pii/S1389128615001139>
24. Distefano S, Longo F, Scarpa M (2015) Qos assessment of mobile crowdsensing services. *J Grid Comput* 1–22. <http://dx.doi.org/10.1007/s10723-015-9338-7>
25. Ajmone Marsan M, Balbo G, Bobbio A, Chiola G, Conte G, Cumani A (1989) The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Trans Softw Eng* SE-15:832–846
26. Telek M, Bobbio A, Puliafito A (1996) Steady state solution of MRSPN with mixed preemption policies. In: *International computer performance and dependability symposium—IPDS96*. IEEE CS Press
27. Kulkarni V (1995) *Modeling and analysis of stochastic systems*. Chapman Hall
28. Neuts M (1981) *Matrix geometric solutions in stochastic models*. Johns Hopkins University Press, Baltimore
29. Horvath A, Puliafito A, Scarpa M, Telek M (2000) Analysis and evaluation of non-markovian stochastic Petri nets. In: Haverkort B, Bohnenkamp H, Smith C (eds) *Computer performance evaluation, modelling techniques and tools*, ser. *Lecture notes in computer science*, vol 1786. Springer, Berlin Heidelberg, pp 171–187. http://dx.doi.org/10.1007/3-540-46429-8_13
30. Ciardo G, Zijal R (1996) *Discrete deterministic and stochastic Petri nets*. Tech. Rep, NASA
31. German R (1995) New results for the analysis of deterministic and stochastic Petri nets. In: *International computer performance and dependability symposium—IPDS95*. IEEE CS Press, pp 114–123
32. Neuts M (1975) Probability distributions of phase type. In: *Liber Amicorum Prof. Emeritus H. Florin*. University of Louvain, pp 173–206
33. Miner AS (2001) Efficient solution of gspns using canonical matrix diagrams. In: *Proceedings of the 9th international workshop on Petri nets and performance models (PNPM'01)*, ser. *PNPM '01*. Washington, DC, USA: IEEE Computer Society, pp 101–. <http://dl.acm.org/citation.cfm?id=882474.883484>
34. Puliafito A, Horvath A, Scarpa M, Telek M (2000) Analysis and evaluation of non-markovian stochastic Petri nets. In: Haverkort B, Bohnenkamp H, Smith C (eds) *Proceedings of 11th international conference on modelling techniques and tools for performance analysis*. Springer, Schaumburg, Illinois (LNCS 1786, March 2000), pp 171–187
35. Ajmone Marsan M, Balbo G, Bobbio A, Chiola G, Conte G, Cumani A (1989) The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Trans Softw Eng* 15(7):832–846
36. Cumani A (1985) ESP—a package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In: *PNPM*, pp 144–151
37. Neuts MF (1981) *Matrix-geometric solutions in stochastic models: an algorithmic approach*. Johns Hopkins University Press, Baltimore
38. Pérez-Ocón R, Castro JER (2004) Two models for a repairable two-system with phase-type sojourn time distributions. *Reliab Eng Syst Safety* 84(3):253–260
39. Scarpa M (1999) *Non markovian stochastic Petri nets with concurrent generally distributed transitions*. PhD dissertation, University of Turin
40. Bellman R (1997) *Introduction to matrix analysis*, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA
41. Burch J, Clarke E, McMillan K, Dill D, Hwang L (1990) Symbolic model checking: 1020 states and beyond. In: *Proceedings of fifth annual IEEE symposium on logic in computer science*, June 1990. LICS '90, pp 428–439

42. Srinivasan A, Ham T, Malik S, Brayton R (1990) Algorithms for discrete function manipulation. In: IEEE international conference on computer-aided design, 1990. ICCAD-90. Digest of Technical Papers, Nov 1990, pp 92–95
43. Miner AS, Ciardo G (1999) Efficient reachability set generation and storage using decision diagrams. In: Proceedings 20th international conference on applications and theory of Petri nets. Springer, pp 6–25
44. Ciardo G, Luttgen G, Siminiceanu R (2001) Saturation: an efficient iteration strategy for symbolic state space generation. In: Proceedings of tools and algorithms for the construction and analysis of systems (TACAS), LNCS 2031. Springer, pp 328–342
45. Ciardo G, Marmorstein R, Siminiceanu R (2003) Saturation unbound. In: Proceedings of TACAS. Springer, pp 379–393
46. A. Bobbio and M. Scarpa, “Kronecker representation of stochastic petri nets with discrete ph distributions,” in Proc. Third IEEE Ann. Int’l Computer Performance and Dependability Symp. (IPDS ’98), 1998
47. G. Ciardo and R. Siminiceanu, “Structural symbolic ctl model checking of asynchronous systems,” in Proc. CAV. Springer-Verlag, 2003, pp. 40–53
48. A. Horváth and M. Telek, “Phfit: A general phase-type fitting tool,” in Proceedings of TOOLS ’02. London, UK: Springer-Verlag, 2002, pp. 82–91
49. A. Bobbio, A. Horvth, M. Scarpa, and M. Telek, “Acyclic discrete phase type distributions: properties and a parameter estimation algorithm,” Performance Evaluation, vol. 54, no. 1, pp. 1–32, 2003. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0166531603000440>

Modeling Availability Impact in Cloud Computing

Paulo Romero Martins Maciel

Internet-based services have become critical to several businesses in which many aspects of our lives depend on (e.g., online banking, collaborative work, video-conferencing). Business continuity is a remarkable property and it is a chief concern for many companies, since service disruption may cause huge revenue and market share losses.

In recent years, cloud computing has turned into a remarkable alternative due to its resource on-demand and pay-as-you-go models. More specifically, additional resources, such as virtual machines (VMs), are only allocated when disaster takes place, and the automated virtual platform also performs a transparent recovery to minimize the service time to restore. This chapter presents availability models to evaluate cloud computing infrastructures.

1 Introduction

The 80s saw the emergence and proliferation of personal computers. This prompted one of the greatest growths in computing. The 90s witnessed the network bandwidth advances that made the Internet available to general public. The lessons learned from the bursting of the financial bubble in 2000 required computing companies to rethink their business models. In the pursuit of converting computational resources into capital, many organizations came to understand that they could provide their solutions and resources as services.

P.R. Martins Maciel (✉)
Centro de Informática, Universidade Federal de Pernambuco,
Av. Jornalista Aníbal Fernandes, s/n - Cidade Universitária,
Recife CEP 50740560, Brazil
e-mail: prmm@cin.ufpe.br
URL: <http://www.modcs.org>

Cloud computing refers to the access to computing resources across a network. These resources include but are not limited to networks, storage, servers, and services. Clouds usually have common characteristics such as being allocated as required, accessibility across Internet devices, and metered and billed based upon resource usage. These services have a number of different terms, such as Software as a Service (SaaS), IaaS (Infrastructure as a Service), and PaaS (Platform as a Service) to describe their classes of products. Nevertheless, the terminology still varies widely [1, 2].

The deployment models are usually classified as Private Cloud, Public Cloud, Community Cloud, and Hybrid Cloud. A Private Cloud is usually adopted by a single organization and Public Clouds provide service to the public. Private clouds that are used by more than one organization, but not to the public, have commonly been named Community Clouds. Hybrid Clouds mix Public Clouds and Private and/or Community Clouds [2].

Business continuity is a remarkable asset and its assurance is essential to companies in general, since service disruption may cause huge revenue and market share losses. In recent years, cloud computing has become a prominent solution for many players due to its resource on-demand and pay-as-you-go models.

Failures are inevitable in complex systems, such as cloud systems, since hardware fails, dormant residual software bugs are activated, electrical power may be interrupted, etc. In such a system, service failures may be caused by service response time degradation, service request responses beyond defined time constraints, and recurrent service requests, which may produce service denial [3]. Therefore, regardless of class of resources provided, the architecture and deployment models adopted, cloud computing supported services can experience denial-of-service attacks, performance slowdowns, resource outages, and natural disasters.

Dependability is an umbrella expression that encompasses subjects such as availability, reliability, safety, and fault tolerance [4]. Dependability of a system can be understood as the ability to deliver a specified service that can be justifiably trusted. This chapter presents four studies evaluating cloud computing infrastructures and services availability related measures.

This chapter is structured in eight sections. Section 2 provides a brief historical view and describes some seminal works, their motivation, and the succeeding advances. Section 3 presents some basic concepts on dependability. After this, four case studies are discussed. Section 4 offers an availability model for redundant private cloud architectures. Section 5 describes an availability model for Video on Demand (VoD) Streaming service. Section 6 studies the availability impact on cloud computing infrastructure deployed into geographically distributed data centers and the effects related to disaster occurrence. Section 7 introduces an availability model to evaluate the impact of live VM migration as a software rejuvenation mechanism to mitigate aging-related failures and improving system availability. Finally, Sect. 8 presents some final considerations.

2 Reliability and Availability Models: History at a Glance

Dependability is an expression related to subjects such as availability, reliability, safety, and fault tolerance. The concept of dependable computing first appeared in the 1820s when Charles Babbage conceived his mechanical calculating engine for removing the possibilities of human errors [5].

In 1907, A.A. Markov began the study of processes in which the outcome of an experiment can affect the outcome of the next. This type of processes is now named a Markov chain. In the 1910s, A.K. Erlang studied reliable service provisioning in telephone traffic planning [6]. In 1931, Kolmogorov, in his famous paper “Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung” (Analytical methods in probability theory) set the grounds for the present theory of Markov processes [7].

The first electronic computers were quite undependable, thus many strategies were studied to enhance their reliability. In the 1950s, reliability turned out to be a subject of remarkable interest. Epstein and Sobels 1953 paper on the exponential distribution was a milestone [8], and radar system availability was studied through Markov chains by Anselone [9].

In 1961 Birnbaum, Esary, and Saunders published a landmark paper introducing coherent structures [10]. The reliability models might be classified as combinatorial (non-state-space model) and state-space models. Reliability Block Diagrams (RBD) and Fault Trees (FT) are combinatorial models and the most widely adopted models in reliability evaluation. RBD is probably the oldest combinatorial technique for reliability analysis. Fault Tree Analysis (FTA) was originally developed in 1962 at Bell Laboratories by H.A. Watson to evaluate the Minuteman I Intercontinental Ballistic Missile Launch Control System. Afterward, in 1962, Boeing and AVCO expanded use of FTA to the entire Minuteman II [11]. In 1967, A. Avizienis integrated masking methods with practical techniques for error detection, fault diagnosis, and recovery into the concept of fault-tolerant systems [12].

In the late 1970s, some works were proposed for mapping Petri nets to Markov chains [13–16]. These models have been widely adopted as high-level Markov chain automatic generation models as well as for discrete event simulation. Natkin was the first to apply what are now generally called Stochastic Petri nets to dependability evaluation of systems [16].

3 Basic Concepts

This section defines some basic concepts and quantitative measures for dependability.

As mentioned, dependability of a system is its capability to deliver a set of trustable services that are observed by outside agents. A system **failure** occurs when the system fails to provide its specified functionality. A **fault** is the basic abnormal condition or event which may lead to a system failure. A fault can be defined as the failure of a component of the system, a subsystem of the system, or another system which

interacts with the system considered. Hence, every fault is from some point of view. A fault can cause other faults, a system failure, or neither. A system with faults that delivers its specified functionality is said to be fault tolerant, that is, the system does not fail even when there are faulty components.

Consider an indicator random variable $X(t)$ that represents the system state at time t . $X(t) = 1$ represents the operational state and $X(t) = 0$ the faulty state. More formally:

$$X_s(t) = \begin{cases} 0 & \text{if } S \text{ has failed} \\ 1 & \text{if } S \text{ is operational} \end{cases} \quad (1)$$

Consider a random variable T that denotes the time needed to reach the state $X(t) = 0$, since the system started in state $X(0) = 1$. T is the system S time to failure, $F_T(t)$ its **cumulative distribution function**, and $f_T(t)$ the respective **density function**, where:

$$\begin{aligned} F_T(0) = 0 \quad \text{and} \quad \lim_{t \rightarrow \infty} F_T(t) = 1, \\ f_T(t) = \frac{dF_T(t)}{dt}, \\ f_T(t) \geq 0 \quad \text{and} \quad \int_0^{\infty} f_T(t) dt = 1. \end{aligned} \quad (2)$$

The probability that the system S survives to time t (**Reliability**) is

$$\begin{aligned} P\{T > t\} = R(t) = 1 - F_T(t) \text{ and} \\ R(0) = 1 \quad \text{and} \quad \lim_{t \rightarrow \infty} R(t) = 0. \end{aligned} \quad (3)$$

The **hazard rate** may be obtained through

$$\lambda(t) = \frac{dF_T(t)}{dt} \times \frac{1}{R(t)} = \frac{f_T(t)}{R(t)} = \frac{-dR(t)}{dt} \times \frac{1}{R(t)}. \quad (4)$$

From 4,

$$R(t) = e^{-\int_0^t \lambda(t) dt} = e^{-H(t)} \quad (5)$$

is obtained, where $H(t)$ is defined as the **cumulative failure rate function**.

The hazard rate $\lambda(t)$ may be decreasing, increasing, or constant. If $\lambda(t) = \lambda$, $R(t) = e^{-\lambda t}$. The **mean time to fail** (*MTTF*) is defined by:

$$MTTF = E[T] = \int_0^{\infty} t f_T(t) dt. \quad (6)$$

From 6,

$$MTTF = E[T] = \int_0^{\infty} R(t) dt \quad (7)$$

is obtained.

The simplest definition of **Availability** is expressed as the ratio of the expected system uptime to the expected system up and downtimes:

$$A = \frac{E[UpTime]}{E[UpTime] + E[DownTime]}. \quad (8)$$

The system availability may also be expressed by:

$$A = \frac{MTTF}{MTTF + MTR} = \frac{MTTF}{MTTF + MNRT + MTTR}, \quad (9)$$

where *MTR* is **mean time to restore**, *MTTR* **mean time to repair**, and *MNRT* is the **mean no-repairing time**.

If $MNRT \approx 0$,

$$A = \frac{MTTF}{MTTF + MTR} = \frac{MTTF}{MTTF + MTTR}. \quad (10)$$

As $MTBF = MTTF + MTR = MTTF + MNRT + MTTR$, and considering $MNRT \approx 0$, then $MTBF = MTTF + MTTR$. Therefore:

$$A = \frac{MTBF}{MTBF + MTTR}, \quad (11)$$

where *MTBF* is the **mean time between failures**. The **instantaneous availability** ($A(t)$) is the probability that the system is operational at t .

A system may provide a set of services. The correct provision of a service defines an **operational mode**. Therefore, the system services may be represented by a set of operational modes. If the system performs more than one service (function or operation), a Boolean function should define each operational mode (for each service, function, or operation). The meaning of intended functionality must be specified and depends on the objective of the study. Hence, the system being operational for

a particular service does not mean it also is operational for another service. The system state may also be described by the respective structure functions of its components or subsystems, so that the system structure function evaluates to 1 whenever at least a minimal number of components is operational [17]. A **failure mode** specifies one way (mode) in which the system might fail. Hence, for the system services, the system failures may be specified by a set of failure modes.

Reliability and availability models may be broadly classified into combinatorial and state-space models. State-space models may also be referred to as non-combinatorial, and combinatorial can be identified as non-state-space models. Combinatorial models capture conditions that make a system fail (or work) in terms of structural relationships between the system components. These relations observe the *set of components* (and subsystems) that should be correctly working (faulty) for the *system* to be working properly (faulty).

State-space models represent a system's behavior (failures and repair activities) by its states and event occurrences expressed as labeled state transitions. Labels can be probabilities, rates, or distribution functions. These models allow representation of more complex relations between components of the system, such as dependencies involving subsystems and resource constraints. Some state-space models may also be evaluated by discrete-event simulation. The most prominent combinatorial models are Reliability Block Diagrams (**RBD**) and Fault Trees (**FT**). **Markov chains**, **stochastic Petri nets**, and **stochastic process algebras** are the most widely used state-space models [11, 13, 16, 18–21].

4 Case Study 1: A Private Cloud Infrastructure Platform

Cloud computing infrastructures must rely on various fault tolerance mechanisms to cope with software and hardware failures, so that resources are accessible anywhere and anytime as expected [1, 22]. The replication of components and subsystems is an important approach to guarantee high system availability [17]. Even relatively small infrastructures, such as those that support private clouds, should consider replication as a key requirement by employing multiple clusters and redundant service endpoints [23].

An availability model is useful to define expected metrics that may be included in service level agreements (SLAs). Hierarchical and composite modeling approaches are useful to deal with the complexity of such systems, especially when using redundancy mechanisms [24, 25]. This section presents an availability model for redundant private cloud architectures. The environment considered follows the common architectural components of cloud frameworks such as Eucalyptus, OpenNebula, and CloudStack.

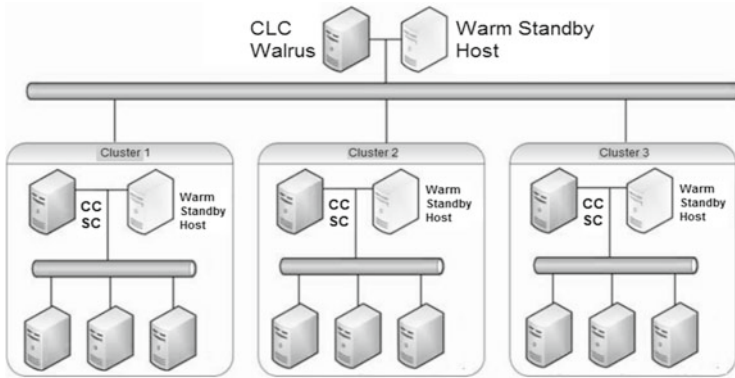


Fig. 1 Eucalyptus cloud with three clusters

4.1 Infrastructure—An Example

Figure 1 shows an architecture with three clusters and redundancy for some Eucalyptus components, according to the study presented in [26]. A front-end computer plays the role of Cloud Manager and runs the Eucalyptus components known as the Cloud Controller (CLC), and Walrus. A warm standby host is capable of keeping the Cloud Manager subsystem working if the primary host fails. Each cluster has one machine called the Cluster Manager, which runs the Cluster Controller (CC) and Storage Controller (SC) components. Warm standby redundancy also empowers each Cluster Manager. Each cluster also has three machines that run the Eucalyptus component known as the Node Controller (here named as NC'). The set of three nodes in each cluster is a Nodes Subsystem.

The system infrastructure is available if the Cloud Subsystem is running and at least one Cluster Subsystem is available with one or more nodes completely operational in that cluster (operational mode).

4.2 Infrastructure Model

The proposed model is depicted in three levels: a high-level RBD system model, Markov Reward models (MRM) to represent the warm standby redundancy mechanisms, and RBD models to evaluate nodes and managers (cloud and cluster) component stacks. The components' stacks are presented in Fig. 2. An accurate representation of the warm standby redundancy mechanisms requires the use of state-based models, such as an MRM, which is a Continuous Time Markov Chain (CTMC) extended with a reward rate assigned to each state [27].

The system-level RBD describes the availability at the highest view (see Fig. 3), whereas the MRM represents the detailed behavior of subsystems that employ an

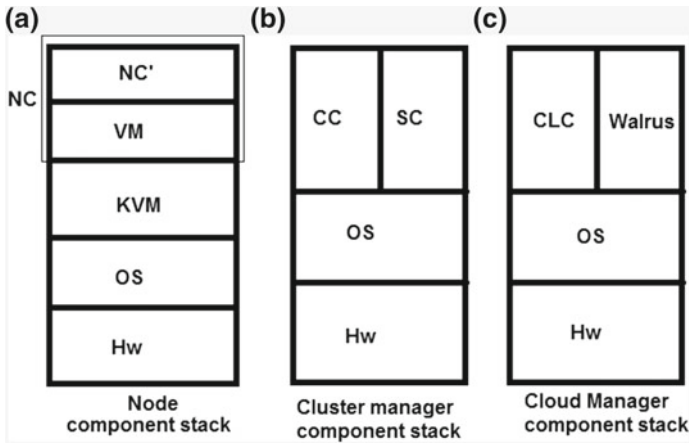


Fig. 2 Component's stack. a Node component stack, b cluster manager component stack, c cloud manager component stack

Fig. 3 RBD model for three-cluster private cloud

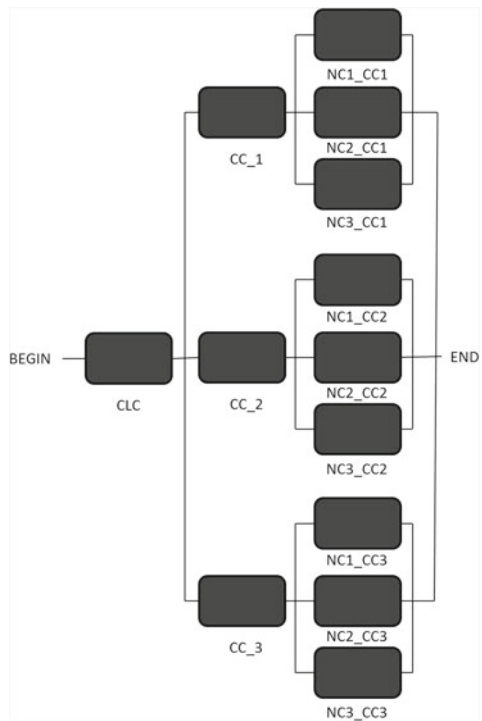




Fig. 4 RBD model for one cloud node

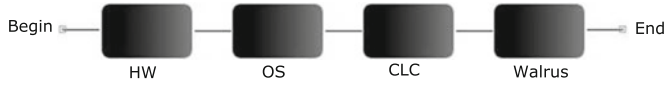


Fig. 5 Nonredundant Cloud Manager RBD



Fig. 6 Nonredundant Cluster Manager RBD

active redundancy mechanism [17]. The block named CLC represents the Cloud Manager Subsystem. Blocks CC_j represent each of the Cluster Subsystems, where $j \in [1, N]$, and N is the number of clusters. Each block $NC_i CC_j$ represents the i th node that integrates the j th cluster.

Each node is composed of hardware, an operating system, hypervisor (e.g., KVM), a virtual machine (VM), and a Eucalyptus Node Controller (NC'). For the sake of simplicity, in each node, the Eucalyptus Node Controller, and the respective VM are represented as only one component, named NC. A node is properly working only if all these components are active (non-failed). Figure 4 shows the RBD model that represents one node in each Node Subsystem (NM). The Cloud Manager stack (depicted in Fig. 2.a) consists of hardware, the operating system, and software components CLC (Cloud Controller), and Walrus. Figure 5 depicts an RBD that represents a single host that composes the Cloud Manager Subsystem (CIM). A similar RBD model represents a nonredundant Cluster Subsystem (CM), which is composed of hardware, operating system, Cluster Controller (CC) and Storage Controller (SC), as depicted in Fig. 6. As this study considers redundancy in both the Cloud Manager and Cluster Subsystems, these RBDs enable obtaining the equivalent MTTF and equivalent MTTR values that will be parameters in the MRM for the corresponding redundant subsystem.

The redundant Cloud Manager (RCIM) and redundant Cluster Subsystem (RCM) steady-state availability were computed through the MRM shown in Fig. 7. The MRM has five states: UW , UF , FF , FU , and FW , and considers one primary and one spare server, respectively. The state UW represents the primary server, S_1 , is functional (U) and secondary server, S_2 , in standby (W). When S_1 fails (F), the system goes to state FW , since a time interval is required to detect S_1 failure. FU represents the state where S_2 leaves the waiting condition and assumes the active role, whereas S_1 is down. If S_2 fails before taking the active role, or before the repair of S_1 , the system goes to the state FF . This model represents a setup where the primary server repair

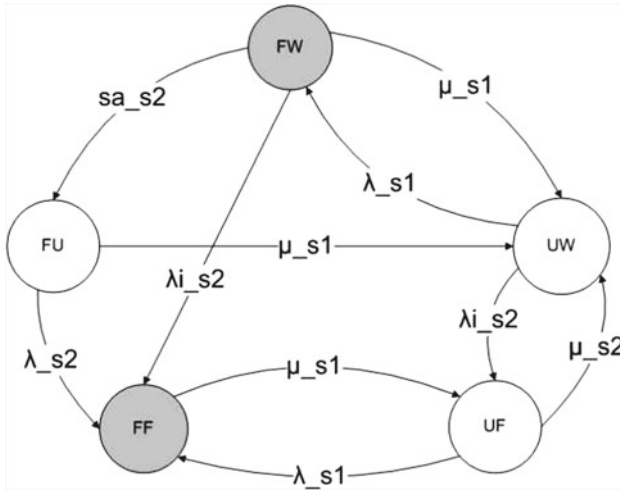


Fig. 7 Markov Reward Model for redundant Cloud Manager

has priority over the secondary server repair. Therefore, when both servers have failed (state FF), there is only one possible outgoing transition: from FF to UF . If S_2 fails when S_1 is up, the system goes to state UF , and returns to state UW when the S_2 repair is accomplished. Otherwise, if S_1 also fails, the system transitions to the state FF . The failure rates of S_1 and S_2 are λ_{s1} and λ_{s2} , respectively. The rate λ_{i_s2} denotes the failure rate of the secondary server when it is inactive. The repair rates assigned to S_1 and S_2 are μ_{s1} and μ_{s2} . The rate sa_{s2} represents the switchover rate, i.e., the reciprocal of the mean time to activate the secondary server after a failure of S_1 .

The state reward rate $\rho(s)$ assigned to UW , UF , and FU is equal to 1, since the subsystem is available in these states. The state reward rate assigned to FF and FW (shaded states) is equal to 0, since the subsystem is down in these states. Therefore, the steady-state availability of the subsystem can be computed as the steady-state reward rate of the MRM, so $A = \sum_{s \in S} \pi_s \times \rho(s)$, where π_s is the steady-state probability of being in the state s , and $\rho(s)$ is the reward rate assigned to state s .

4.3 Scenario Evaluation and Results

The input parameters for such models are presented in Table 1 [24]. It is possible to compute measures for systems with one and two clusters using similar models. Table 2 presents the values for all the mentioned parameters of the MRM. The value of μ_{s1} is equal to the value of μ_{s2} , the rates λ_{s1} and λ_{s2} also have equal values.

These λ and μ values were obtained from the equivalent $MTTF$ and $MTTR$ estimated from the Cloud Manager subsystem (CIM) RBD (Fig. 5). When the secondary Cloud Manager subsystem is inactive, its failure rate (λ_{i_s2}) is assumed to be 20 %

Table 1 Input parameters for the controllers and nodes

Component	MTTF (h)	MTTR
HW	8760.0	100 min
OS	2893.0	15 min
CLC	788.4	1.0h
CC	788.4	1.0h
SC	788.4	1.0h
Walrus	788.4	1.0h
NC	788.4	1.0h
KVM	2990.0	1.0h

Table 2 Input parameters for the MRM

Parameter	Description	Value (h^{-1})
$\lambda_{s1} = \lambda_{s2} = \lambda$	Host failure rate	0.00553
λ_{is2}	Inactive host failure rate	0.00461
$\mu_{s1} = \mu_{s2} = \mu$	Host repair rate	1.03000
sa_{s2}	Rate for spare host activation	2.00000

smaller than the failure rate of the respective active subsystem (λ_{s2}). The value of sa_{s2} comes from default monitoring interval and activation times found in softwares such as Heartbeat [28]. After evaluating MRM (RCIM and RCM subsystem), the Cluster subsystems RBDs, and Node subsystems' RBDs, the corresponding availability values of each subsystem were used in the high-level system RBD model.

Table 3 presents the availability measures of the cloud system considering the architectures with one, two, and three clusters, called hereinafter A_1 , A_2 , and A_3 . Besides the steady-state availability, Table 3 also shows the number of nines, which constitutes a logarithmic view of the availability, and the downtime, which clearly describes the impact of service unavailability from the user's perspective.

Table 3 reveals that architectures A_2 and A_3 decrease system downtime by about 50 % when compared to A_1 . When comparing A_2 and A_3 , very small differences are traceable. Therefore, increasing the number of clusters beyond three will have

Table 3 Availability results

Architectures			
Measure	A_1	A_2	A_3
Steady-state Availab.	0.999938749	0.999969376	0.999969377
Number of 9s	4.21288	4.51394	4.51395
Annual downtime (min)	32.194	16.096	16.095

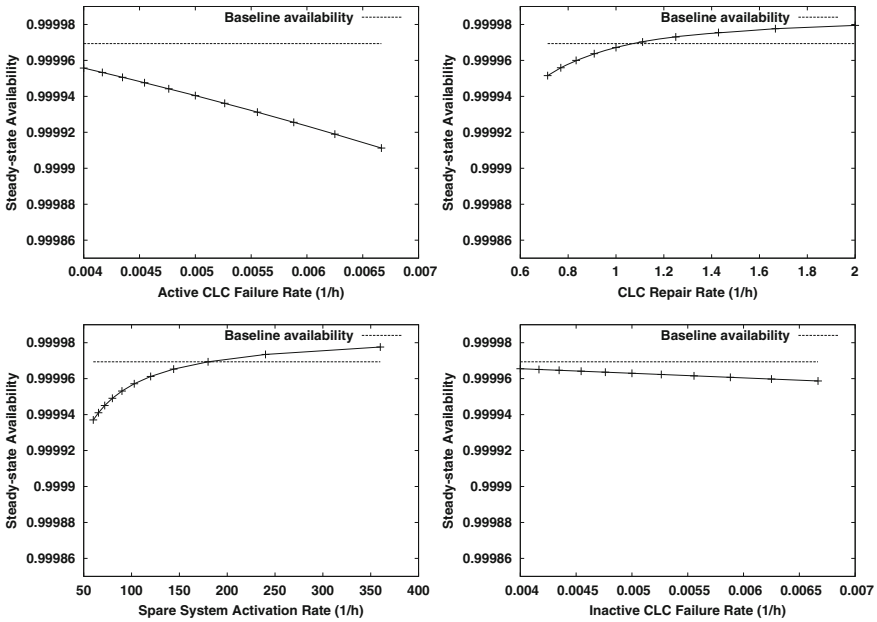


Fig. 8 Sensitivity analysis: four most impacting parameters

negligible impact on the availability, and would be justified only by a need to increase computing capacity and performance. The sensitivity analysis depicted in Fig. 8 shows the parameters that produce the highest impact on the steady-state availability and, therefore, deserve priority in their enhancement. The failure and repair rates of the Cloud Manager subsystem, the activation of the spare warm standby server, and the failure rate of inactive spare server are the most important points of improvement shown in the results.

5 Case Study 2: Redundant VoD Streaming Service

In recent years, VoD service has been dominant in Internet traffic [29]. This class of traffic requires secure, scalable, and highly available infrastructures and cloud services to maintain users’ confidence and avoid revenue losses due to events such as hardware failures, planned maintenance, resources replacement, software bugs, or updates [24, 30].

This section studies an availability model for Video on Demand (VoD) Streaming service and evaluates the impact of different component parameters on the overall system availability. The analysis employs a heterogeneous modeling strategy by merging RBD and Markov chains. A sensitivity analysis is also applied to identify

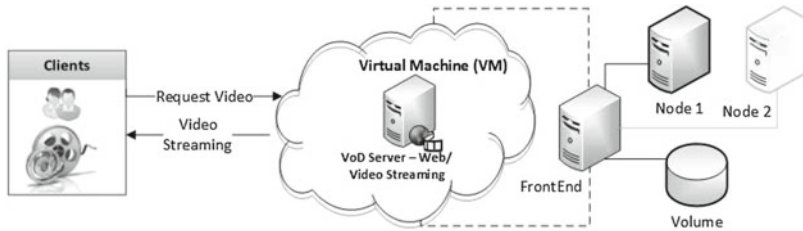


Fig. 9 Architecture of video streaming service

the critical components with respect to availability. We examine how the system availability is affected by the adoption of redundant mechanisms.

5.1 VoD Service Infrastructure

Figure 9 shows an architecture with one front-end and two redundant Nodes based on some Eucalyptus components [31, 32]. A front-end computer runs the Eucalyptus components known as the Cloud Controller (CC), Storage Controller (SC), Walrus, and Cluster Controller. A Node computer runs the Eucalyptus components known as Node Controller (NC) and one hypervisor (KVM) that is responsible for virtualization management, in addition to bringing hardware (HW) and operating system (OS) components [33, 34]. A storage volume is allocated in the front-end to store the videos. A Virtual Machine (VM), running the Apache web service [35] and a VoD server (VLC player configured as server—VLC server) applications, is instantiated in the cloud Nodes. VoD server provides the video streaming features, whereas Apache is responsible for hosting the service on a dedicated web page.

When a user requests a video hosted on a specific web page, VLC server, in turn, captures the requested video from the remote storage volume and relays the stream to the user. Users connect to the video streaming server through the Internet and a storage volume is allocated in the front-end for storing videos. In order to conduct the evaluation, a baseline architecture was first considered. The baseline architecture consists of one front-end and only one Node.

5.2 VoD Service Model

This section presents the availability models proposed to evaluate the system. First, a model is introduced to represent the baseline architecture (nonredundant) depicted in Fig. 9. This infrastructure is composed by the following components: a front-end, a storage volume, a cloud node, and service.



Fig. 10 Nonredundant system top-level RBD



Fig. 11 Nonredundant front-end subsystem RBD



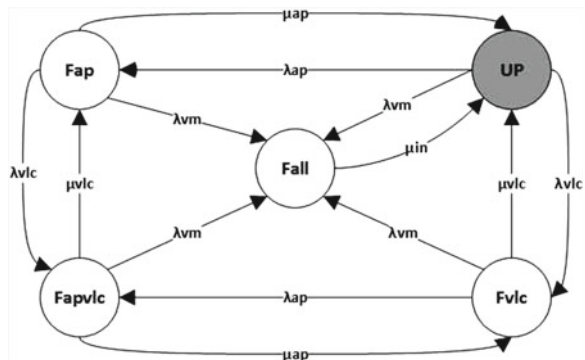
Fig. 12 RBD model for nonredundant cloud Node

The top-level system RBD represents these four parts, depicted by the blocks *front-end*, *volume*, *node*, and *vm-service* (see Fig. 10). The front-end subsystem consists of hardware (HW), and operating system (OS), and the following Eucalyptus components: CLC (cloud controller), CC (cluster controller), SC (storage controller), and Walrus. The front-end subsystem RBD is shown in Fig. 11.

A node subsystem is composed of hardware (HW), operating system (OS), a hypervisor (KVM), and node controller (NC) (see Fig. 12). The *vm-service* component represents the VoD server, which is composed of: one virtual machine—VM, VLC application, and Apache application. Due to interdependency between these components, the service subsystem is further refined by a CTMC (see Fig. 13). The respective availability is calculated from the CTMC and then fed into the top-level RBD (Fig. 10) for the computation of overall system availability.

This CTMC is composed of five states: *UP*, *Fap*, *Fapvlc*, *Fvlc*, and *Fall*. The white circles denote the downstates (when the service is unavailable due to failure in at least one component) and the gray circle indicates the operational state. The state

Fig. 13 CTMC model service



Fapvlc represents that both Apache and VLC have failed. The state *Fall* represents a VM failure. The rates λ_{ap} , λ_{vlc} , and λ_{vm} denote the failure rates for Apache, VLC, and VM, respectively. As the failure and repair rates (λ_i and μ_i) are constant, the related mean time to failure and repair are given by $MTTF_i = \frac{1}{\lambda_i}$, and $MTTR_i = \frac{1}{\mu_i}$. The repair rate for Apache and for the VLC are, consequently, μ_{ap} and μ_{vlc} . μ_{in} is the rate for starting a new VM instance combined with Apache web service and a VoD server application. μ_{in} is the reciprocal of the mean time to initiate a VM after it has been requested.

In order to improve the availability of the VoD Service, a redundant mechanism can be adopted. The proposed architecture has two redundant nodes working in warm standby mode. The VM service is switched between a working node and standby node, when the operational node fails. The analytical model used to represent this redundant architecture is given by the RBD in Fig. 14, which is composed by: frontend, volume and *vm-service+nodes*.

The respective frontend and volume blocks are the same as in the baseline model. The *vm-service+nodes* block is composed of two redundant nodes (*Node₁* and *Node₂*) and *vm-service*. Due to dependencies between the nodes and *vm-service*, a CTMC (see Fig. 15) was conceived to compute the *vm-service+nodes* subsystem availability.

This CTMC is composed of 11 states: *UUW*, *UUD*, *UWU*, *UDU*, *DUW*, *DUD*, *DDW*, *DDU*, *DWU*, and *DWD*. The three letters represent the operating condition of the three components, that is, *vm-service*, the *Node₁* and the *Node₂*, respectively.

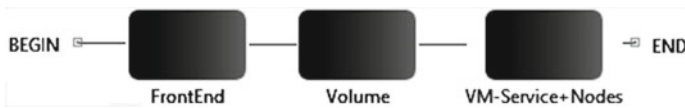


Fig. 14 RBD model for redundant architecture

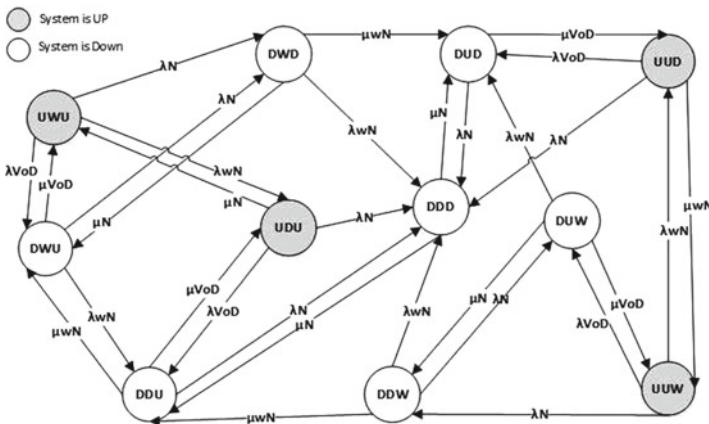


Fig. 15 CTMC model for the service module

The *vm-service* may be up (U) or down (D). The Nodes $Node_i$ works alternately in warm standby mode, and only one of them should be up (U) at any one time, whereas the other is either in warm standby (W) or down (D). In this model, the initial state is denoted by UWU , where the *vm-service* is available, $Node_1$ in warm standby, and $Node_2$ is operational.

5.3 Scenario Evaluation

The input parameters for FrontEnd, nonredundant and redundant Node system are presented in Table 4. Tables 5 and 6 present the values for all the mentioned parameters of the CTMC. The value of μ_{in} comes from the monitoring of start-up time of a virtual machine.

The value of λ_{N1} is equal to the value of λ_{N2} and it is represented by λ_N , the rates μ_{N1} and μ_{N2} also have equal values and it is represented by μ_N . These λ_N and μ_N values were obtained from the equivalent MTTF and MTTR estimated from Node subsystem RBD (Fig. 12). When the $Node_2$ subsystem is inactive, its failure rate (λ_{wN}) is assumed to be 20% smaller than the failure rate of the respective

Table 4 Input parameters for frontend and node cloud systems

Component	MTTF (h)	MTTR
HW	8760.0	100 min
OS	2893.0	15 min
CLC	788.4	1.0h
CC	788.4	1.0h
SC	788.4	1.0h
Walrus	788.4	1.0h
NC	788.4	1.0h
KVM	788.4	1.0h
Volume	100000.0	1.0h

Table 5 Input parameters for nonredundant service—CTMC

Rate	Description	Value (h^{-1})
λ_A	Mean time to Apache failure	0.001268
λ_{VLC}	Mean time to VLC failure	0.002976
λ_{VM}	Mean time to VM failure	0.000347
μ_A	Mean time to Apache repair	2.000000
μ_{VLC}	Mean time to VLC repair	2.000000
μ_{in}	Mean time to instantiate a new VM	52.164841

Table 6 Parameters—redundant service+nodes CTMC

Rate	Description	Value (h ⁻¹)
$\lambda_{N1} = \lambda_{N2} = \lambda_N$	Mean time to node failure	0.002075
$\lambda_{wN1} = \lambda_{wN2} = \lambda_{wN}$	Mean time to standby node failure	0.001730
$\mu_{N1} = \mu_{N2} = \mu_N$	Mean time to node repair	1.098901
$\mu_{wN1} = \mu_{wN2} = \mu_{wN}$	Mean time to standby node repair	30.303030
λ_{VOD}	Mean time to service failure	0.004592
μ_{VOD}	Mean time to service instantiate	37.037037

Table 7 Steady-state availability results

Measure	Nonredundant system	Redundant system
Steady-state availability	0.9886	0.9944
Number of 9s	1.9420	2.5118
Annual downtime (h)	100.12	49.05

active subsystem (λ_N). The λ_{VOD} and μ_{VOD} values were obtained from the equivalent MTTF and MTTR estimated from *vm-service* CTMC (Fig. 13). The availability values obtained in CTMCs (see Figs. 13 and 15) were used as input parameters for the high-level system RBD models for the nonredundant and the redundant architecture (see Figs. 12 and 14).

Table 7 gives the dependability results for these scenarios. The values of 0.9886 and 0.9944 were obtained for the availability of the nonredundant and redundant architecture, respectively. The availability growth of 2.5118 nines represents 51.01 % annual downtime reduction when compared to the baseline architecture. The sensitivity analysis depicted in Fig. 16 shows the parameters that produce the highest impact to the steady-state availability, hence deserving priority for enhancement. The failure and repair rate availability impact related to the frontend components are depicted in Fig. 16a, b, respectively.

The sensitivity analysis for the *vm-service* (see Fig. 13) shows that the VLC failure rate has greater impact on system availability in comparison to the Apache and VM (see Fig. 16c) and the nonredundant architecture. The result also reveals that the greatest attention should be paid to frontend and VLC components.

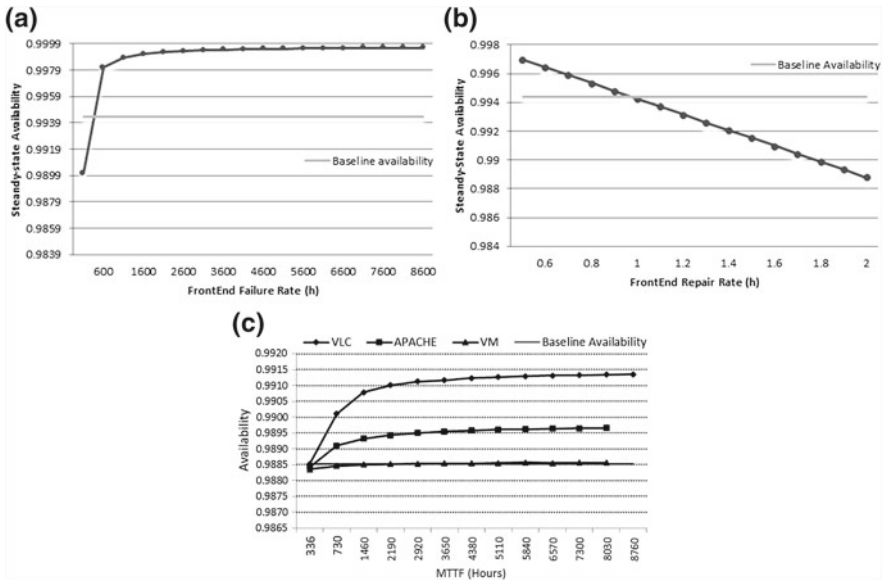


Fig. 16 Sensitivity analysis: **a** frontend failure rate; **b** frontend repair rate; **c** MTTF of VLC, Apache, and VM

6 Case Study 3: Modeling Disaster Tolerant Cloud Infrastructures

This case study presents an availability model for cloud computing systems deployed into geographically distributed data centers as well as taking into account disaster occurrence.

A disaster recovery plan requires the utilization of different data centers located far enough apart to mitigate the effects of unforeseen disasters (e.g., earthquakes) [36]. If multiple data centers are located in different geographical locations (considering disaster independent places), the availability level of the whole system is likely to improve. On the other hand, VM backup time will probably increase due to the distance between data centers. Additionally, failures and overloads can lead to system downtime when considering cloud computing systems. Consequently, performability evaluation considering VM data backup time and different user load levels is of utmost importance when considering the analysis of distributed cloud systems.

In order to address these concerns, a model is presented to evaluate dependability metrics in IaaS systems deployed into geographically distributed data centers as well as taking into account disaster occurrence. The proposed model represents IaaS providers distributed in different sites and allows the sensitivity evaluation of VM transmission time on performability metrics. A tool is also conceived to support the evaluation by automatically creating the system models, namely Geoclouds [37–39].

6.1 An Architecture

This section introduces a case study to illustrate the proposed model. In this case study the systems are deployed into two different data centers, which are located in five different sites. The pairs of data center locations are: Rio de Janeiro (Brazil)-Brasilia (Brazil), Rio de Janeiro-Recife (Brazil), Rio de Janeiro-NewYork (USA), Rio de Janeiro-Calcutta (India), and Rio de Janeiro-Tokyo (Japan). The Backup Server is located in São Paulo (Brazil).

Each data center consists of two physical machines and each machine is capable of running two virtual machines. This study assumes that all physical machines (PMs) are identical. PMs may share a common network attached storage (NAS) or a storage area network (SAN) to provide distributed storage and to allow the migration of a virtual machine from one server to another in the same data center [40]. In case of failure, a VM must be instantiated in another physical machine. If there is no available PM, the VM image is migrated to another data center. Furthermore, a Backup Server (BS) is assumed to provide VM data backup. This component periodically receives a copy of each VM image during data center operation. Hence, whenever a disaster renders one data center unavailable, BS periodically sends VM copies to an operational data center.

Figure 17 presents a cloud infrastructure, which is composed of a data center located in Site 1 (Data Center 1), a second data center in Site 2 (Data Center 2), and a Backup server in Site 3. Each data center consists of two physical machines and each machine is capable to run up two virtual machines. The evaluation will consider performance and dependability metrics to assess the system quality.

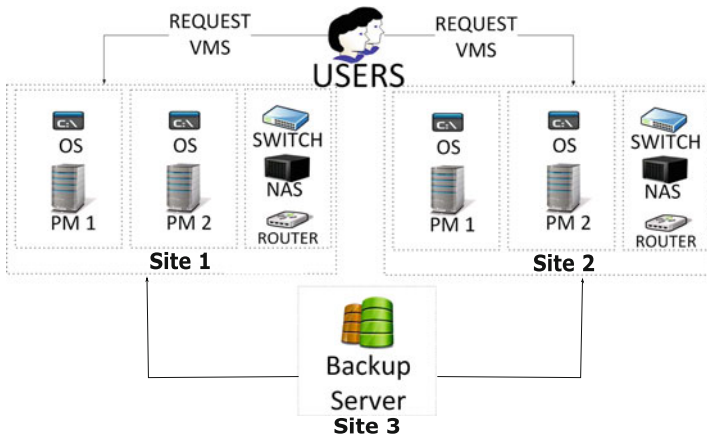


Fig. 17 Cloud system architecture

Table 8 Dependability parameters for components of Fig. 18

Component	MTTF (h)	MTTR (h)
Operating System (OS)	4000.0	1.0
Hardware of Physical Machine (PM)	1000.0	12.0
Switch	430000.0	4.0
Router	14077473.0	4.0
NAS	20000000.0	2.0
VM	2880.0	0.5
Backup server	50000.0	0.5

Table 8 presents the dependability parameters associated with the devices [41–43]. The default approach adopted to assess the network throughput considers the distance between the communication nodes [44, 45]. Nevertheless, other methods may also be adopted, for instance, experiments can be conducted to estimate the available bandwidth between sites. Expressions 12–14 estimate an upper bound on the transfer rate.

$$\text{Rate} < \frac{MSS}{RTT_{\min} \times \sqrt{p}}, \quad (12)$$

$$RTT_{\min} = \frac{2 \times D}{\alpha \times V_p}, \quad (13)$$

$$\alpha = \frac{EED}{HD}, \quad (14)$$

where MSS is the maximum segment size (typically 1460 bytes), p is the packet loss probability, and RTT_{\min} is the minimum RTT (Round Trip Time), D is distance (Km), V_p is propagation speed in Km/ms, and α is the *Directness*. $\alpha \in (0, 1]$ is a measure that relates the hop distance (HD) and the end-to-end distance (EED) of two nodes, where $EED \geq HP$.

In this particular study, we adopted $p = 1\%$, $MSS = 1460$ bytes, $\alpha = 0.45$, $V_p = 200$ Km/ms, and the VM size as 4 GB. We considered the mean time between disaster to be 100 years and the data center to take 30 days to be repaired. Moreover, a VM takes 5 min to set up and the mean time between requests is half an hour. The mean time for using a VM is 720 h.

In order to perform the evaluation, the user should provide the parameters and execute the evaluation. The assessment process can be conducted in a transparent way directly on GeoClouds tool or creating the SPN models from scratch using the Mercury [46–48], TimeNET [49], and Sharpe [20] tools.

6.2 Architecture Model

The SPN model related to the previous example is presented in Fig. 18, which is composed of four VM performability submodels, and one transmission submodel as well several generic submodels.

A generic model represents components without redundancy. These components may be in two states, operational or failed. *MTTFs* and *MTTRs* are the only parameters needed for computing their availability. *OSPM_1*, *OSPM_2*, *OSPM_3*, *OSPM_4*, *DISASTER1*, *DISASTER2*, *NAS_NET_1*, and *NAS_NET_2* are generic models. Places *UP* and *DOWN* (e.g., *DC_UP_1* and *DC_DOWN_1*) denote component's operational and failed states, respectively. *OSPM_1* and *OSPM_2* represent physical machines in Data Center 1, and *OSPM_3* as well as *OSPM_4* represent PMs in Data Center 2. *DISASTER1* and *DISASTER2* depict disasters in Data Centers 1 and 2, respectively. *NAS_NET_1* and *NAS_NET_2* correspond to network devices in Data Center 1 and 2 [38, 39].

The VM performability model represents VM requests to a single physical machine, considering failures and repairs on the underlying infrastructure. Whenever a user request is performed, a new VM is started (considering that the infrastructure is operational) and becomes available for some time. This component interacts with three generic components: (i) disaster occurrence (*DC*), (ii) network infrastructure (*NAS_NET*), and (iii) the physical machine (*OSPM*).

If while executing a VM, the external infrastructure or the VM fails, a VM image should migrate to other physical machine. If no physical machine is available, the task is rejected and a new request should be made. Figure 19 presents the VM performability model, which is composed of three main parts: (i) *VM_PART* represents the VM; (ii) *DC_PART* depicts incoming requests to data center; and (iii) *CLOUD_PART* models generation of requests.

In *VM_PART*, places *VM_UP*, *VM_DOWN*, *VM_STRTD*, and *VM_WAIT* denote, respectively, the number of operational and failed VMs as well as those starting and waiting for instantiation requests. *VM_LT* represents the VM operational period. Transitions *VM_F*, *VM_R* and *VM_STRT* represent VM failure, repair and starting activities, respectively. A VM is provided if the respective infrastructure is capable of supporting the service, otherwise no VM is granted. If the system is providing a VM and the physical machine or a network device fails or a disaster occurs, every VM instance in the infrastructure is freed. This event is represented by the transition *EXT_FAIL* firing. Physical machine and network device failures as well as a disaster are denoted by $\#OSPM_UP=0$ OR $\#NAS_NET_UP=0$ OR $\#DC_UP=0$. This condition turns the *EXT_FAIL* guard expression true (see Table 9). At this marking, *EXT_FAIL* fires. Its firing removes every token from places *VM_UP*, *VM_DOWN* and *VM_STRTD* (denoting VMs in the respective conditions) and stores $\#VM_UP + \#VM_DOWN + \#VM_STRTD$ in place *VM_WAIT*. Transition *VM_Subs* denotes virtual machine starting. Its execution requires an operational infrastructure, represented by $(\#OSPM_UP>0)$ AND $(\#NAS_NET_UP>0)$ AND $(\#DC_UP>0)$. *DC_PART* models incoming requests to a data center, and events denoting failed VMs that should be migrated to other machine. Incoming requests are denoted by

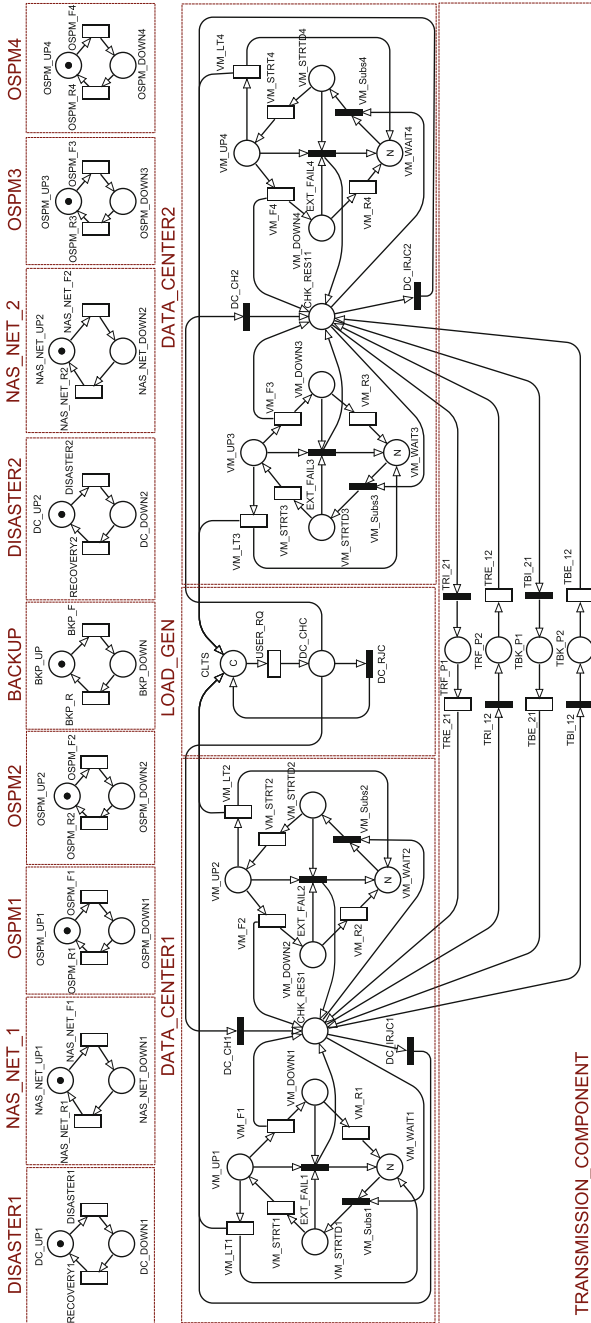


Fig. 18 Case study output model

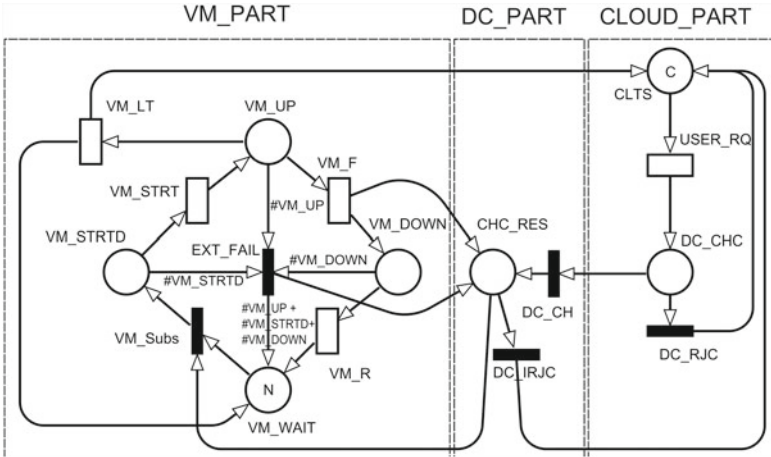


Fig. 19 VM performability SPN model

tokens in place *CHC_RES*. If no VM instance could be created, the request is canceled. Request canceling is represented by transition *DC_IRJC*. Its firing occurs when $\#NAS_NET_UP=0$ (network infrastructure failure), $\#DC_UP=0$ (disaster incident), $\#VM_WAIT=0$ (server cannot instantiate a VM), or $\#OSPM_UP=0$ (the underlying physical machine is not operational). The guard expressions are presented in Table 9.

Table 9 Guard expressions

Transition	Condition	Description
EXT_FAIL	$(\#OSPM_UP=0)$ OR $(\#NAS_NET_UP=0)$ OR $(\#DC_UP=0)$ AND $((\#VM_UP + \#VM_DOWN + \#VM_STRTD) > 0)$	Failure of physical machine or infrastructure
VM_Subst	$(\#OSPM_UP>0)$ AND $(\#NAS_NET_UP>0)$ AND $(\#DC_UP>0)$	Physical machine and infrastructure working
DC_IRJC	$(\#NAS_NET_UP=0)$ OR $(\#DC_UP=0)$ OR $(\#VM_WAIT=0)$ OR $(\#OSPM_UP=0)$	Task rejection
DC_CH	$(\#NAS_NET_UP>0)$ AND $(\#DC_UP>0)$ AND $(\#VM_WAIT>0)$ AND $(\#OSPM_UP>0)$	Request acceptance to data center
DC_RJC	$(\#NAS_NET_UP=0)$ OR $(\#DC_UP=0)$ OR $(\#VM_WAIT=0)$ OR $(\#OSPM_UP=0)$	Task rejection

CLOUD_PART model clients' requests for VM instantiation and request denial when all data centers are unavailable. It is assumed that a client requests a single VM. Transition *USER_RQ* denotes the VM instantiation request. *DC_RJC* represents instantiation request denial. This event arises when all data centers are unavailable. Both transitions have single server semantics (sss). A data center cannot support a VM instantiation request if the underlying infrastructure has failed or the servers have no further capacity to instantiate additional VMs. It is important to stress that the guard expressions assigned to *DC_IRJC*, *DC_CH*, and *DC_RJC* are evaluated considering the numbers of physical machines and data centers. In this particular case, two physical machines are provided by each data center. However, the model is generic enough to consider multiple physical machines and data centers.

A VM should migrate to another data center whenever the current data center capacity does not allow additional VM instantiations or the underlying infrastructure has failed. Moreover, Backup Server is responsible for transmitting the VM image in case of disaster or network error. Transmission component (see Fig. 18) represents the data transfer from a data center to another and the data transference from Backup Server to the data center.

Two metrics have been considered in the evaluation: VM Utilization (U) and the Probability of Successful Task Completion (P). U is computed considering:

$$U = \frac{\sum_{j=1}^M E\{\#VM_UPj\}}{M \times N}, \quad (15)$$

where M denotes the number of PMs and N is the maximum number of VMs supported by a physical machine. P is depicted by:

$$P = \frac{(\sum_{j=1}^{P_m} E\{\#VM_UPj\})(1/T)}{P\{\#CLTS > 0\}(1/R)}, \quad (16)$$

where T and R are delays assigned to transitions *VM_LT* and *USER_REQ*. P divided by incoming request rate provides the percentage of completed requests, and the probability of a task to be rejected is ($R = 1 - P$).

6.3 Results and Discussion

This section presents the evaluation results related to different workload intensity ($\#C$ —number of users) and different pair of locations.

A task rejection considering the proposed approach happens if the infrastructure is broken or the servers are full. Figure 21 presents the P percentage increase consider-

Table 10 P and U for the baseline architecture (Rio de Janeiro-Brasilia)

Client load	U	P
#C = 2	0.2497989	0.9999299
#C = 4	0.4989833	0.9952341
#C = 6	0.7471487	0.9338905
#C = 8	0.9861435	0.4542866
#C = 10	0.9867157	0.2934411

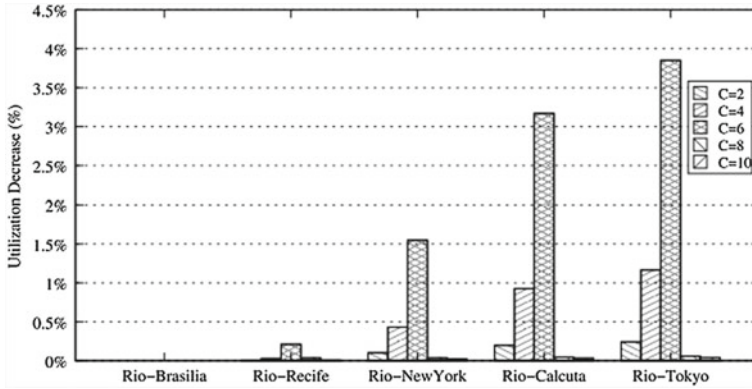


Fig. 20 Utilization for different distributed cloud configurations

ing the same set of scenarios. It is possible to observe that, in this particular case, P rises with the distance between the data centers (until $\#C = 6$). With the growth of system load ($\#C > 6$), P decreases depending on the data centers distance. Therefore, for this particular system we can conclude that P is highly impacted by server utilization. Table 10 depicts U and P of the baseline architecture (Rio de Janeiro-Brasilia) considering $\#C = 2, 4, 6, 8$ and 10 users.

Figure 20 shows utilization variation between pairs of cities and the baseline (Rio de Janeiro-Brasilia), taking into account different number of clients ($\#C$). The results show that the utilization decreases with the distance when considering light workloads ($\#C \leq 6$). In case of heavy workloads ($\#C > 6$), on the other hand, the effect of distance on the utilization is reversed.

Figure 21 presents the P rise for the same set of scenarios. P increases with the distance between the data centers (until $\#C \leq 6$). For heavier workloads ($\#C > 6$), P decreases with distances between the data centers.

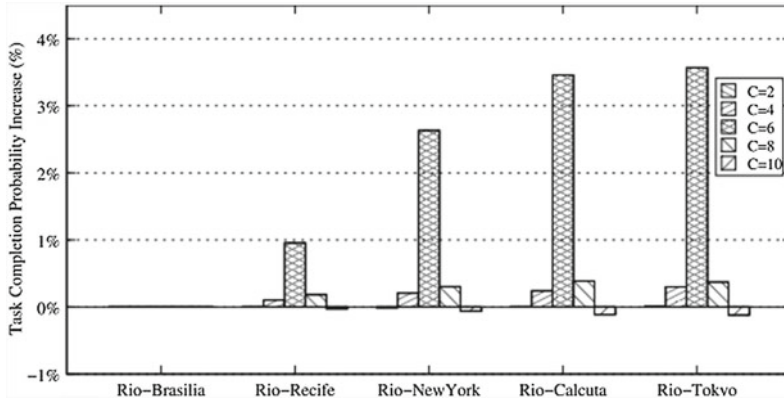


Fig. 21 P increase for different cloud configurations

7 Case Study 4: Live VM Migration in Cloud Computing

Cloud Computing absorbs characteristics of various previous technologies like Grid Computing and Virtualization for better utilization of computational resources and load balancing [50]. An important feature inherited from virtualization is the ability to move Virtual Machines (VMs) from one physical host to another. This characteristic is known as Live Virtual Machine Migration (*Live VM Migration*) [51]. In general, Live VM Migration supports significant gains to systems, such as manageability by allowing server consolidation, resources remapping before scheduled maintenance and VM rearrangement to implement fault tolerance [52]. Therefore, system administrators can perform Live VM Migration-based actions in order to improve system dependability.

One such opportunities is the adoption of Live VM Migration for improving system availability when coping with software aging-related failures in both VM and VMM. A software component aging consists of performance degradation and/or failure rate growth as the software component executes [53, 54]. This study considers that the VMM is affected by software aging [55]. Software rejuvenation is a proactive fault management, since it can either prevent or postpone failures in VMs and VMMs [56, 57]. Nevertheless, exceedingly often rejuvenation actions directed to a VMM may cause too many VM interruptions or termination, thus jeopardizing the infrastructure availability. Therefore, to cope with these conflicting aspects, Live VM Migration strategies may be applied before carrying out rejuvenation actions so as to reduce system downtime [51]. It is also important to stress that live migration is an intensive process, so it may also degrade the system availability.

Finding a suitable rejuvenation frequency, adopting a proper class of redundant mechanism, and speeding up the migration process are key aspects to be evaluated and tuned for reaching required availability levels.

7.1 An Example

Consider a system with three main components: *Management Server*, *Main Node*, and *Standby Node*. *Management Server* is a component responsible for controlling the cloud environment. The *Main Node* represents the main physical host in which a VMM controls a VM that executes a software application. *Standby Node* is a spare host which assumes the role of Main Node when a VM migrates to it. This mechanism is similar to warm standby replication [58]. The system’s organization is presented in Fig. 22. Besides these three components, a remote storage volume is accessed by the VM and managed by the *Management Server*. All components are interconnected through a private network.

The system operational mode is described as follows: the *MainNode* and its VM should be running and working properly, *Management Server* also should be operational, because it controls the cloud environment. If the *Standby Node* fails, the system does not stop, the migration mechanism, however, would be disabled. It is worth highlighting that the roles of *Standby Node* and *Main Node* are swapped when the VM migrates, hence the respective host availability becomes essential to system availability as soon as an incoming migration is completed.

When the *Main Node* is up and running, the VMM is aging (what can lead to failure), and the VM or other *Main Node* components (hardware and operating system) may also fail. The *Standby Node* fails if its hardware or operating system goes down. If the *Main Node* or *Standby Node* suffer a nonaging failure all the aging effects are cleared due to the repair mechanisms, which encompasses restarting the related components, and, subsequently, starting over the VMM.

In order to clear aging effects on VMM, rejuvenation is periodically scheduled, which is supported by VM live migration to minimize the downtime. When a VM migration is requested, the *Main Node* moves the VM to the *Standby Node*. When the VM migration completes, the *Standby Node* assumes *Main Node* role and a rejuvenation process is performed on the previous *Main Node*. When this process finishes, the original *Main Node* assumes the *Standby Node* role. The rejuvenation process allows to clear aging status by taking the VMM (on the node) to a fresh state, and prepares to receive a VM again when needed.

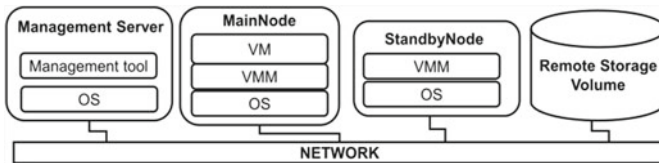


Fig. 22 System organization

7.2 System Model

The availability of proposed models are represented by an extended Deterministic Stochastic Petri Nets (eDSPNs) [59, 60]. The models cover the occurrence of both nonaging and aging-related failures in the system. Such a characteristic enables the analysis of the rejuvenation impact on the whole system. The evaluation performed does not take into neither account failure detection time nor remote storage volume and network failures.

Figure 23 presents the eDSPN model for the system under study. The model is composed of three submodels: (a) ManagementServer Model, (b) Clock Model, and (c) System Model. The *ManagementServer Model* is a straightforward availability model that represents the Management Server cloud behavior. The failure event is represented by *MS_fail* transition and repair event is denoted by *MS_repair* transition. The *Clock Model* depicts the rejuvenation scheduling. Transition *Trigger* fires after a specified deterministic delay. Its firing stores a token in the place *ReadyToMigrate*. Transition *ResetClock* is guarded by an enabling function. This function is evaluated as true only when the migration process is over (see Table 11). The *System Model* represents the main events related to the Main Node, Standby Node, and VM. A token in place *MN_UP* denotes the *Main Node* and its VM as operational (see Fig. 22). At this marking, several transitions may be fired for

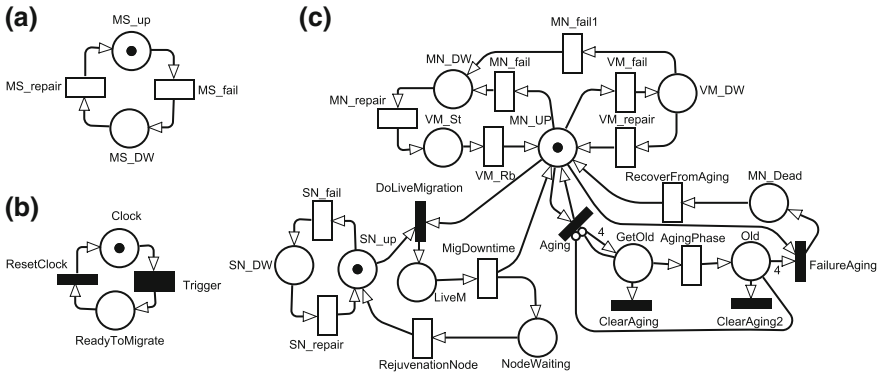


Fig. 23 eDSPN model for a cloud computing environment with scheduled live VM migration to support software rejuvenation. **a** ManagementServer Model, **b** Clock Model, **c** System Model

Table 11 Enabling function

Transition	Enabling function
<i>ResetClock</i>	$\#LiveM = 1$
<i>DoLiveMigration</i>	$\#ReadyToMigrate = 1$
<i>ClearAging</i> and <i>ClearAging2</i>	$\#VM_DW = 1$ OR $\#MN_DW = 1$ OR $\#LiveM = 1$

representing the respective system event. *MN_fail* fires when an internal (nonaging) failure occurs in the *Main Node*. As the VM is on the top of the node infrastructure, the VM also becomes unavailable. The recovering process is carried out in two steps: first, the Main Node is repaired, then the VM is restarted.

Another possible issue is the VM failure. This is represented by the transition *VM_fail* firing, which stores one token in place *VM_DW*. This place represents a failed VM. At this marking, two events are possible: VM repairing, so the system returns to the operational state, or the Main Node failure. If the later occurs, the same two-step recovery process is carried out. In any case, the system only goes up when both the VM and Main Node are.

In this case study, the software aging phenomenon related to VMM is represented by a sub-net that depicts a 4-phase Erlang probability distribution.¹ This model has been adopted since the respective process has an increasing failure rate. The sub-net depicted by transitions *Aging*, *AgingPhase*, *FailureAging*, *ClearAging*, *ClearAging2*, and places *GetOld* and *Old* represents the 4-phase Erlang distribution. Therefore, this sub-net denotes the aging phenomenon. The transitions *ClearAging* and *ClearAging2* represent events that clear the VMM aging effects. The model consider that these events occur when the *Main Node* or the VM fails, since the respective repair actions involve rejuvenation actions. When a Live VM migration is performed the VM is moved to a fresh VMM environment, therefore justifying the removal of aging effects. If none of these events occurs, the VMM will reach a critical age, that leads to a failure (represented by the transition *FailureAging* firing). After failure, the node reaches an inactive state (represented by a token in place *MN_Dead*). When place *MN_Dead* is marked, it enables transition *RecoverFromAging* (that represents a repairing action). Its firing depicts the aging failure system repairing.

A token in *SN_up* means the spare machine is operational and can receive a VM by live migration. However, if *SN_fail* fires, the Standby Node reaches an inactive state and needs to be repaired to return to active state (*SN_repair* is fired). It is important to stress that when the Standby Node fails, the system will fail only if the Main Node or the Management Server also fails.

The rejuvenation process, supported by the live migration, is modeled as follows. The transition *DoLiveMigration* represents the live migration start event. This transition will fire if some conditions are observed. The *MainNode* and *StandbyNode* must be properly executing and the clock should reach the time to migrate (denoted by token in place *ReadyToMigrate* of *Clock Model*). If only one of these two conditions is satisfied the migration cannot occur. When both conditions are satisfied, *DoLiveMigration* fires and stores a token in place *LiveM*. Live VM Migration consists of moving a VM to other host, which is represented by transition *MigrationTime* [51]. Thus, a token in place *LiveM* denotes that the system is down. After finishing the Live VM migration, the source node will undergo a rejuvenation process, the *StandbyNode* will take the *MainNode* role, and the system will be up again. Therefore, in this model, the VM live migration mitigates long downtimes

¹Erlang distributions (with number of phase large than one) have increasing failure rate [61].

due to rejuvenation action by keeping a spare machine clear of aging effects. The rejuvenation process is depicted by transition `RejuvenationNode`.

7.3 Scenario Evaluation and Results

The model parameters used for evaluation (presented in Table 12) are based on [56, 57]. The main objective is to evaluate the impact that different rejuvenation policies based on Live VM migration will produce on the steady-state availability. For this purpose, a scenario was conceived in which a set of different rejuvenation policies was considered and evaluated for estimating steady-state availability and annual downtime. A suitable rejuvenation trigger interval was estimated for maximizing the system availability.

The rejuvenation time interval varied from 1 to 12 h, using a 1-h step. It is important to highlight that these values represent the mean time interval between migrations. The values adopted for each phase of the Erlang sub-net (that represents the aging phenomenon) were obtained from [55]. Figure 24 presents the evaluation results. Table 13 shows a comparison with the baseline, where *BLA* is baseline availability, *AR* is availability achieved when applying rejuvenation schedules, *Tr* denotes the rejuvenation time interval, and *DTr* denotes the downtime avoided in hours per year.

Table 12 SPNs parameters

Transition name	Description	Mean time
MS_fail	MS internal failure	481.50 h
MS_repair	MS repair	1.03 h
MN_fail, MN_fail1	MainNode internal failure	1236.70 h
MN_repair	MainNode repair	1.09 h
SN_fail	StandbyNode internal failure	1236.70 h
SN_repair	StandbyNode repair	1.09 h
VM_fail	VM failure	2880.00 h
VM_repair	VM repair	0.50 h
VM_Rb	VM reboot	5 min
AgingPhase	Time to aging (phases)	25.00 h
RecoverFromAging	Time to recover from aging failure	1.00 h
MigrationTime	Time to live migrate a VM	4 s
RejuvenationNode	Time to rejuvenate node	0.50 h

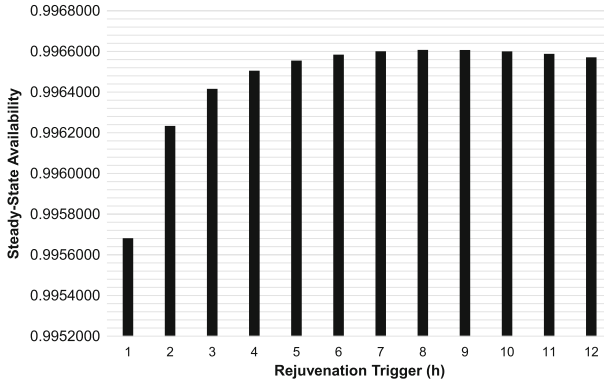


Fig. 24 Sensitivity analysis of rejuvenation policies on steady-state availability

Table 13 Results from model analysis

<i>BLA</i>	<i>AR</i>	<i>Tr</i>	<i>DTr</i> (hr/yr)
0.9873428	0.9966080	8 h	81.1631

The results show that Live VM Migration may substantially improve the system availability. We can also observe that overly frequent Live VM Migrations can harm the system availability. The models provide support for tuning rejuvenation policies to achieve availability improvements.

8 Final Considerations

Nowadays, Internet-based services are essential for businesses, and many aspects of our lives depend on its continuous provision. Cloud computing became a remarkable means to provide on-demand resources, since the respective costs may be proportional to the business actual computational demand. This book chapter presented a set of availability models for evaluating cloud computing infrastructures.

Four case studies were discussed. Section 4 presented an availability model for redundant private cloud architectures. Section 5 showed an availability model for Video on Demand (VoD) Streaming service. Section 6 evaluated the availability impact on cloud computing infrastructure deployed into geographically distributed data centers and the effects related to disaster occurrence. Section 7 presented an availability model to evaluate the impact of VM live migration as a software rejuvenation mechanism for mitigating aging-related failure.

Acknowledgments I would like to show my gratitude to all my students and colleagues, whom have worked with me over the years. I wish to present my special thanks to Bruno Silva, Jamilson Dantas, Jean Araújo, Maria Clara Bezerra, Matheus Torquato, Rosangela Melo, and Rubens Matos

for carrying out the experiments and tests. These models are part of their graduation studies. I am in indebted to the anonymous referees for their very helpful comments and suggestions. I am also deeply grateful to the editors, and particularly to Lance Fiondella who read the draft and made several suggestions to improve the text. Naturally, any remaining mistakes and omissions are of my own responsibility.

References

1. Sun (2009) Introduction to cloud computing architecture. Sun Microsystems Inc
2. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I et al (2010) A view of cloud computing. *Commun ACM* 53(4):50–58
3. Bauer E, Adams R (2012) Reliability and availability of cloud computing. Wiley-IEEE Press, Hoboken. ISBN 1118177010
4. Avizienis A, Laprie J, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans Dependable Secure Comput* 1:11–33
5. Schaffer S (1994) Babbage’s intelligence: calculating engines and the factory system. *Crit Inquiry* 21(1): 203–227. ISSN 00931896. <http://www.jstor.org/stable/1343892>
6. Erlang AK (1909) The theory of probabilities and telephone conversations. *Nyt Tidsskr Mat* 20(B):33–39
7. Kolmogoroff A (1931) Ber die analytischen methoden in der wahrscheinlichkeitsrechnung. *Math Ann* 104(1):415–458. ISSN 0025–5831. doi:10.1007/BF01457949. <http://dx.doi.org/10.1007/BF01457949>
8. Epstein B, Sobel M (1953) Life testing. *J Am Stat Assoc* 48(263): 486–502. ISSN 01621459. <http://www.jstor.org/stable/2281004>
9. Anselone PM (1960) Persistence of an effect of a success in a bernoulli sequence. *J Soc Ind Appl Math* 8(2):272–279. ISSN 03684245. <http://www.jstor.org/stable/2098965>
10. Birnbaum ZW, Esary JD, Saunders SC (1961) Multi-component systems and structures and their reliability. *Technometrics* 3(1):55–77. ISSN 00401706. <http://www.jstor.org/stable/1266477>
11. Ericson C (1999) Fault tree analysis—a history. In: 17th international systems safety conference
12. Avizienis A (1997) Toward systematic design of fault-tolerant systems. *Computer* 30(4):51–58
13. Molloy MK (1981) On the integration of delay and throughput measures in distributed processing models. Los Angeles, CA, USA
14. Natkin S (1980) Les Reseaux de Petri Stochastiques et leur Application a l’Evaluation des Systkmes Informatiques. Thèse de Docteur Ingegneur, CNAM, Paris, France
15. Symons FJW (1978) Modelling and analysis of communication protocols using numerical Petri Nets. Essex Ph
16. Ajmone Marsan M, Conte G, Balbo G (1984) A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Trans Comput Syst (TOCS)* 2(2):93–122
17. Maciel PRM, Trivedi KS, Matias R, Kim DS (2012) Dependability modeling. In: Performance and dependability in service computing, pp 53–97. IGI Global, 2012. doi:10.4018/978-1-60960-794-4.ch003. <http://dx.doi.org/10.4018/978-1-60960-794-4.ch003>
18. Trivedi K (2002) Probability and statistics with reliability, queueing, and computer science applications, 2 edn. Wiley Interscience Publication
19. Ajmone Marsan M, Balbo G, Conte G, Donatelli S, Franceschinis G (1994) Modelling with generalized stochastic Petri Nets, 1st edn. Wiley, New York. ISBN 0471930598
20. Sahner RA, Trivedi KS (1987) Reliability modeling using sharpe. *IEEE Trans Reliability* 36(2):186–193
21. German R (2000) Performance analysis of communication systems with Non-Markovian Stochastic Petri Nets. Wiley, New York

22. Yeow W-L, Westphal C, Kozat UC (2010) A resilient architecture for automated fault tolerance in virtualized data centers. In: 2010 IEEE Network operations and management symposium (NOMS), pp 866–869
23. Dantas J, Matos R, Araujo J, Maciel P (2015) Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud. *Computing*. ISSN 0010–485X. doi:10.1007/s00607-015-0447-8. <http://dx.doi.org/10.1007/s00607-015-0447-8>
24. Dantas J, Matos R, Araujo J, Maciel P (2012) An availability model for eucalyptus platform: an analysis of warm-standby replication mechanism. In: 2012 IEEE international conference on systems, man, and cybernetics (SMC), IEEE, pp 1664–1669
25. Longo F, Ghosh R, Naik VK, Trivedi KS (2011) A scalable availability model for infrastructure-as-a-service cloud. In 2011 IEEE/IFIP 41st international conference on dependable systems networks (DSN), pp 335–346. doi:10.1109/DSN.2011.5958247
26. Dantas J, Matos R, Araujo J, Maciel P (2012) Models for dependability analysis of cloud computing architectures for eucalyptus platform. *Int Trans Syst Sci Appl* 8:13–25
27. Bolch G, Greiner S, de Meer H, Trivedi KS (2005) *Queueing networks and Markov chains*. Wiley-Interscience. ISBN 0471565253
28. Linux-HA Project (2014) Heartbeat. Linux-HA Project. <http://www.linux-ha.org>
29. Cisco (2012) Cisco visual networking index: forecast and methodology, 2011–2016. CISCO White paper, pp 2011–2016
30. Sun D, Chang G, Guo Q, Wang X (2010) A dependability model to enhance security of cloud environment using system-level virtualization techniques. In: 2010 First International Conference on Pervasive Computing Signal Processing and Applications (PCSPA), IEEE, pp 305–310
31. Bezerra MC, Melo R, Dantas J, Maciel P, Vieira F (2014) Availability modeling and analysis of a vod service for eucalyptus platform. In: 2014 IEEE international conference on systems, man, and cybernetics (SMC), IEEE
32. de Melo RM, Bezerra MC, Dantas J, Matos R, de Melo Filho IJ, Maciel P (2014) Redundant VoD streaming service in a private cloud: availability modeling and sensitivity analysis. *Math Prob Eng* 2014:1–14. doi:10.1155/2014/764010. <http://dx.doi.org/10.1155/2014/764010>
33. Open source private and hybrid clouds from Eucalyptus. <http://www.eucalyptus.com>
34. Eucalyptus (2010) Eucalyptus cloud computing platform—administrator guide. Technical report, Eucalyptus Systems Inc, Version 2
35. Apache (2015) The apache software foundation. <http://www.apache.org/>
36. Hitachi Hyper-V Live Migration over Distance. <http://goo.gl/GzlkNk>
37. Silva B, Maciel P, Brilhante J, Zimmermann A (2014) Geoclouds modcs: a performability evaluation tool for disaster tolerant iaas clouds. In: 2014 8th annual IEEE systems conference (SysCon), pp 116–122. doi:10.1109/SysCon..6819245
38. Silva B, Maciel PRM, Tavares E, Zimmermann A (2013) Dependability models for designing disaster tolerant cloud computing systems. In: The third international workshop on dependability of clouds, data centers and virtual machine technology (DCDV)
39. Silva B, Maciel PRM, Zimmermann A (2013) Performability models for designing disaster tolerant infrastructure-as-a-service cloud computing systems. In: The 8th international conference for internet technology and secured transactions (ICITST)
40. Clark C, Fraser K, Hand S, Hansen JG, Jul E, Limpach C, Pratt I, Warfield A (2005) Live migration of virtual machines. In: Proceedings of the 2nd conference on symposium on networked systems design & implementation—volume 2, NSDI'05, pp 273–286, Berkeley, CA, USA. USENIX Association. <http://dl.acm.org/citation.cfm?id=1251203.1251223>
41. Kim DS, Machida F, Trivedi KS (2009) Availability modeling and analysis of a virtualized system. In: 15th IEEE Pacific Rim international symposium on dependable computing, 2009, PRDC'09, IEEE, pp 365–371
42. Cisco (2012) Cisco systems: switch dependability parameters. <http://tinyurl.com/cr9nssu>
43. Cisco (2012) Cisco systems: router dependability parameters. <http://tinyurl.com/d7kcnqo>
44. Mathis M, Semke J, Mahdavi J, Ott T (1997) The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput Commun Rev* 27(3):67–82. ISSN 0146–4833

45. Matthews W, Cottrell L, Logg C (1996) Tutorial on internet monitoring and pinger at SLAC. Technical report, Stanford. <http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html>
46. Silva B, Callou G, Tavares E, Maciel P, Figueiredo J, Sousa E, Araujo C, Magnani F, Neves F (2012) Astro: an integrated environment for dependability and sustainability evaluation. In: Sustainable computing: informatics and systems, 2012. ISSN 2210–5379. doi:[10.1016/j.suscom.2012.10.004](https://doi.org/10.1016/j.suscom.2012.10.004)
47. Callou G, Figueiredo J, Oliveira D, Ferreira J, Dantas J, AL Vandi Alves, Silva B, Matos R, Maciel P (2015) Mercury: an integrated environment for performance and dependability evaluation of general systems. In: 2015 IEEE/IFIP 45st international conference dependable systems networks (DSN), IEEE
48. Silva B, Maciel PRM, Tavares EAG, Araujo C, de Almeida Callou GR, Sousa E, Rosa NS, Marwah M, Sharma RK, Shah A, Christian T, Pires JP (2010) Astro: a tool for dependability evaluation of data center infrastructures. In: SMC'10, pp 783–790
49. German R, Kelling C, Zimmermann A, Hommel G (1995) Timenet: a toolkit for evaluating Non-Markovian Stochastic Petri Nets. *Perform Eval* 24(1–2):69–87
50. Gong C, Liu J, Zhang Q, Chen H, Gong Z (2010) The characteristics of cloud computing. In: 2010 39th international conference on parallel processing workshops (ICPPW), IEEE, pp 275–279
51. Clark C, Fraser K, Hand S, Hansen JG, Jul E, Limpach C, Pratt I, Warfield A (2005) Live migration of virtual machines. In: Proceedings of the 2nd symposium on networked systems design & implementation-volume 2, pp 273–286. USENIX Association
52. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. *J Internet Serv Appl* 1(1):7–18
53. Liu Y, Ma Y, Han JJ, Levendel H, Trivedi KS (2005) A proactive approach towards always-on availability in broadband cable networks. *Comput Commun* 28(1):51–64. ISSN 0140–3664
54. Grottke M, Matias R, Trivedi K (2008) The fundamentals of software aging. In: Proceedings of 1st international workshop on software aging and rejuvenation (WoSAR), in conjunction with 19th IEEE international symposium on software reliability engineering, Seattle. ISBN 978-1-4244-3416-9
55. Matos R, Araujo J, Alves V, Maciel P (2012) Characterization of software aging effects in elastic storage mechanisms for private clouds. In: IEEE 23rd international symposium on software reliability engineering workshops (ISSREW), pp 293–298
56. Melo M, Araujo J, Matos R, Menezes J, Maciel P (2013) Comparative analysis of migration-based rejuvenation schedules on cloud availability. In: 2013 IEEE international conference on systems, man, and cybernetics (SMC), IEEE, pp 4110–4115
57. Melo M, Maciel P, Araujo J, Matos R, Araujo C (2013) Availability study on cloud computing environments: live migration as a rejuvenation mechanism. In: 2013 43rd annual IEEE/IFIP international conference on dependable systems and networks (DSN), IEEE, pp 1–6
58. Guimaraes AP, Oliveira HMN, Barros R, Maciel PRM (2011) Availability analysis of redundant computer networks: a strategy based on reliability importance. In: 2011 IEEE 3rd international conference on communication software and networks (ICCSN), pp 328–332. doi:[10.1109/ICCSN.2011.6014733](https://doi.org/10.1109/ICCSN.2011.6014733)
59. German R, Kelling C, Zimmermann A, Hommel G (1995) Technische Universitt Berlin, and Fachgebiet Prozedatenverarbeitung Und Robotik. Timenet—a toolkit for evaluating Non-Markovian Stochastic Petri Nets. *Perform Eval* 24:69–87
60. Marsan MA, Chiola G (1987) On petri nets with deterministic and exponentially distributed firing times. In: Rozenberg G (ed) *Advances in Petri Nets 1987*, volume 266 of lecture notes in computer science, pp 132–145. Springer, Berlin, Heidelberg. ISBN 978-3-540-18086-9. doi:[10.1007/3-540-18086-9_23](https://doi.org/10.1007/3-540-18086-9_23). http://dx.doi.org/10.1007/3-540-18086-9_23
61. Trivedi KS (1982) *Probability and statistics with reliability, queuing, and computer science applications*. Prentice-Hall

Scalable Assessment and Optimization of Power Distribution Automation Networks

Alberto Avritzer, Lucia Happe, Anne Kozirolek, Daniel Sadoc Menasche,
Sindhu Suresh and Jose Yallouz

Abstract In this chapter, we present a novel state space exploration method for distribution automation power grids built on top of an analytical survivability model. Our survivability model-based approach enables efficient state space exploration in a principled way using random-greedy heuristic strategies. The proposed heuristic strategies aim to maximize survivability under budget constraints, accounting for cable undergrounding and tree trimming costs, with load constraints per feeder line. The heuristics are inspired by the analytical results of optimal strategies for simpler versions of the allocation problem. Finally, we parameterize our models using historical data of recent large storms. We have looked into the named storms that occurred during the 2012 Atlantic hurricane season as provided by the U.S. Government National Hurricane Center and numerically evaluated the proposed heuristics

A. Avritzer (✉) · S. Suresh
Siemens Corporation, Corporate Technology, 755 College Road East,
Princeton, NJ 08540, USA
e-mail: avritzer@icloud.com

S. Suresh
e-mail: sindhu-suresh@siemens.com

L. Happe · A. Kozirolek
Institute for Program Structures and Data Organization,
Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany
e-mail: lucia.kapova@kit.edu

A. Kozirolek
e-mail: anne.kozirolek@kit.edu

D.S. Menasche
Department of Computer Science, Federal University of Rio de Janeiro,
Av. Athos da Silveira Ramos, 274,
Rio de Janeiro (UFRJ), RJ 21941-916, Brazil
e-mail: sadoc@dcc.ufrj.br

J. Yallouz
Department of Electrical Engineering, Israel Institute of Technology (Technion),
Fishbach Building, Room 380, 32000 Haifa, Israel
e-mail: jose@tx.technion.ac.il

with data derived from our abstraction of the Con Edison overhead distribution power grid in Westchester county.

1 Introduction

The impact of superstorm Sandy on the New York metropolitan area has triggered several initiatives for investments in storm hardening of the Con Edison nonnetwork overhead system [15]. Nonnetwork or non-reliability regions are areas where reliability features as fully automated restoration are not yet implemented. The total cost of undergrounding the nonnetwork overhead power distribution in New York City and Westchester has been estimated at \$42.9 Billion [15]. Therefore, there is a need to develop an optimization approach to select the most cost effective investments while minimizing the impact of large storms on customers.

In this chapter, we assess the performance of greedy [1] and randomized greedy heuristic [18] algorithms, when applied to the selection of investment alternatives for storm hardening of distributed automation power grids [2] to minimize the AENS metric (Average Energy Not Supplied) resulting from storm-related failures. The AENS metric is computed using a phased-recovery survivability model that is implemented as a Markov-chain with reward rates [2]. The reward rate associated to each state of the Markov-chain is the energy that is not supplied in that state for the duration of the recovery of the cyber-physical system. We introduce a new random-greedy heuristic for storm hardening and evaluate the approach introduced in this chapter using data from several storms [9].

There is a vast literature on the infrastructure upgrade problem accounting for survivability in the context of transport [5, 13, 16] and computer networks [22]. In transport networks, researchers usually consider additive metrics such as delay to assess the cost of a path as a function of the cost of the links. Computer networks usually consider the probability that a path is functional as a function of its constituent links, which is a multiplicative metric. In this chapter, we consider power networks and a weighted multiplicative metric when establishing the budget allocation problem.

The domain model of interest is a cyber-physical system modeled as an abstraction of the Con Edison overhead distribution power grid in Westchester county. This power grid consists of 154 auto-loops line feeders [9]. The electric power research institute is modeling distribution line storm hardening by conducting tests to assess how to physically harden the overhead structure [7]. Alternatives that can be used for storm hardening of the electric overhead distribution system are the addition of reclosers and sectionalizer switches, tree trimming, and selective undergrounding [15].

Optimal constrained security hardening of power grids has been presented in [1]. The authors present a dynamic programming solution to the maximization of overall network security under a fixed budget constraint. They propose a set of strategies with independent cost and coverage and show that the combinatorial aspects of the problem make the enumeration of all possibilities an NP-Hard problem. The authors assume that the optimal strategy decided for a particular substation at step i depends

only on the budget expended up to step i . In contrast, the decision points at step i in our optimization problem for selective undergrounding of 154 auto-loop line feeders are dependent not only on the budget expended up to step i but also on the impact of the selection on the objective function. In particular, the energy not supplied due to a physical failure in a distributed automation network depends on the distribution of repair time and the load demanded in the section affected. Our greedy heuristic takes advantage of known polynomial time approximations for the knapsack problem.

In [2], the authors introduced a scalable approach to model and analyze complex power distribution networks to evaluate investment alternatives for storm hardening of the nonnetwork overhead power system. There, heuristics have been developed to assess the impact of using reclosers and tie-switches for storm hardening. Specifically, superstorm Sandy wind gusts were used to build a survivability model of the Con Edison overhead distribution power grid in Westchester county.

Survivability is the ability of the system to sustain performance conditioned on the occurrence of a failure event [11]. More formally, survivability is defined as conditional performability. In this chapter, we present a novel approach for power distribution optimization accounting for survivability metrics.

The main contributions introduced in this chapter are the following:

Storm historical data: Incorporation of storm history into the failure model introduced in [2]. Engineering for storm hardening requires the characterization of failures. As extreme weather conditions are becoming more frequent in the Northeast of the United States, it is important to use a recent history of large storms. In this chapter, we considered the named storms that occurred during the 2012 Atlantic hurricane season as provided by the U.S. Government National Hurricane Center.

Analytical model and problem formalization: Presentation of an analytical survivability model and associated optimization problem formulation. The survivability-related metric (AENS) is minimized under budget constraints, using undergrounding, tree trimming costs, and load constraints per feeder line. We prove that a simple version of the optimization problem is NP-hard, which motivates the use of heuristics.

Investment recommendations: Application of survivability modeling based on analytical transient solution to assess investment alternatives to support FDIR (Failure Detection Isolation and Restoration) in distribution automation smart-grids. Whereas the state of the art in power systems optimization is based on detailed simulation of generation and transmission, our survivability model-based approach enables efficient state space exploration in a principled way using random-greedy heuristic strategies. We illustrate our results through the numerical evaluation of the proposed heuristics impact on AENS for the 2012 Atlantic hurricane season with data derived from our abstraction of the Con Edison overhead distribution power grid in Westchester county.

The chapter outline is as follows. In Sect. 2, we describe the overhead power grid distribution automation system, the modeling of multiple storms, and the wind gust map derived for superstorm Sandy. Section 3 presents alternative formalizations of the optimization problem. Section 4 describes the heuristics used for optimization. Section 5 contains the evaluation of the proposed heuristics using numerical data and Sect. 6 concludes.

2 Power Grid Distribution Automation System

In this section, we present (a) an overview of the proposed methodology, (b) the Con Edison non-network overhead system, (c) the incorporation of large storm history into the model, and (d) a description of storm hardening strategies being proposed to increase survivability of the nonnetwork overhead system.

Figure 1 shows an overview of the proposed methodology. Calligraphic symbols correspond to measurements obtained from traces. Each section of the power grid has its associated measurements. Let \mathcal{W} and \mathcal{L} be the historical wind gust and load (power demand) measurements, respectively. Let \mathcal{C} and \mathcal{D} be the distributed generation capacities and section lengths, respectively.

Given historical data about hurricanes, \mathcal{W} , we parameterize a failure model. The failure model yields the probability that each section of the power grid fails after a hurricane, and is affected by survivability-related investments. The failure model, together with a power grid logical model, is used to parameterize the survivability model. The parameterization also relies on the load of each section, \mathcal{L} , and distributed generation capacities, \mathcal{C} . The survivability model outcome, in turn, leads to new investments. The cost of the investments is a function of the section lengths, \mathcal{D} , and is restricted by a budget B . The investments will affect the failure model, which will be used to issue new investment recommendations. The cycle repeats until a satisfactory result is obtained or the budget is met.

In Sects. 2.2 and 2.3, we describe the power grid logical model and the hurricane model. The failure model together with hardening strategies and costs are introduced in Sect. 2.4. The survivability-related heuristics for investments, inspired by the formal results presented in Sect. 3, are presented in Sect. 4. Section 4.4 of [2] contains the methodology used to parameterize the survivability model. The survivability model was introduced in [2], and is presented in Appendix A for completeness.

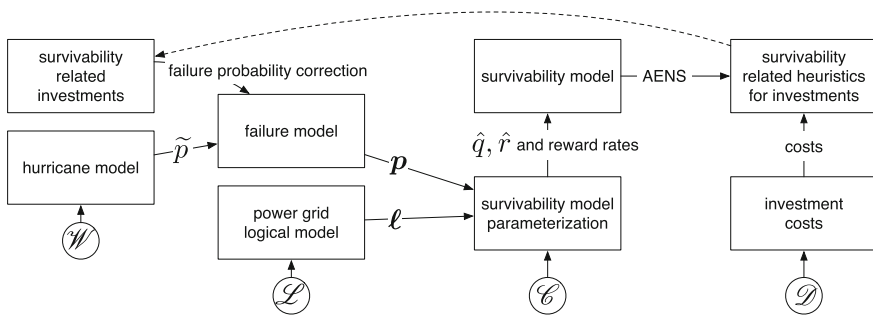


Fig. 1 Overview of methodology

2.1 Abstraction of the Con Edison Nonnetwork Overhead System

The system under study is an abstract model based on the Con Edison overhead distribution power grid [9]. The network consists of 154 auto-loops and 219 substations that supply power to Westchester County, Staten Island, as well as parts of the Bronx, Brooklyn, and Queens. This model of the Con Edison nonnetwork overhead system was developed by capturing the most important aspects required to assess the impact of large storms on the distribution power grid. Each main feeder line is modeled as a sequence of sections with the following data associated with it: (1) \mathcal{L}_s —average load demanded at section s , (2) \mathcal{C}_s —distributed generation capacity at section s , (3) \mathcal{W}_{si} —maximum wind gust observed at section s for storm i , and (4) \mathcal{D}_s —length of section s in miles.

The length of overhead power lines in each county is reported in [9]. In our study, we assume that in each county the number of loops is proportional to the length of overhead power lines, as we did not have access to the exact number of loops in each county. The segmentation of each auto-loop is sampled from an interval of a minimum of 8 and a maximum of 12 sections. The segmentation interval is based on the loop description given by [9]. The load for the sections are based on the load profile introduced in [19]. To build a sampling interval for wind gust values of hurricane Sandy, we took measured maximum wind gust in locations close to the Con Edison overhead power grid as described in [15].

2.2 Power Grid Logical Model

The logical power grid model considered in this chapter is illustrated in Fig. 2. Figure 2a presents a single feeder line, which connects a substation to a tie switch. It comprises upstream and downstream sections, separated by failed sections. In

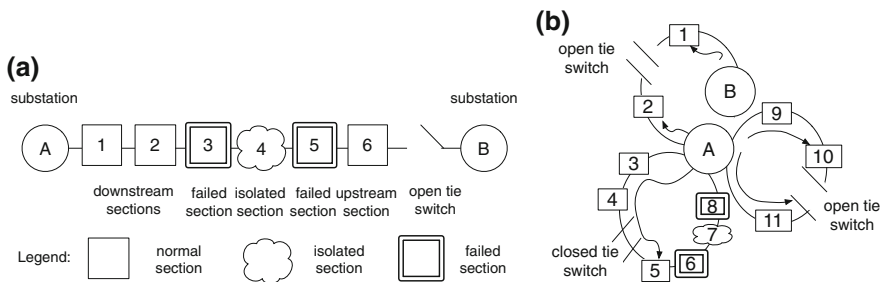


Fig. 2 **a** A feeder line connects a substation to a tie switch, and comprises upstream and downstream sections, separated by failed sections; **b** A failure at Sect. 6 causes a tie switch to close so as to feed Sect. 5. Power flows are represented by arrows

the power grid logical model, two logical substations might correspond to the same physical substation in the real power plant. For instance, substations A and B in Fig. 2a could correspond to one single physical substation of an auto-loop. Figure 2b shows how power flows in the network. At each section, power can only flow in one direction at any point in time. After a failure, reclosers might be closed so as to feed sections that otherwise would be disconnected.

2.3 Large Storms History

Until now, we considered investment alternatives for storm hardening against the impact of superstorms such as Sandy. However, the engineering of storm hardening strategies requires characterization of the largest storm which the electric utility is planning to harden the network against. To evaluate optimality of a hardening strategy, it is important to use a history of large storms in the observed region. We scaled our approach to aggregate data about the storms that occurred in the 2012 Atlantic hurricane season as provided by the U.S. Government National Hurricane Center [14].

We incorporated a large storm history into the model to support evaluation of different investment alternatives under fixed budget constraints. In the dataset considered, 14 of 19 storms achieved wind gust of less than 80 knots. As such, we assume the most effective investment before the 2012 Atlantic hurricane season would be to harden the network against the 74 % percentile storm (e.g., Sandy is a storm of 100 % strength), which would correspond to a wind strength of 80 knots in our dataset. In the following, we denote the wind strength observed for the 74 % percentile storm in section s as ω_s (or simply ω when referring to a typical section).

The effect of wind gust on overhead power lines has been investigated by several researchers. For the effect of flying debris from trees on overhead line fragility, Winkler et al. [21] used the model from Canham et al. [6], which identified a logistic regression model for effect of storm severity on the probability of wind-throw, for example. Canham's model yields the probability of wind-throw \tilde{p} based on species-specific parameters \tilde{a} , \tilde{b} , \tilde{c} , the diameter of a tree \tilde{d} , and the normalized observed storm severity S in their data, which ranges between 0 (lowest observed storm) and 1 (highest observed storm),

$$\tilde{p} = \frac{\exp(\tilde{a} + \tilde{c}S\tilde{d}^{\tilde{b}})}{1 + \exp(\tilde{a} + \tilde{c}S\tilde{d}^{\tilde{b}})} \quad (1)$$

As historical data of the effect on flying debris on probability of section failure is not available for the Con Edison network, we assume the following parameters in this work: $\tilde{a} = -2.261$, $\tilde{b} = 0.426$, $\tilde{c} = 1.140$ (as given by Canham et al. [6] for red maple), $\tilde{d} = 40$ and map the 74 % percentile of our observed wind gusts to S so that $\omega = 10$ knots correspond to $S = 0$ and $\omega = 80$ knots correspond to $S = 1$.

The resulting mapping is $S = (\omega - 10)/70$. Additionally, because we do not have data about the exact number of trees in the different sections, this work assumes a uniform number of trees in each section and uses the resulting probability as the failure probability per section. Note that more sophisticated models mapping observed maximum wind gust to probability of failure of a section could be used in our approach as well, such as a model taking into account the actual density of growth, the length of the section, the terrain, and the fragility of lines and trees.

2.4 Storm Hardening Strategies

The most effective strategies for storm hardening of nonnetwork overhead system are undergrounding and tree trimming. However, because of the large investments required for undergrounding feeder lines [15], the tradeoffs between undergrounding and tree trimming need to be carefully assessed. In addition, as the length and load of sections vary greatly, electric utilities must conduct systematic evaluations of the investment alternatives for storm hardening before large investments are committed to the storm hardening effort. The phase recovery model introduced in [2] and the optimization approach introduced in Sect. 4 can be employed to support these required studies.

In [12], the 4-year Con Edison 1 billion dollar program to protect New Yorkers from the next major storm is presented. The most important aspects of the plan are: (1) to ensure that new equipments located in the NY flood zone be flood proof, (2) mandate that new connections be located above flood level, and (3) ensure that new nonnetwork distribution equipments be integrated with Con Edison storm hardening strategies. The storm hardening strategies to be implemented by Con Edison 4-year plan are: (1) increase the number of sections and reduce the length of feeder line segments to reduce the customer impact of feeder line failures, (2) add tie-switches to improve feeder line design by allowing power to be fed from both ends of the feeder line, and (3) structural reinforcement of nonnetwork overhead poles. For example, structural poles will be required to withstand wind gusts of up to 110 mph. Other survivable structural designs are also being considered to ensure faster repair times of nonnetwork overhead components.

Wood poles can lose their structural integrity by decay leading to high failure probability during large storms [4]. Therefore, it is recommended that power utilities conduct regular tests for wood poles [4]. The objective of this test program is to ensure that the structural integrity of the poles under test will not fail the strength requirement (e.g., measured strength shall not be less than 30 % of designed strength), anytime before the next scheduled inspection.

To improve the survivability of the network described in Sect. 2.1, we consider two types of investments used by Con Edison [10], namely (1) more aggressive tree trimming in selected sections to prevent branches falling onto the overhead lines and (2) placing selected sections underground. Undergrounding costs about \$8.2 million per mile [10]. For tree trimming, we assume costs of \$210 per mile when using

mechanical tree trimming equipment [20]. Normally, Con Edison trims the trees of a section every 2 years [8]. In our more aggressive tree trimming investment option, we consider one additional tree trimming per year for a section. To compare the more aggressive tree trimming investment with the one-time undergrounding investment, we consider the net present value of the tree trimming investment as a perpetuity at an interest rate of 1%. Then, the annual investment of trimming one mile has a net present value of \$21,000. We assume that all sections have tree growth with uniform density and trimming effort, but this is just an assumption of the case study data, not of the approach in general.

The effect of tree trimming is a reduced probability of failure per section. Con Edison trims trees close to 27 and 33 kV distribution lines more often than trees close to 4 and 15 kV distribution lines, which indicates a higher risk of failure if trees are trimmed less often [8]. In this chapter, we assume that tree trimming yields a shift of the original failure probability by 4 knots, so that after tree trimming $S = \max(0, (\omega - 14)/70)$.

3 Formal Problem Definition

In this section, we consider simplified versions of the combinatorial optimization problem to be addressed in Sect. 4. Accordingly, we aim to (a) show that even under simplified assumptions the problem is already NP-hard and (b) illustrate further simplifications and alternative formulations that yield tractable analysis motivating the heuristics evaluated in Sect. 5. Generally, the optimization problem goal is to allocate a given budget in order to maximize the expected energy supplied under a failure occurrence. The allocation process consists of two main decisions, namely which equipment to invest in and where to install the investment.

The topology considered consists of auto-loops, where each loop comprises two feeder lines (also referred to simply as *lines*). Let N be the number of feeder lines. Each line starts at a source and ends at a sink, and comprises M sections. We assume that lines are independent of each other. Each section j of line n is associated with two parameters, namely a probability of failure $p_{n,j}$ and a load $\ell_{n,j}$. If a section k fails, all sections between k and the sink of that line are not supplied. Therefore, a failure on a specific section affects succeeding sections in the direction of the sink. Let $u_{n,j} \in \{0, 1\}$ be the indicator variable that characterizes whether an investment is applied in the j -th section of line n . Let $q_{n,j}$ be the price of an investment in section j of line n , whereas $h_{n,j}$ is the additive increase in the probability of success due to the investment. Table 1 summarizes the previous definitions.

Note that in this section, we assume that there is only one path from the source to each node in the line. Therefore, if a section fails all upstream sections also fail and cannot be fed by any backup substation or alternative path. In our numerical evaluations, we remove this assumption and assume that the upstream sections can be fed by alternative routes if available.

Let U be the system utility,

Table 1 Table of notation

Variable	Description
U	Expected energy supplied
$\ell_{n,j}$	Load at section j of line n
B	Budget
$h_{n,j}$	Increase in success probability after investing in section j of line n
$q_{n,j}$	The price of an investment in section j of line n
$u_{n,j}$	1 if investment at section j of line n is made, 0 otherwise
$p_{n,j}$	Probability that section j of line n fails

$$U = \sum_{n=1}^N \sum_{i=1}^M \frac{\ell_{n,i}}{\sum_n \sum_i \ell_{n,i}} \varphi \left(\prod_{j=1}^i (1 - p_{n,j} + h_{n,j} u_{n,j}) \right) \quad (2)$$

$\varphi(x)$ is a function which yields the revenue per unit of supplied load, i.e., the utility, of a set of sections as a function of the probability that all sections in the set are functioning. When $\varphi(x) = x$, U is the normalized expected energy supplied after a failure; the numerator of U corresponds to the non-normalized expected energy supplied after a failure and can also be obtained from Eq. (8) in [2].

We proceed to define the following optimization problem.

Definition 1 *Survivability Budget Allocation Power Distribution Problem (SBAPDP)*: Given a budget B , for each section and line, find an indicator variable assignment $\{u_{n,j}\}$ such that:

$$\max U \quad (3)$$

$$s.t. \quad \sum_{n=1}^N \sum_{j=1}^M q_{n,j} u_{n,j} \leq B \quad (4)$$

$$u_{n,j} \in \{0, 1\} \quad (5)$$

$$h_{n,j} \leq p_{n,j} \quad (6)$$

In most cases, we consider the expected energy supplied with $\varphi(x) = x$. We also study the case $\varphi(x) = \log(x)$ which yields weighted proportional fairness. Let \tilde{U} and \hat{U} be obtained from (2) by setting $\varphi(x) = x$ and $\varphi(x) = \log(x)$, respectively.

$$\hat{U} = \sum_{n=1}^N \sum_{i=1}^M \frac{\ell_{n,i}}{\sum_n \sum_i \ell_{n,i}} \sum_{j=1}^i \log(1 - p_{n,j} + h_{n,j} u_{n,j}) \quad (7)$$

The logarithmic utility function is used in the proof of Theorem 1, while the optimization in Sect. 4 considers a linear utility. The former inspires heuristics for the latter.

3.1 Hardness of Budget Allocation

We next establish that the optimal budget allocation problem is NP-hard.

Theorem 1 *The budget allocation problem is NP-hard.*

Proof We show that an instance of knapsack can be reduced to an instance of the budget allocation problem. To this end, as in [13], we let $c_{n,j}$ be the cumulative expected benefit of an investment. The cumulative expected benefit is given by

$$c_{n,j} = \sum_{i=j}^M \frac{\ell_{n,i}}{\sum_n \sum_i \ell_{n,i}} (\log(1 - p_{n,j} + h_{n,j}) - \log(1 - p_{n,j})) \quad (8)$$

Therefore, we can reformulate the problem as

$$\max \sum_n \sum_j c_{n,j} u_{n,j} \quad (9)$$

$$s.t. \sum_{n=1}^N \sum_{j=1}^M q_{n,j} u_{n,j} \leq B \quad (10)$$

$$u_{n,j} \in \{0, 1\} \quad (11)$$

The problem above is an instance of the knapsack optimization. Given an instance of the knapsack with budget B , where each item (n, j) has utility $c_{n,j}$ and cost $q_{n,j}$, solving the optimization problem above generates a solution to the knapsack problem. Therefore, the budget allocation problem is NP-hard. \square

Note that for the knapsack problem the greedy strategy is optimal as long as the goods are infinitesimally divisible (fractional knapsack [17]). In reality, we do not have infinitesimally divisible goods. The smaller generation options tend to be less efficient than the larger ones. Nonetheless, in the remainder of this section our results rely on this assumption. Our aim is to motivate heuristics which are analyzed in our numerical experiments.

3.2 Infinitesimally Divisible Investments

The main reason for the intractability of the SBAPDP problem is the fact that the allocation vector $\{u_{n,j}\}$ is integer. We proceed to show that the relaxation of the integrality assumption by considering an infinitesimal version of the SBAPDP problem results in a well-known convex optimization problem formulation, namely *geometric programming*.

Let $x_{n,j}$ be the investment applied in the j -th section of line n . Moreover, assume that $f_{n,j}(x_{n,j})$ be a polynomial function representing the additive increase in the

probability of success of the j -th section of the line n due to investment $x_{n,j}$. For simplicity of presentation, in the remainder of this chapter we assume $f_{n,j}(x_{n,j}) = x_{n,j}$, i.e., the additive increase in the probability of success due to investment is given by $x_{n,j}$. We proceed to formalize the following optimization problem:

$$\max \sum_{n=1}^N \sum_{i=1}^M \frac{\ell_{n,i}}{\sum_n \sum_i \ell_{n,i}} \prod_{j=1}^i (1 - p_{n,j} + x_{n,j}) \quad (12)$$

$$s.t. \sum_{n=1}^N \sum_{j=1}^M x_{n,j} \leq B \quad (13)$$

$$x_{n,j} \leq p_{n,j} \quad (14)$$

The problem above is an instance of a geometric program [3] which can be easily transformed into convex problems. The field of geometric programming has been studied for several decades. Once a problem can be defined as an instance of geometric programming, an optimal solution for the problem can be found by employing convex optimization methods. We note that convex optimization methods often employ numerical methods, while only a few problems can be solved analytically. Therefore, we omit a specific solution for the above problem from this discussion.

We proceed to consider the special homogeneous case where all sections initially support the same survivability conditions. Without loss of generality, we assume that the j -th section of line n is fully faulty, i.e., $p_{n,j} = 1$. For this special case, which we named SBAPDP HG, we provide the following insight about the optimal solution.

Theorem 2 *The optimal solution for the SBAPDP HG problem is such that $x_{n,j} \geq x_{n,i}, \forall j < i$.*

Proof We begin by rewriting the numerator of the objective function (12) in a simplified form

$$\sum_{n=1}^N \sum_{i=1}^M \ell_{n,i} \prod_{j=1}^i x_{n,j} = \sum_{n=1}^N x_{n,1} (\ell_{n,1} + x_{n,2} (\ell_{n,2} + x_{n,3} (\ell_{n,3} + \dots x_{n,M} \ell_{n,M})))$$

Assume for the sake of contradiction that there is an optimal solution such that $x_{n,j} < x_{n,i}$ for $j < i$, i.e., $x_{n,i} = x_{n,j} + \delta$. Next, we show that if this were the case we could increase the value of the objective function by increasing $x_{n,j}$ by δ and

decreasing $x_{n,i}$ accordingly. Let $\tilde{x}_{n,j} = x_{n,j} + \delta$, $\tilde{x}_{n,i} = x_{n,i} - \delta$ and $\tilde{x}_{n,k} = x_{n,k}$ for $k \neq i$ and $k \neq j$. Then,

$$\begin{aligned} & \sum_{n=1}^N x_{n,1}(\ell_{n,1} + \dots x_{n,j}(\ell_{n,j} + \dots (x_{n,j} + \delta)(\ell_{n,i} + \dots x_{n,M}\ell_{n,M}))) \\ & < \sum_{n=1}^N x_{n,1}(\ell_{n,1} + \dots (x_{n,j} + \delta)(\ell_{n,j} + \dots x_{n,j}(\ell_{n,i} + \dots x_{n,M}\ell_{n,M}))) \end{aligned}$$

Therefore, $\sum_{n=1}^N \sum_{i=1}^M \ell_{n,i} \prod_{j=1}^i x_{n,j} < \sum_{n=1}^N \sum_{i=1}^M \ell_{n,i} \prod_{j=1}^i \tilde{x}_{n,j}$ which contradicts our assumption that $\{x_{n,j}\}$ is optimal. \square

According to Theorem 2, sections closer to substations (associated to smaller indices) should receive more investment ($x_{n,j}$ is nondecreasing as a function of index j). Theorem 2 motivates heuristics proposed in Sect. 4 prioritize investment at sections closer to substations.

Discussion of Analytical Results

Theorems 1 and 2 naturally yield heuristics that we explore in our numerical evaluations. According to Theorem 1, the budget allocation problem can be mapped into the knapsack problem. For the knapsack problem, a greedy strategy consists of selecting, at each step, the option with maximum benefit-cost ratio. In case we have infinitesimally divisible goods, the greedy strategy is optimal. This motivates our greedy heuristic to maximize the ratio of investments benefit over cost at each step. Accordingly, Theorem 2 motivates allocating resources at the sections closer to substations. We numerically investigate this and other heuristics in our experimental evaluations.

4 Optimization Approach

In this section, we present a new random-greedy heuristic to minimize AENS under budget constraints, using undergrounding, tree trimming costs and load constraints per feeder line. As the budget is limited, we choose a local search approach starting from the given distribution network. We first discuss the individual heuristics used to select the feeder line and section in which to invest, and the investment decision to make in the selected section. Then, we describe the algorithm composed from these heuristics.

4.1 Investment Heuristics

Next, we introduce the investment heuristics considered in this chapter.

4.1.1 Select Section Within Feeder Line

Within a feeder line, we use static heuristics to apply investments. For undergrounding, our heuristic starts undergrounding at the substation, at section 1 of each feeder line (in accordance to Theorem 2). Thus, the load of the now underground section can no longer be affected by storms. If sections other than the first are put underground first, these would still be subject to storm damage in case the preceding vulnerable overhead sections fail. The tree trimming investment is applied for all (remaining) overhead sections of a feeder line. As machinery has to be moved to the loop, the trimming of single individual sections is not efficient. Thus, when deciding for the next investment during the local search, the first *overhead* section of a given feeder line a is considered and its index is denoted o_a .

4.1.2 Select Feeder Line

To select the feeder line $a \in A$ to invest in, we approximate the cost-benefit ratio of the investment. Let \mathbf{p}_a be the current failure probabilities for feeder line a with M_a sections, $\mathbf{p}_a = (p_{a,1}, \dots, p_{a,M_a})$. An investment x changes the failure probabilities. We let $\mathbf{p}_a(x)$ denote the changed vector of failure probabilities. Let ℓ_a denote the vector of load values of feeder line a , $\ell_a = (\ell_{a,1}, \dots, \ell_{a,M_a})$. In the following paragraph, we consider a given feeder line a and drop the subscript a from the variables considered.

The load at section i after a failure depends on the failure probability of the section itself and all preceding sections, with expected value given by $\ell_i \prod_{j=1}^i (1 - p_j)$. The expected load in a feeder line immediately after a failure is given by $L(\mathbf{p}, \ell)$,

$$L(\mathbf{p}, \ell) = \sum_{i=1}^M \ell_i \prod_{j=1}^i (1 - p_j) \tag{15}$$

Note that after a failure occurs, backup paths require a setup time before they can be used. $L(\mathbf{p}, \ell)$ is the expected load supplied exclusively by primary paths immediately after a failure at the feeder line considered. $L(\mathbf{p}(x), \ell)$ is the corresponding expected load after investment x . Then, the expected gain of an investment x in terms of how much more load will be connected to the primary substation after a failure for feeder line a is

$$gain(x, a) = L(\mathbf{p}_a(x), \ell_a) - L(\mathbf{p}_a, \ell_a) \tag{16}$$

Additionally, we consider the cost of the investment, which is calculated based on section o_a 's length and the subsequent section lengths. The cost of undergrounding a section o_a is denoted $c_u(o_a)$. The cost for annually trimming trees in section o_a and all subsequent sections of feeder line a is denoted $c_t(o_a)$ (see Sect. 2.4). Then, we

rank all feeder lines based on the expected benefit-cost ratio of the investment. The benefit-cost ratio of investment x for $x \in \{u, t\}$ is calculated as follows:

$$g_x(a) = \text{gain}(x, a) / c_x(o_a) \quad (17)$$

4.1.3 Select Investment

Based on the current configuration of the network, we generate and evaluate two new configurations.

1. Undergrounding is applied in the feeder line a_u that has the maximum expected benefit-cost ratio, i.e., that maximizes g_u . Consequently, section o_{a_u} is placed underground. Note that undergrounding can be applied multiple times to each feeder line. At each application, one additional section is undergrounded.
2. Tree trimming is applied to all sections of the feeder line a_t that has the maximum expected benefit-cost ratio, i.e., that maximizes g_t . For each feeder line, we assume that only one investment in tree trimming is possible. Thus, the tree trimming investment, which consists of performing one additional tree trimming per year in all overhead sections of a feeder line, is applied at most once to each feeder line.

For both investments, the AENS and cost are evaluated. Then, our heuristic selects the most efficient investments, i.e., the investment that minimizes AENS/cost.

4.2 Optimization Algorithms

Based on the heuristics described above, we implemented a greedy hill climbing algorithm in MATLAB, denoted GRDY in the following and explained below. The algorithm pseudocode is shown in Appendix B, and the numbers below refer to the lines in Algorithm 1.

The main input of GRDY is the network model, characterized by a the set of auto-loop feeder lines, each with a list of sections. Each section comes with the data described in Sect. 2.1. Additionally, a budget B and a survivability requirement τ can be specified to restrict the search to interesting regions. The budget can be set to infinity and the survivability requirement to zero to get a full search. The output is a sequence of network models, one per algorithm iteration.

In each iteration of the algorithm (between lines 4 and 20 in Algorithm 1), the heuristics are applied as follows. Both investments are considered, tree trimming (denoted t in the algorithm) and undergrounding (denoted u in the algorithm). Let x be the current investment, $x \in \{t, u\}$. The algorithm first selects the feeder line that maximizes $g_x(a)$ as described in Eq. (17) (line 8). If multiple feeder lines have the same benefit-cost ratio, the first one in the underlying data structure is taken. The

selected feeder line is denoted $a(x)$. Next, the failure probabilities of the sections of the selected feeder line are reduced in the function `APPLYINVESTMENT` (line 10). In the case of the Con Edison network, this means that the failure probability of underground sections is set to zero and the failure probability of the sections with trimmed trees is set according to Eq. (1) with $S = \max(0, \omega_s - 14)/70$, i.e., based on the 74% percentile wind gust ω_s observed for the section s in the considered storm. The tree trimming investment is only applied once per feeder line at maximum and undergrounding is only applied once per section at maximum.

Given two sets A and B , let $A \setminus B$ denote the set difference of A and B . Let $(A \cup \{a'(i)\}) \setminus \{a(i)\}$ be the adjusted network model after adding the updated feeder line $a'(i)$ and after removing the old version, $a(i)$. Then, the survivability of the adjusted network model is evaluated in function `EVALUATESURVIVABILITY`, using the probability of section failure and loads as input. The Markov model of Fig. 4 yields the average expected energy not supplied (AENS). Finally, the costs of an investment are calculated (line 13). As introduced above, $c_x(s)$ denotes the costs of applying investment x to section s . Then, `EVALUATECOST(A)` is defined as follows,

$$\text{EVALUATECOST}(A) = \sum_{a \in A} \left(\sum_{s \in \mathcal{S}_a(A)} c_u(s) + \sum_{s \in \mathcal{T}_a(A)} c_t(s) \right) \quad (18)$$

where

$$\mathcal{S}_a(A) = \{s \in a : s \text{ underground in } A\} \quad (19)$$

and

$$\mathcal{T}_a(A) = \{s \in a : s \notin \mathcal{S}_a \text{ and } s \text{ has trees trimmed in } A\}. \quad (20)$$

The two candidate networks created (one by placing a section underground, one by adding tree trimming) are compared based on the benefit-cost ratio (line 16 in Algorithm 1). The one that maximizes the benefit-cost ratio is taken as a basis for the next iteration (line 18). The algorithm terminates if the survivability result meets the given requirements, reaches 0, or if the budget is met (line 20).

Additionally, to avoid local optima, we implement a randomized greedy version (denoted `RNDGRDY`) of the proposed algorithm. `RNDGRDY` uniformly at random selects (a) the feeder line to invest in from the top 5 feeder lines based on expected benefit-cost ratio (line 8) and (b) the investment to apply (tree trimming or undergrounding, line 16). In the following section, we compare `GRDY` against `RNDGRDY`.

5 Evaluation

This section presents our numerical results for the two algorithms presented when applied to the Con Edison network described in Sect. 2.1. Figure 3 shows the results of running different algorithm variants (`GRDY` once, `RNDGRDY` three times). Each

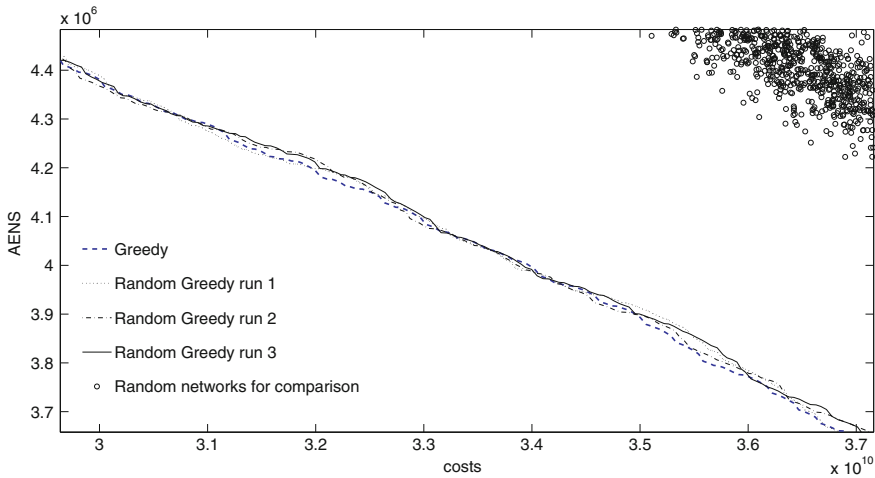


Fig. 3 Comparison of algorithm variants: average energy not supplied versus cost

run took about 1 h 10 min on a single 3 GHz core of a standard notebook machine. In each run, 1856 candidate networks were generated and their AENS and costs were evaluated.

We observe that the GRDY variant and the RNDGRDY variant perform quite similarly. In some portions of the search space, GRDY performed slightly better than our three runs of the RNDGRDY. In some other parts (not all shown in Fig. 3), a random run performed slightly better, but all four runs perform similarly overall.

As a reference for comparison, we also evaluated alternative investments (shown as circles in Fig. 3) which use the static heuristic to select the section to invest within a feeder line, i.e., always the first sections that can fail are placed underground and if trees are trimmed all trees in a feeder line are trimmed. The decisions about in which line to invest, and whether to perform undergrounding or tree trimming, are selected uniformly at random. Figure 3 illustrates the gains obtained as a consequence of exploring the state space in a principled way.

The heuristics evaluated in this section allowed us to explore the state space to find solutions which balance costs and survivability. The precise study of these solutions, accounting for stability and power flow analysis, must be performed in order to guarantee their feasibility. We envision the methodology introduced in this chapter as a first step to assist in the planning of a network.

6 Conclusion

In this chapter, we have introduced a new scalable methodology to assess the survivability impact of storm hardening investments. We have developed an abstraction of

the Con Edison overhead system using number of loops, sections, loads, and available distributed generation. We have also developed a failure model for the hurricane impact on the overhead system that estimates the section failure probability as a function of the maximum wind gust produced by the hurricane in a certain location. As a case study, we considered the series of named storms for 2012 Atlantic hurricane season.

We have shown that the budget allocation problem in the context of survivability optimization of distribution automation networks is NP-hard. However, as the budget allocation problem can be mapped into the knapsack problem, in the case of infinitesimally divisible goods, the greedy strategy that selects at each step the maximum improvement in the ratio of survivability improvement over cost would be optimal. Therefore, we were motivated to evaluate the performance of greedy and random-greedy heuristics.

Our evaluations of investment alternatives for storm hardening, which were based on the abstraction of Con Edison overhead system, shows that investment selections based on our greedy and random-greedy heuristics perform significantly better than randomly selected investments. Our heuristics are efficient and scalable as they use analytical solutions of the phased-recovery model for each circuit evaluation. We are encouraged by the efficiency and practicality of our approach as demonstrated by these initial results. As topics for future research, we envision the implementation of these heuristics into distribution automation optimization tools that could be used by power electric utilities to plan for storm hardening investments by taking into account different topologies, the impact of distributed generation, demand response, and failure detection and restoration features.

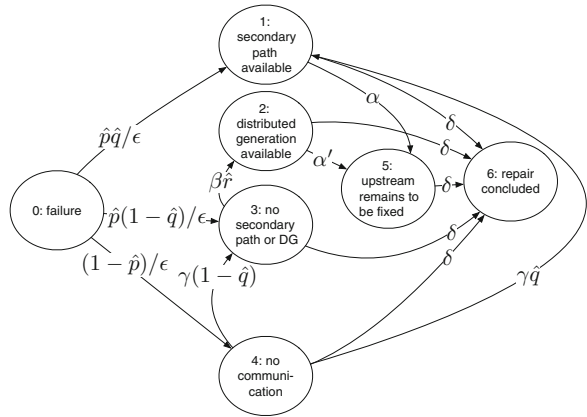
Appendices

Appendix A: Survivability Model

We describe the survivability model already presented in [2], which is one of the building blocks of the proposed power distribution optimization approach. Figure 4 illustrates the model.

As shown in Fig. 4, after a failure, the system transitions to one of three states depending on the availability of backup power, distributed generation, and communication. If communication is available, the system transitions to state 1 if backup power from a backup substation is available and to state 3 if distributed generation (e.g., from solar panels) will be tried. These transitions occur with rates $\hat{p}\hat{q}/\varepsilon$ and $\hat{p}(1 - \hat{q})/\varepsilon$, respectively. From state 1, the system is amenable to automatic restoration, which produces a transition to state 5. In state 5, manual repair is required in order to finalize the repair procedure. Manual repair occurs with rate δ . From state 3, distributed generation is activated with rate $\beta\hat{r}$. In case of success, the system transitions to state 2. In state 2, automatic repair occurs at rate α' . Note that in case

Fig. 4 Phased-recovery model characterizes recovery after failure in a set of sections of a line



communication is not available after the failure, it is restored with rate γ , and the system transitions to states 1 if a secondary backup path is available and to state 3 otherwise. In any other state, the system is also amenable to manual repair, which occurs at rate δ .

We use the model in Fig. 4 to compute the survivability metric of interest, namely average energy not supplied (AENS). To this end, we parametrize the model using the constants presented in Table 2. The default values of the parameters are set based on expert knowledge. Note that \hat{q} and \hat{r} are additional parameters of the model, which are derived from the probabilities of section failure as explained in [2, Sect. 4.4]. In order to compute the expected energy not supplied, to each state prior to repair in the model we associate reward rates, which are the expected energy not supplied per time unit at that state. These reward rates are derived from the probabilities of section failure and the section loads as explained in [2, Sect. 4.4]. Thus, the inputs

Table 2 Table of parameters

Parameter	Description	Value (1/h)
$\alpha' = \alpha$	Automatic restoration rate	30
δ	Manual repair rate	0.25
γ	Communication restoration rate	1
β	Automatic repair rate	4
$1/\epsilon$	Detection rate	∞
\hat{p}	Probability that communication is working after failure	0.5
Variable	Description	
\hat{r}	Probability that automatic use of DG is effective	
\hat{q}	Probability that secondary path is available after failure	

to the survivability model for use in this chapter are the probability of failure in each section and the average load in each section.

Appendix B: Algorithm Pseudocode

Algorithm 1 shows the pseudocode of the optimization solution discussed in Sect. 4.2.

```

input : network model, i.e. set of auto-loop feeder lines  $A$ 
         budget  $B$ 
         survivability requirement  $\tau$ 
output: sequence of network models  $T = (t_0, t_1, \dots)$ 
1  $j \leftarrow 1$ ; // iteration counter
2  $t_0 \leftarrow A$ ;
3  $I \leftarrow \text{EVALUATESURVIVABILITY}(A)$ ; // initial
4 repeat
5   //  $t$  = tree trimming;  $u$  = undergrounding
6   for  $x \in \{t, u\}$  do
7     // select feeder line  $a$  with highest expected benefit-cost ratio  $g_x(a)$ 
8      $a(x) \leftarrow \text{argmax}_{a \in A} (g_x(a))$ ;
9     // apply selected investment to feeder line, reducing failure probabilities
10     $a'(x) \leftarrow \text{APPLYINVESTMENT}(a(x), x)$ ;
11    // compute AENS using survivability model
12     $\text{AENS}(x) \leftarrow \text{EVALUATESURVIVABILITY}(A \cup \{a'(x)\} \setminus \{a(x)\})$ ;
13     $o(x) \leftarrow \text{EVALUATECOST}(A \cup \{a'(x)\} \setminus \{a(x)\})$ ;
14  end
15  // select investment to apply
16   $i \leftarrow \text{argmax}_{x \in \{t, u\}} (I - \text{AENS}(x)) / o(x)$ ;
17  // update power network model
18   $A \leftarrow (A \cup \{a'(i)\}) \setminus \{a(i)\}$ ;
19   $t_j \leftarrow A$ ;  $j \leftarrow j + 1$ ;
20 until  $(\text{AENS}(i) < \tau)$  or  $(o(i) \geq B)$ ;

```

Algorithm 1: Component allocation algorithm

References

1. Anwar Z, Montanari M, Gutierrez A, Campbell R (2009) Budget constrained optimal security hardening of control networks for critical cyber-infrastructure. Int J Crit Infrastruct Prot
2. Avritzer A, Suresh S, Koziolok A, Happe L, Menasche DS, Paollieri M, Carnevalli L (2014) A scalable approach to the assessment of storm impact in distributed automation power grids. In QUEST
3. Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge Univ Press
4. Brown R (2010) Storm hardening the distribution system. Quanta Technology

5. Campbell A, Lowe T, Zhang L (2006) Upgrading arcs to minimize the maximum travel time in a network. *Networks* 47(2):72–80
6. Canham CD, Papaik MJ, EF Latty (2001) Interspecific variation in susceptibility to windthrow as a function of tree size and storm severity for northern temperate tree species. *Can J Forest Res* 31(1):1–10
7. Cashman T (2013) Distributed line storm hardening
8. Conedison. Tree trimming faq. http://www.coned.com/publicissues/tree_trimming_faq.asp
9. Consolidated Edison of New York, Inc. (2013) Report on the preparation and system restoration performance. City of New York, Jan 2013
10. Consolidated Edison of New York, Inc. (2013) Storm hardening and resiliency collaborative report. City of New York, Dec 2013
11. Heegaard PE, Trivedi KS (2009) Network survivability modeling. *Comput Netw* 53(8):1215–1234
12. Mauldin P (2014) Storm hardening the grid. *Trans Distrib World*
13. Maya Duque P, Coene S, Goos P, Sørensen K, Spieksma F (2013) The accessibility arc upgrading problem. *Eur J Oper Res* 224(3):458–465
14. National Climatic Data Center (2014) National oceanic and atmospheric admin. <http://www.ncdc.noaa.gov>
15. Office of Long-Term Planning and Sustainability (2013) Utilization of underground and overhead power lines in the city of New York. City of New York, December 2013
16. Duque PM, Sørensen K (2011) A grasp metaheuristic to improve accessibility after a disaster. *OR spectrum* 33(3):525–542
17. Papadimitriou C, Steiglitz K (1998) *Combinatorial optimization*. Dover
18. Poloczeki M, Szegedy M (2012) Randomized greedy algorithms for the maximum matching problem with new analysis. *FOCS*
19. Rudion K, Orths A, Styczynski ZA, Strunz K (2006) Design of benchmark of medium voltage distribution network for investigation of DG integration. In: Power engineering society general meeting
20. Seitz J (1986) Rural mechanical tree trimming. *J Arboriculture*
21. Winkler J, Dueas-Osorio L, Stein R, Subramanian D (2010) Performance assessment of topologically diverse power systems subjected to hurricane events. *Reliab Eng Syst Safety* 95(4):323–336
22. Yallouz J, Orda A (2013) Tunable qos-aware network survivability. In: *INFOCOM*, pp 944–952

Model Checking Two Layers of Mean-Field Models

Anna Kolesnichenko, Anne Remke, Pieter-Tjerk de Boer
and Boudewijn R. Haverkort

Abstract Recently, many systems that consist of a large number of interacting objects have been analysed using the mean-field method, which allows a quick and accurate analysis of such systems, while avoiding the state-space explosion problem. To date, the mean-field method has primarily been used for classical performance evaluation purposes. In this chapter, we discuss model-checking mean-field models. We define and motivate two logics, called *Mean-Field Continuous Stochastic Logic* (MF-CSL) and *Mean-Field Logic* (MFL), to describe properties of systems composed of many identical interacting objects. We present model-checking algorithms and discuss the differences in the expressiveness of these two logics and their combinations.

1 Introduction

Present-day computational technologies are massive and can cope with a huge amount of data. However, for modelling or simulation of a large system of interacting objects this computational power is often not enough. The mean-field method can be used to model such large systems efficiently. This method [1, 2] does not consider the state of each individual object separately. Instead, only their average

A. Kolesnichenko (✉)

UL Transaction Security Division, De Heyderweg 2, 2314 XZ Leiden, The Netherlands
e-mail: anna.kolesni4enko@gmail.com

A. Remke

Department of Computer Science, University of Münster, Einsteinstrasse 62,
48149 Münster, Germany
e-mail: Anne.Remke@wwu.de

P.-T. de Boer · B.R. Haverkort

Department of Computer Science, University of Twente, P.O. Box 217,
7500 AE Enschede, The Netherlands
e-mail: p.t.deboer@utwente.nl

B.R. Haverkort

e-mail: b.r.h.m.haverkort@utwente.nl

© Springer International Publishing Switzerland 2016

L. Fiondella and A. Puliafito (eds.), *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering,
DOI 10.1007/978-3-319-30599-8_13

behaviour, i.e. the fraction of objects in each possible state at any time, is considered. The model obtained possesses two levels: (i) the *local* level describes the behaviour of a random individual; (ii) the *global* level addresses the overall model behaviour. On the global level, the mean-field model computes the exact limiting behaviour of an infinite population of identical objects, which is a good approximation when the number of objects is not infinite but sufficiently large. Examples of systems for which the mean-field method has been successfully applied include gossiping protocols [3], disease spread between islands [4], peer-to-peer botnets spread [5] and many more.

Thus far, the mean-field method was primarily used for classical system performance measures, however, also more involved measures might be of interest. Therefore, methods for efficient and automated *model-checking* of such non-trivial measures (or properties) are essential and non trivial. One challenge lies in the fact that the model has two layers. Therefore, it is desirable to be able to formulate properties on both levels. Another challenge is that the local model is a *time-inhomogeneous* Markov chain (ICTMC). Therefore, the results of the model-checking procedure depend on time. Finally, the state-space of the global mean-field model is infinite. Hence, finding the satisfaction set is difficult.

In this chapter we discuss two logics, namely *Mean-Field Continuous Stochastic Logic* (MF-CSL) and *Mean-Field Logic* (MFL) together with the corresponding model-checking algorithms. While MF-CSL was proposed in Ref. [6], MFL is a valuable addition, since it enables reasoning about timed properties on the global level. MF-CSL first expresses the property of a random node in a system (including timed properties) and then lifts this to the system level using *expectation* operators. In contrast, MFL expresses the property of the whole system directly and it does not take into account the behaviour of the individual objects.

The contribution of this chapter is to adapt the existing logic *Signal Temporal Logic* (STL) [7] to mean-field models. This is done by defining *global atomic properties* of the mean-field model, which define binary trajectories based on the real-valued behaviour of the mean-field model. Moreover, three ways to approximate the full satisfaction set of an MFL formula are discussed. One of the methods is specifically tailored to MFL and may be very computationally expensive. The two other methods are based on existing algorithms, proposed in Refs. [8, 9], where sensitivity analysis is used to partition the parameter set, which results in more efficient algorithms. We adapt these methods to approximate the satisfaction set of an MFL formula. This chapter compares the three methods and discusses available tools. Moreover, we compare the previously proposed logic MF-CSL to the logic MFL and motivate the existence of both. The possible combination of both logics is also discussed and illustrated by an example.

This chapter is further organized as follows. Section 2 discusses related work on model-checking mean-field models. In Sect. 3, a brief overview of the mean-field model and mean-field analysis is provided. The logic MF-CSL is described in Sect. 4, whereas Sect. 5 introduces MFL. The two logics are compared in Sect. 6. Section 7 concludes the chapter.

2 Related Work

Model-checking analyses whether a system state satisfies certain properties. It was initially introduced for finite deterministic models, for the validation of computer and communication systems, and later extended to stochastic models and models with continuous time. Checking models of large systems is complicated by the state-space explosion problem. Hence, model-checking mean-field models is considered a valuable continuation. For an overview, we refer the reader to Ref. [10].

The first work on model-checking mean-field models was presented in Refs. [6, 11]. Reference [11] presented first steps towards approximate model-checking of bounded CSL properties of an individual object (or group of objects) in a large population model. The Fast Simulation Theorem is used to characterise the behaviour of a single object via the average system behaviour, as defined by mean-field approximation. The proposed method is called *fluid model-checking*, which has been supplemented with next and steady-state operators in Ref. [12].

In contrast to fluid model checking, cf. [6], focuses on the properties of the whole population and proposes the logic MF-CSL. Such formulas consist of two layers, namely the local CSL formula, which describes the property of an individual object and a global *expectation* formula describing the fraction of objects, satisfying the local formula. The algorithm to check the until operator on the local level is based on the algorithm presented in Ref. [11]. An extra layer on top of local CSL properties allows the description of global properties of the whole system. We discuss this logic in more details later in this chapter.

Another two-layer logic is introduced in Ref. [13]. The time-bounded local properties are described using 1-global-clock Deterministic Timed Automata. Next, the global probability operator is introduced to estimate the probability that a certain fraction of local objects satisfies the local property, instead of the fractions of objects satisfying a certain property as in Ref. [6].

A different approach for model-checking mean-field models was proposed in Ref. [14]. There, the authors focus on the properties of an individual object, which is modelled as a discrete-time model in contrast to previously mentioned works. The, so-called, *on-the-fly* model-checking approach examines only those states that are required for checking a given property instead of constructing the entire state space.

Another way of looking at the mean-field model is to consider the behaviour of the whole system without addressing its local behaviour. Having in mind the representation of the global behaviour as a real-valued signal, one can use STL-like properties [7], as will be discussed further in this chapter. Moreover, a great effort has recently been made to enhance temporal properties with a real value, which is addressed as *quantitative semantics* or *robustness degree* [9, 15–17]. These methods can be successfully applied to mean-field models as well. The robustness of stochastic systems (including mean-field or fluid models) has been discussed in Ref. [18]. The design problem has also been addressed, where the parameters of the model can be optimized in order to maximize robustness. Several tools are available which allow calculating the robustness degree [19–21].

3 Mean-Field Models

The main idea of mean-field analysis is to describe the overall behaviour of a system that is composed of many similar objects via the average behaviour of the individual objects. In Sect. 3.1 we briefly recall the definition of the *local model*, which describes the behaviour of each individual object as well as how to build the *overall model* that describes the complete system. In Sect. 3.2, we then describe how to compute transient and steady-state occupancy measures using mean-field analysis.

3.1 Model Definition

Let us start with a random individual object which is part of a large population. We assume that the size N of the population is constant; furthermore, we do not distinguish between classes of individual objects for simplicity of notation. However, these assumptions can be relaxed, see, e.g., [22].

The behaviour of a single object can be described by defining the state space $S^l = \{s_1, s_2, \dots, s_K\}$ that contains the states or “modes” this object may experience during its lifetime, the labelling of the state space $L : S^l \rightarrow 2^{LAP}$ that assigns *local atomic propositions* from a fixed finite set of Local Atomic Properties (LAP) to each state and the transitions between these states.

In the following we will consider a large population of N objects, where each individual is modelled as described above, and denoted as \mathcal{M}_i for $i \in \{1, \dots, N\}$. Let us first try to preserve the identity of each object and build the model, describing the behaviour of N objects individually. It is easy to see that when the population grows linearly the size of the state space of the model grows exponentially. Fortunately, the mean-field approach allows modelling such a system of indistinguishable objects and avoids exponential growth of the state space (*state-space explosion*).

Given the large number of objects, where each individual is modelled by \mathcal{M} , we proceed to build the overall model of the whole population. We assume that all objects behave identically, therefore, we can move from the *individual* representation to a *collective* one, that does not reason about each object separately, but gives the number (or fraction) of individual objects in a given state of the model \mathcal{M} . It is done by taking the following steps:

Step 1. Lump the state space. When preserving the identity of the objects in a population $(\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_N)$ the sequence of the models of individual objects can be considered as a model of the population. However, the size of such sequence depends on N . Due to the identical and unsynchronized behaviour of the individual objects, a *counting abstraction* (or, stated differently, a transition from the individual to a collective representation) is used to find a smaller stochastic process, denoted as $\mathbf{M}^{(N)}$, of which the states capture the number of the individual objects across the states of the local model \mathcal{M} :

$$\mathbf{M}_j^{(N)} = \sum_{i=1}^N \mathbb{1}\{\mathcal{M}_i = j\}.$$

The state of $\mathbf{M}^{(N)}$ at time t is a counting vector $\overline{M}(t) = (M_1(t), M_2(t), \dots, M_K(t))$, where $M_i \in \{0, \dots, N\}$, and $\sum_{i=1}^K M_i = N$. The initial state is denoted as $\overline{M}(0)$.

Step 2. Defining transition rates. Given $\mathbf{M}^{(N)}$ and $\overline{M}(0)$ as defined above the Continuous-Time Markov Chain (CTMC) $\mathcal{M}^{(N)}(t)$ can be easily constructed. The transition rates are defined as follows [23]:

$$\mathbf{Q}_{i,j}(\overline{M}(t)) = \begin{cases} \lim_{\Delta \rightarrow 0} \frac{1}{\Delta} \text{Prob}\left\{ \mathcal{M}(t + \Delta) = j \right. \\ \left. \mathcal{M}(t) = i, \overline{M}(t) \right\}, & \text{if } M_i(t) > 0, \\ 0, & \text{if } M_i(t) = 0, \\ -\sum_{h \in S^l, h \neq i} \mathbf{Q}_{i,h}(\overline{M}(t)), & \text{for } i = j, \end{cases}$$

where $\mathcal{M}(t)$ indicates the state of the individual object at time t . The transition matrix depends on time via $\mathcal{M}(t)$.

Step 3. Normalize the population. For the construction of the mean-field model, which does not depend on the size of the population, the state vector is normalized as follows:

$$\overline{m}(t) = \frac{\overline{M}(t)}{N},$$

where $0 \leq \overline{m}_i(t) \leq 1$ and $\sum_{i=1}^K m_i = 1$.

When normalizing, first we have to make sure that the related transition rates are scaled appropriately. The transition rate matrix for the normalized population is given by:

$$Q(\overline{m}(t)) = \mathbf{Q}(N \cdot \overline{m}(t)).$$

Secondly, the initial conditions have to scale appropriately. This is commonly called *convergence of the initial occupancy vector* [24, 25]:

$$\overline{m}(0) = \frac{\overline{M}(0)}{N}.$$

The overall mean-field model can then be constructed as follows:

Definition 1 (Overall mean-field model) An overall mean-field model \mathcal{M}° describes the limit behaviour of $N \rightarrow \infty$ identical objects and is defined as a tuple (S°, Q) , that consists of an infinite set of states:

$$S^\circ = \left\{ \overline{m} = (m_1, m_2, \dots, m_K) \mid \forall j \in \{1, \dots, K\}, m_j \in [0, 1] \wedge \sum_{j=1}^K m_j = 1 \right\},$$

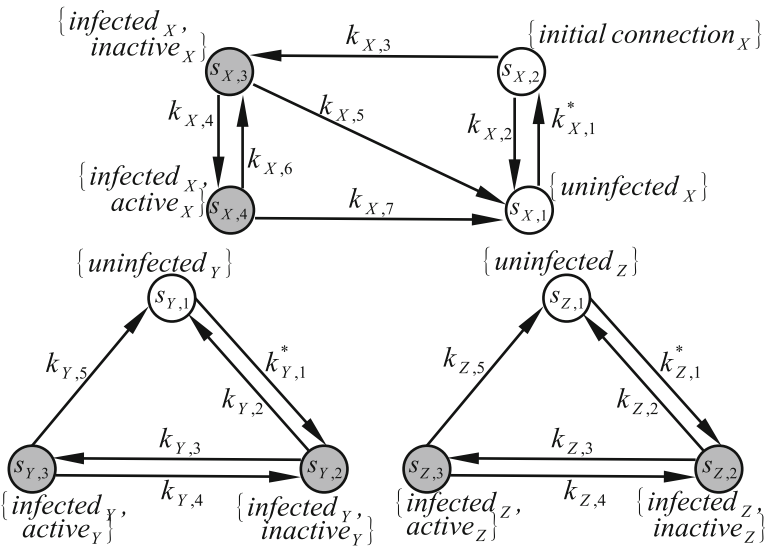


Fig. 1 The model describing computer virus spread in three groups of computers

where \bar{m} is called *occupancy vector*, and $\bar{m}(t)$ is the value of the occupancy vector at time t ; m_j denotes the fraction of the individual objects that are in state s_j of the model \mathcal{M} . The transition rate matrix $Q(\bar{m}(t))$ consists of entries $Q_{s,s'}(\bar{m}(t))$ that describe the transition rate of the system from state s to state s' .

Note that for any finite N the occupancy vector \bar{m} is a discrete distribution over K states, taking values in $\{0, \frac{1}{N}, \frac{2}{N}, \dots, 1\}$, while for infinite N , the m_i are real numbers in $[0, 1]$.

Example 1 We describe a model of virus spread in a system of interacting computers (see Fig. 1). We divide the whole computer system into three groups (e.g. different departments or geographical locations). We name these groups X, Y and Z. Each group has a fixed number of nodes (computers) N_X , N_Y and N_Z , respectively, where $N = N_X + N_Y + N_Z$. Communication between the groups is possible, but less probable than within a group. The system we model has three different locations and two different types of hardware (or software), where for each type the same computer virus would behave differently, i.e. computers in group X differ from the computers in groups Y and Z. The model of computer behaviour must take into account the possibility of being (i) in each of the three groups and (ii) being in each of the states of infection. Let us now consider the spread of the infection.

States represent the modes of an individual computer, which can be *uninfected*, *infected and active* or *infected and inactive* in all three groups; in addition, in group X a computer must first stay in the *initially connected* state before the virus is fully embedded in the computer system and the computer is infected. An *active* infected computer spreads the virus, while an *inactive* computer does not. Given this

information, the state-space of the local model must be extended to incorporate the possibility of being in each of the three groups. This results in the finite local state-space $S^l = \{s_{X,1}, s_{X,2}, s_{X,3}, s_{X,4}, s_{Y,1}, s_{Y,2}, s_{Y,3}, s_{Z,1}, s_{Z,2}, s_{Z,3}\}$, with $|S^l| = K = 10$ states, which are labelled as *infected_X*, *uninfected_X*, *initial connection_X*, *active_X* and *inactive_X*, etc., as indicated in Fig. 1.

The transition rates for computers in the first group are as follows: the infection rate $k_{X,1}^*$ is the rate for the initial connection; after that a computer must first try to pass the initially connected state with rate $k_{X,3}$ or return to the uninfected state with rate $k_{X,2}$. The recovery rate for an inactive infected computer is $k_{X,5}$, the recovery rate for an active infected computer is $k_{X,7}$, the rate with which computers become active is $k_{X,4}$ and they return to the inactive state with rate $k_{X,6}$. Rates $k_{X,2}, k_{X,3}, k_{X,4}, k_{X,5}, k_{X,6}$ and $k_{X,7}$ are specified by the individual computer and the properties of the computer virus and *do not* depend on the overall system state. The infection rate $k_{X,1}^*$ *does* depend on the rate of attack $k_{X,1}$, the fraction of computers that is infected and active and, possibly, the fraction of uninfected computers. The dependence on the overall system state is intended to reflect real-world scenarios and might be different for different situations. We discuss the infection rate of the current model further in this example.

Given a system of N computers, we can model the average behaviour of the whole system through the global mean-field model, which has the same underlying structure as the individual model (see Fig. 1), however, with state-space $S^o = \{m_{X,1}, m_{X,2}, m_{X,3}, m_{X,4}, m_{Y,1}, m_{Y,2}, m_{Y,3}, m_{Z,1}, m_{Z,2}, m_{Z,3}\}$, where $m_{X,1}$ denotes the fraction of uninfected computers in group X , $m_{X,2}$ represents the fraction of computers in the initially connected state, and $m_{X,3}$ and $m_{X,4}$ denote the fraction of active and inactive infected computers in group X , etc.

The infection rate can then be seen as the number of attacks performed by all active infected computers in group X , which is uniformly distributed over all uninfected computers in a chosen group, that is, $k_{X,1} \cdot \frac{m_{X,4}(t)}{m_{X,1}(t)}$. Note that we assume here that computer viruses are “smart enough” to only attack computers which are not yet infected, see [5, 26]. As discussed above, the computers from the different groups might interact with a certain probability. In the context of our virus spread model, this interaction plays a role when infected computers from groups Y and Z might contact uninfected computers in group X and vice versa. In this example model, we describe a virus which chooses one of the groups of computers with probability $p_{X,X}, p_{X,Y}, p_{X,Z}$ (for an infected computer from group X). The complete infection rates are composed by multiplying the above rates for one group with the probability of choosing a given group and accumulating all possible interactions between the three groups. Given the reasoning above, the infection rate in group X is

$$k_{X,1}^* = p_{X,X} \cdot k_{X,1} \cdot \frac{m_{X,4}(t)}{m_{X,1}(t)} + p_{Y,X} \cdot k_{Y,1} \cdot \frac{m_{Y,3}(t)}{m_{X,1}(t)} + p_{Z,X} \cdot k_{Z,1} \cdot \frac{m_{Z,3}(t)}{m_{X,1}(t)}.$$

The infection rates of groups Y and Z are constructed in a similar way.

For example, a completely healthy system without infected computers would be in state $\bar{m} = \left(\frac{N_1}{N}, 0, 0, 0, \frac{N_2}{N}, 0, 0, \frac{N_3}{N}, 0, 0 \right)$. A system with 50% *uninfected* computers and 40 and 10% of *inactive* and *active* computers in group X , and no infected computers in groups Y and Z would be in state:

$$\bar{m} = \left(0.5 \cdot \frac{N_1}{N}, 0, 0.4 \cdot \frac{N_1}{N}, 0.1 \cdot \frac{N_1}{N}, \frac{N_2}{N}, 0, 0, \frac{N_3}{N}, 0, 0 \right).$$

3.2 Mean-Field Analysis

Here we express a reformulation of Kurtz's theorem [27] which relates the behaviour of the sequence of models $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_N$ with increasing population sizes to the limit behaviour. We reformulate the theorem to make it more suitable for our purposes.

Before the theorem can be applied one has to check whether the overall mean-field model satisfies the following two conditions:

1. the model preserves the so-called *density dependence* condition in the limit $N \rightarrow \infty$ for all $N > 1$. This means that transition rates scale together with the model population, so that in the normalized models they are independent of the size of the population.
2. The rate functions are required to be Lipschitz continuous (informally it means that rate function are not too step).

When the three steps for constructing the mean-field model are taken and the above-mentioned conditions are satisfied, Kurtz's theorem can be applied, which can be reformulated as follows: *For increasing values of the system size ($N \rightarrow \infty$) the sequence of the individual models converges almost surely [28] to the occupancy vector \bar{m} , assuming that functions in $Q(\bar{m}(t))$ are Lipschitz continuous, and for increasing values of the system size, the initial occupancy vectors converge to $\bar{m}(0)$.* The above statement can be formally rewritten as in Ref. [23].

Theorem 1 (Mean-field convergence theorem) *The normalized occupancy vector $\bar{m}(t)$ at time $t < \infty$ tends to be deterministic in distribution and satisfies the following differential equations when N tends to infinity:*

$$\frac{d\bar{m}(t)}{dt} = \bar{m}(t) \cdot Q(\bar{m}(t)), \text{ given } \bar{m}(0). \quad (1)$$

The ODE (1) is called the *limit ODE*. It provides the results for the population of size $N \rightarrow \infty$, which is often an unrealistic assumption for real-life systems. However, when the number of objects in the population is finite but sufficiently large, the limit ODE provides an accurate approximation and the mean-field method can be successfully applied.

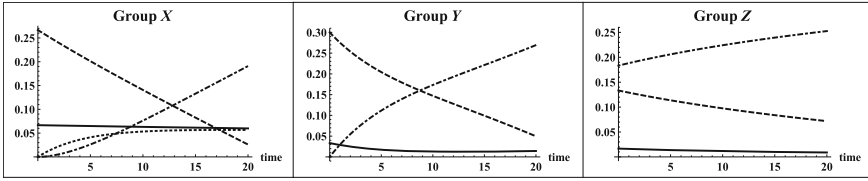


Fig. 2 Distribution of the computers over the states of the model for each group. The *dashed*, *dotted*, *dash-dotted* and *solid* lines show the fraction of uninfected, initially infected, infected inactive and infected active computers respectively

The transient analysis of the overall system behaviour can be performed using the above system of differential equations (1), i.e. the fraction of objects in each state of \mathcal{M} at every time t is calculated, starting from some given initial occupancy vector $\bar{m}(0)$, as illustrated in the following example.

Example 2 We provide an example, applying the mean-field method to the virus spread model, as in Example 1. We describe how to construct the mean-field model and obtain the performance results. The model below will be used throughout the chapter as a running example. The distribution of objects over three groups in this example is uniform, where $N_1 = N_2 = N_3 = \frac{N}{3}$.

Theorem 1 is used to derive the system of ODEs (1) describing the mean-field model. These ODEs are solved, given the following initial vector¹ $\bar{m}(0) = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0.1\}, \{0.4, 0.55, 0.05\})$, and the following parameters:

$$\begin{aligned}
 k_{X,1} &= 0.2, & k_{Y,1} &= 0.9, & k_{Z,1} &= 0.25, & p_{X,X} &= 0.93, \\
 k_{X,2} &= 0.01, & k_{Y,2} &= 0.005, & k_{Z,2} &= 0.001, & p_{Y,Y} &= 0.94, \\
 k_{X,3} &= 0.2, & k_{Y,3} &= 0.01, & k_{Z,3} &= 0.001, & p_{Z,Z} &= 0.97, \\
 k_{X,4} &= 0.0001, & k_{Y,4} &= 0.1, & k_{Z,4} &= 0.05, & p_{X,Y} &= 0.05, \\
 k_{X,5} &= 0.0001, & k_{Y,5} &= 0.06, & k_{Z,5} &= 0.001, & p_{X,Z} &= 0.02, \\
 k_{X,6} &= 0.005, & p_{Y,X} &= 0.05, & p_{Z,X} &= 0.02, \\
 k_{X,7} &= 0.005, & p_{Y,Z} &= 0.01, & p_{Z,Y} &= 0.01,
 \end{aligned}$$

The distribution of the objects between the states of the model over time (see Fig. 2) is obtained using Wolfram Mathematica [29].

In the following we discuss a couple of topics which lie beyond the convergence theorem discussed above. We first explain how the behaviour of individual objects within the overall population can be modelled. Second, possible relaxations of the assumptions made for this theorem are discussed. Then the behaviour in the stationary regime is briefly recalled.

Local model. The rates of the model for an individual object within the population may depend on the overall system state, which means that the local model is a *Time-Inhomogeneous Continuous-Time Markov Chain* (ICTMC). To formally describe

¹Note that for ease of interpretation, we group the elements of the vector according to the three submodels.

the behaviour of a single individual in the population the *asymptotic decoupling* of the system is used, and the result is often referred to as *Fast Simulation* [25, 30]. The main idea of this method lies in the fact that every single object (or group of objects) behaves independently from other objects, and can only sense the *mean* of the system behaviour, which is described by $\bar{m}(t)$. The model of one object within the population is called “local mean-field model” in the following and is defined as:

Definition 2 (*Local model*) A local model \mathcal{M}^l describing the behaviour of one object is defined as a tuple (S^l, \mathbf{Q}, L) that consists of a finite set of K local states $S^l = \{s_1, s_2, \dots, s_K\}$; an infinitesimal generator matrix $\mathbf{Q} : (S^l \times S^l) \rightarrow \mathbb{R}$; and the labelling function $L : S^l \rightarrow 2^{LAP}$ that assigns *local atomic propositions* from a fixed finite set of Local Atomic Properties (LAP) to each state.

Relaxing the assumptions. For models considered in practice the assumption of density dependence may be too restrictive [25]. Furthermore, also the assumption of (global) Lipschitz continuity of transition rates can be unrealistic [31]. Therefore, these assumptions can be relaxed and a more general version of the mean-field approximation theorem, having less strict requirements and which is applied to *prefixes* of trajectories rather than to full model trajectories, can be obtained. We will not focus on this reformulation of the convergence theorem here, instead we refer to [2].

Moreover, the mean-field approach has recently been expanded to a class of models with both Markovian and deterministically timed transitions, as introduced for *generalized semi-Markov processes* in Ref. [32]; and generally distributed timed transitions for *population generalized semi-Markov processes* [33]. In addition, an extension towards hybrid Markov population models has recently been made in Refs. [34, 35].

Stationary behaviour. The convergence theorem does not explicitly cover the asymptotic behaviour, i.e. the limit for $t \rightarrow \infty$. However, when certain assumptions hold, the mean-field equations allow to perform various studies including steady-state analysis. In the following we briefly recall how to assess the steady-state behaviour of mean-field models as in [36].

The stochastic process $(\mathbf{M}^{(N)})$, which was approximated by the mean-field model, has to be studied in order to find out whether the stationary distribution exists. It has been shown that, if the stochastic process is reversible, the fixed point approximation addressing the limiting behaviour of the overall mean-field model is indeed valid. Fixed point is an approximation of the stationary behaviour of the stochastic process by the stationary points of the mean-field (fluid) limit [36]. The reversibility of the stochastic process implies that any limit point of its stationary distribution is concentrated on stationary points of the mean-field limit. If the mean-field limit has a unique stationary point, it is an approximation of the stationary distribution of the stochastic process. The stationary distribution $\tilde{m} = \lim_{t \rightarrow \infty} \bar{m}(t)$, if it exists, then is the solution of:

$$\tilde{m} \cdot Q(\tilde{m}) = 0. \tag{2}$$

For some models the above equation can not be applied straightforwardly and more advanced methods are required in order to approximate the stationary distribution or its bounds [37]. This, however, lies outside of the scope of this chapter.

4 Overall System Properties. The Logic MF-CSL

In this section, we briefly recall the logic MF-CSL for checking properties of mean-field models, as introduced in Ref. [6]. MF-CSL is a two-layer logic, where (i) on the local level the property of a random object is specified; (ii) on the global level the fraction of objects satisfying this local property is expressed.

4.1 CSL and MF-CSL

Let us first recall how the properties of a random object can be expressed and checked. As discussed in Sect. 3, the model of one object in a mean-field system is an ICTMC. Therefore, the logic CSL [38] can be used to describe properties on the local layer.

Definition 3 (CSL Syntax) Let $p \in [0, 1]$ be a real number, $\bowtie \in \{\leq, <, >, \geq\}$ a comparison operator, $I \subseteq \mathbb{R}_{\geq 0}$ a non-empty time interval and LAP a set of atomic propositions with $lap \in LAP$. **CSL state formulas** Φ are defined by:

$$\Phi ::= tt \mid lap \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\phi),$$

where ϕ is a **path formula** defined as:

$$\phi ::= \chi^I \Phi \mid \Phi_1 U^I \Phi_2.$$

When the local model is time homogeneous, the semantics of CSL formulas is well known. However, in any non-trivial mean-field model, the transition rates of the local model \mathcal{M}^l are not constant. According to Definition 2, the rates of the local model \mathcal{M}^l may depend on the state of the global model \mathcal{M}^o , which changes with time. There are two ways to formalize this: the local rates depend on (i) the *current state* \bar{m} , which changes with time, or (ii) on *global* time. While the first is more intuitive, it does not allow transition rates to depend explicitly on global time. For ease of notation we restrict ourselves to models that only depend on the overall state. Note that this approach can easily be extended to models that explicitly depend on global time.

Since the local model changes with the overall system state, the satisfaction relation for a local state or path depends on the global state \bar{m} . Therefore, the satisfaction relation $\models^{\bar{m}}$, was introduced in Ref. [6]. Due to the lack of space we do not provide the extended semantics of CSL for the local model and refer to [6] for more details.

In order to reason at the level of the overall model in terms of fractions of objects we introduce an extra layer “on top of CSL” that defines the logic *CSL for mean-field models*, denoted MF-CSL.

Definition 4 (MF-CSL Syntax) Let $p \in [0, 1]$ be a real number, and $\bowtie \in \{\leq, <, >, \geq\}$ a comparison operator. MF-CSL formulas Ψ are defined as follows:

$$\Psi ::= tt \mid \neg\Psi \mid \Psi_1 \wedge \Psi_2 \mid \mathbb{E}_{\bowtie p}(\Phi) \mid \mathbb{ES}_{\bowtie p}(\Phi) \mid \mathbb{EP}_{\bowtie p}(\phi),$$

where Φ is a **CSL state formula** and ϕ is a **CSL path formula**.

This definition introduces three expectation operators: $\mathbb{E}_{\bowtie p}(\Phi)$, $\mathbb{ES}_{\bowtie p}(\Phi)$ and $\mathbb{EP}_{\bowtie p}(\phi)$, with the following interpretation: $\mathbb{E}_{\bowtie p}(\Phi)$ denotes whether the fraction of objects that are in a (local) state satisfying a general CSL state formula Φ fulfils $\bowtie p$; $\mathbb{ES}_{\bowtie p}(\Phi)$ denotes whether the fraction of objects that satisfy Φ in steady state, fulfils $\bowtie p$; $\mathbb{EP}_{\bowtie p}(\phi)$ denotes whether the probability that an object chosen uniformly at random satisfies path formula ϕ fulfils $\bowtie p$.

Example 3 In the following, we provide a couple of example MF-CSL formulas.

- No more than 5% of all computers are *infected_Y* (i.e. these computers belong to the group Y and are infected), is expressed as $\mathbb{E}_{\leq 0.05} \text{infected}_Y$.
- The percentage of all computers which happen to belong to group X and have a probability of less than 10% of going from an uninfected to an active infected state within 3 h is greater than 40%, i.e.

$$\mathbb{E}_{> 0.4} \left(\mathbb{P}_{< 0.1}(\text{uninfected}_X U^{[0,3]} \text{active}_X) \right).$$

- $\mathbb{EP}_{< 0.5}(tt U^{[0,2]} \text{infected}_Z)$ ensures a probability below 50% that a machine, chosen uniformly at random, is in group Z and becomes infected within two hours from now.
- In the long run a very low proportion (less than 2%) of all computer should be infected group Z machines: $\mathbb{ES}_{< 0.02} \text{infected}_Z$.

Definition 5 (MF-CSL Semantics) The satisfaction relation \models for MF-CSL formulas and states $\bar{m} = (m_1, m_2, \dots, m_K) \in S^o$ of the overall mean-field model is defined by:

$$\begin{aligned} \bar{m} &\models tt && \forall \bar{m} \in S^o, \\ \bar{m} &\models \neg\Psi && \text{iff } \bar{m} \not\models \Psi, \\ \bar{m} &\models \Psi_1 \wedge \Psi_2 && \text{iff } \bar{m} \models \Psi_1 \wedge \bar{m} \models \Psi_2, \\ \bar{m} &\models \mathbb{E}_{\bowtie p}(\Phi) && \text{iff } \left(\sum_{j=1}^K m_j \cdot \text{Ind}_{(s_j \models \bar{m}\Phi)} \right) \bowtie p, \\ \bar{m} &\models \mathbb{ES}_{\bowtie p}(\Phi) && \text{iff } \left(\sum_{j=1}^K m_j \cdot \pi^{\mathcal{M}^l}(s_j, \text{Sat}(\Phi, \bar{m})) \right) \bowtie p, \\ \bar{m} &\models \mathbb{EP}_{\bowtie p}(\phi) && \text{iff } \left(\sum_{j=1}^K m_j \cdot \text{Prob}^{\mathcal{M}^l}(s_j, \phi, \bar{m}) \right) \bowtie p, \end{aligned}$$

where $Sat(\Phi, \bar{m})$, $\pi^{\mathcal{M}}(s, Sat(\Phi, \bar{m}))$, $Prob^{\mathcal{M}^l}(s, \phi, \bar{m})$ are defined in the semantics of CSL formula [6]; and $Ind_{(s_j \models \bar{m} \Phi)}$ is an indicator function, which shows whether a local state $s_j \in S^l$ satisfies formula Φ for a given overall state \bar{m} :

$$Ind_{(s_j \models \bar{m} \Phi)} = \begin{cases} 1, & \text{if } s_j \models \bar{m} \Phi, \\ 0, & \text{if } s_j \not\models \bar{m} \Phi. \end{cases}$$

Although \bar{m} is referred to as the \bar{m} vector at time 0, this is only for ease of discussion, without loss of generality.

To check an MF-CSL formula at the global level, the local CSL formula has to be checked first, and the results are then used at the global level. As discussed above, the local model \mathcal{M}^l is a time-inhomogeneous CTMC, i.e. transition rates vary with the state of the overall model \mathcal{M}^o , which makes model-checking at the local level non-trivial. The full algorithms for checking CSL properties of the local model are based on the methods presented in Ref. [11], and have been adapted for the MF-CSL semantics in Ref. [6] according to Definition 5. These algorithms enable the identification of the satisfaction set of a CSL formula for a given initial occupancy vector (i.e. at time $t = 0$), as well as the time-dependent satisfaction set of a CSL formula for a given initial occupancy vector and time bound θ .

Traditionally, the satisfaction set of a given formula is the set of states which satisfies that formula. In the context of MF-CSL model-checking, this would result in the set of all occupancy vectors \bar{m} that satisfy a given formula. While such a set can be built for *time-independent* MF-CSL operators, it is not a trivial task for *time-dependent* operators, since model-checking on the local model \mathcal{M}^l must be done without knowing the initial occupancy vector. Theoretically speaking, partitioning of the whole state-space into sets which satisfy a given MF-CSL formula and the rest is possible, but it is computationally very expensive, as model-checking the local time-inhomogeneous CTMC is very demanding.

Since obtaining the full satisfaction set of an MF-CSL formula is not practically feasible, the notion of *Time Validity Set* of the MF-CSL formula for a given initial occupancy vector and time interval is defined in Ref. [6] as follows:

Definition 6 (*Time Validity Set*) Let $\theta > 0$ be a predefined time bound, Ψ be an MF-CSL formula, and initial conditions of the mean-field model \mathcal{M}^o be an occupancy vector $\bar{m}(0) = \bar{m}$. The *Time Validity Set* (TVS) contains all time intervals, where Ψ holds, for a given \bar{m} , and θ :

$$TVS(\Psi, \bar{m}, \theta) = \{t \in [0, \theta] \mid \bar{m}(t) \models \Psi\}.$$

The time validity set is a set of time intervals, where a given MF-CSL formula holds for fixed initial conditions. We will use this notion later in the chapter when discussing MFL and comparing the two logics. For the details of model-checking MF-CSL we refer to [6]. However, we provide an example that illustrates the general procedure of model-checking two-layer properties.

Example 4 We show how to check whether a given occupancy vector satisfies an MF-CSL formula (Case A), and how to compute the corresponding TVS (Case B) for the model provided in Example 1.3.2.

Case A. Let us consider the following formula

$$\Psi = \mathbb{E}P_{<0.2}(uninfected_Y U^{[0,3]} infected_Y)$$

and the occupancy vector $\bar{m} = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0.1\}, \{0.4, 0.55, 0.05\})$. In order to check the satisfaction relation $\bar{m} \models \Psi$, we first compute the satisfaction set of the until formula on the local level, and then check the satisfaction relation $\bar{m} \models \Psi$ using Definition 5. We use algorithms from [6] as follows.

To find the probability that the until formula holds, the following reachability problem must be solved: Is state *infected_Y* reached within 3 time units by only visiting *uninfected_Y* states? This is done by calculating transient probabilities on the modified local model, where all states satisfying *infected_Y*, and states in groups *X* and *Z* are made absorbing. The Kolmogorov equation is used to calculate the transient probability matrix of the modified local model. The transient (reachability) probabilities for state *s_{Y,1}* to reach states *s_{Y,1}* and *s_{Y,2}* are equal to 0.77 and 0.23, respectively. The probability to reach any state *s* starting at this state is, obviously, one. All other transient probabilities are equal to zero.

To compute the probability that the until formula $\phi = uninfected_Y U^{[0,3]} infected_Y$ holds for each starting state, we have to accumulate the probabilities to start at this state and end at the *infected_Y* state. Therefore, the probability that the until formula holds equals 0.23, since for all states, but *s_{Y,1}*, this probability equals zero. According to Definition 5, the weighted sum of the entries of the occupancy vector \bar{m} and the respective probabilities in the local model define the expected probability $\mathbb{E}P(\phi)$ as follows:

$$\sum_{j=1}^K m_j \cdot Prob^{\mathcal{M}^l}(s_j, \phi, \bar{m}) = m_{Y,1} \cdot 0.23 + m_{Y,1} \cdot 1.0 = 0.3 \cdot 0.23 + \frac{1}{30} \cdot 1 = 0.102.$$

Therefore the occupancy vector \bar{m} satisfies $\mathbb{E}P_{<0.2}(uninfected_Y U^{[0,3]} infected_Y)$.

Case B. Let us consider the same formula Ψ and occupancy vector \bar{m} and compute $TVS(\Psi, \bar{m}, 15)$ for $\theta = 15$. The calculation of the time-dependent probabilities $Prob^{\mathcal{M}^l}(s, uninfected_Y U^{[0,3]} infected_Y, \bar{m}, t)$ requires the initial transient probability (for time $t = 0$), as done in Case A. Next, the ODEs describing the time-dependent transient probability of the modified local model are constructed using both forward and backward Kolmogorov equations [6]. These equations are solved using the initial transient probability (as computed in Case A) as initial condition. The solution of these ODEs defines the required time-dependent reachability probabilities. The time-dependent probability that state *s_{Y,1}* satisfies the until formula is depicted in Fig. 3. The probability equals zero for all other starting states, since these states do not satisfy *uninfected_Y*. To calculate $TVS(\Psi, \bar{m}, 15)$ the time-dependent expected probability is calculated using the time-dependent probability that the until formula holds (see Fig. 3). Then, the time points where the expected probability equals 0.2

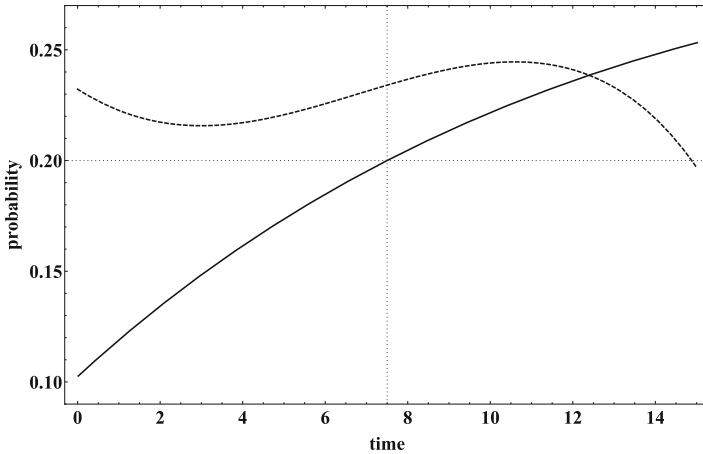


Fig. 3 The *dashed line (top)* shows $Prob^{\mathcal{M}^l}(s_{Y,1}, \text{uninfected}_Y U^{[0,1]} \text{infected}_Y, \bar{m}, t)$. The (increasing) *solid line* shows the time-dependent expected probability that the formula $\mathbb{E}\mathbb{P}_{<0.2}(\text{uninfected}_Y U^{[0,1]} \text{infected}_Y)$ holds. The *horizontal and vertical dotted lines* delimit the period for which the probability is below 0.2, and hence the formula $\mathbb{E}\mathbb{P}_{<0.2}(\text{uninfected}_Y U^{[0,1]} \text{infected}_Y)$ holds

are found. In the current example, this is only $t = 6.228$. The time validity set of the formula $\mathbb{E}\mathbb{P}_{<0.2}(\text{uninfected}_Y U^{[0,3]} \text{infected}_Y)$ then consists of all time intervals, where the probability $\mathbb{E}\mathbb{P}(\text{uninfected}_Y U^{[0,3]} \text{infected}_Y)$ is less than 0.2, i.e. $\text{TVS}(\Psi, \bar{m}, 15) = [0, 7.45)$.

5 Timed Properties of the Global Model: MFL

In the previous section, we introduced a way to express and check properties of the global mean-field model \mathcal{M}^O via properties of a random local object. The properties which MF-CSL can describe include CSL properties (possibly temporal) of the local model and the expected number of objects for a global model to satisfy this property. However, timed properties of the global model can not be expressed using MF-CSL, but can be beneficial for understanding the behaviour of large systems. For that purpose, the new *Mean-Field Logic* (MFL) is introduced in Sect. 5.1. Algorithms for model-checking MFL properties are presented in Sects. 5.2 and 5.3.

5.1 MFL Syntax and Semantics

To be able to express timed properties of the overall model, we adapt the existing Signal Temporal Logic (STL) which was developed to monitor discrete temporal properties of continuous signals [7, 39]. In order to do so, we address the solution

of the ODEs (1) for given initial conditions $\bar{m}(0)$ as a signal or a *trajectory* of the global/overall mean-field model \mathcal{M}^O . In the following, we express properties of real-valued trajectories of mean-field models $\bar{m}(t)$ using STL-like properties. We first introduce the mapping of the model trajectory to a Boolean signal.

Definition 7 (*Global atomic property*) An atomic property *GAP* of the global model is a characteristic function (Boolean predicate) $S^o \rightarrow \{0, 1\}$, from occupancy vector \bar{m} to a Boolean value.

Applying the concept of *GAP* to a given trajectory of a mean-field model $\bar{m}(t)$ results in a Boolean function of time $GAP(\bar{m}(t))$. In order to guarantee decidability, we require that the output Boolean trajectory $GAP(\bar{m}(t))$ has finite variability, i.e. the number of time points where $GAP(\bar{m}(t))$ changes value is finite; the output trajectory is a Boolean robust (cadlag²) function [7]. For simplicity, in the following we use inequalities of the form $\sum_{i \in N} a_i \cdot m_i \bowtie p$ as global atomic properties of mean-field models, where a_i is an indicator factor equal to 1 or 0. However, more advanced functions, satisfying the above requirements, can be used as *GAP*. Given the definition of a global atomic property, the syntax of *Mean-Field Logic* (MFL) can be introduced.

Definition 8 (*Syntax of MFL*) Let $I = [a, b]$, where $0 \leq a < b$, be a non-empty bounded time interval and function *GAP* defining global atomic properties. MFL formulas \mathcal{Y} are defined as:

$$\mathcal{Y} ::= tt \mid GAP \mid \mathcal{Y}_1 \wedge \mathcal{Y}_2 \mid \neg \mathcal{Y} \mid \mathcal{Y}_1 \mathcal{U}^I \mathcal{Y}_2.$$

We can define not only a property of the global model at a given time point but also the evolution of the model over time, as shown in the following example.

Example 5 We first start with the properties of the global model at a given time point (time-independent properties). To define such a property, *GAPs* are combined with the time-independent operators \neg and \wedge .

The following property describes a system in which the fraction of computers that belong to group Y and that are infected is smaller than 0.2:

$$\mathcal{Y}_1 = m_{Y,2} + m_{Y,3} < 0.2,$$

where $m_{Y,2}$ and $m_{Y,3}$ denote the number of infected computers in group Y (inactive and active respectively). The same property can be expressed using atomic properties of the local model as follows: $\text{infected}_Y < 0.2$. Here and in the following, we use labels of the local model instead of fractions in *GAP* to ease the interpretation of the formula.³ A more involved property can be defined by a conjunction of *GAPs*.

²A function is called *cadlag* if it is defined on the real numbers (or a subset of them), if it is everywhere right-continuous and if it has left limits everywhere.

³Note, however, a global atomic property is not always connected to the properties of the local model (unlike the expectation operator in MF-CSL).

For example, the property that a system has more than 20 % infected computers and less than 1 % active infected group-Z computers is formalized as:

$$\mathcal{Y}_2 = \text{infected} < 0.2 \wedge \text{active}_Z < 0.01.$$

Note that the first part of the property \mathcal{Y}_2 includes all infected computers in all three groups. The timed properties of the global system are constructed by combining GAPs (or other MFL formulas) using the until operator. The following property describes the system in which the fraction of computers which belong to group X and are infected is smaller than 0.1 at all times **until** in the time interval between 3 and 5 time units the fraction of computers that are members of group Z and active exceeds 0.4:

$$\mathcal{Y}_3 = (\text{infected}_Y < 0.1) \mathcal{U}^{[3,5]} (\text{active}_Z > 0.4).$$

Definition 9 (*Semantics of MFL*) The satisfaction relation \models for MFL state formulas and state $\bar{m} \in S^o$ is defined as:

$$\begin{aligned} \bar{m} &\models tt && \forall \bar{m} \in S^o, \\ \bar{m} &\models GAP && \text{iff } GAP(\bar{m}) = 1, \\ \bar{m} &\models \mathcal{Y}_1 \wedge \mathcal{Y}_2 && \text{iff } \bar{m} \models \mathcal{Y}_1 \text{ and } \bar{m} \models \mathcal{Y}_2, \\ \bar{m} &\models \neg \mathcal{Y} && \text{iff } \bar{m} \not\models \mathcal{Y}, \\ \bar{m} &\models \mathcal{Y}_1 \mathcal{U}^I \mathcal{Y}_2 && \text{iff } \exists t \in I : (\bar{m}(t) \models \mathcal{Y}_2) \wedge (\forall t' \in [0, t] \bar{m}(t') \models \mathcal{Y}_1), \end{aligned}$$

where $\bar{m} = \bar{m}(0)$ at time $t = 0$, and $\bar{m}(t')$ is a solution of the ODEs (1) at time $t = t'$ with \bar{m} as the initial condition.

The definition of the until formula is different from the usual representation [40], because it requires both \mathcal{Y}_1 and \mathcal{Y}_2 to hold at time t , in order to guarantee closure [7].

As was explained in the previous section, in this chapter we discuss mean-field models, where the dependency on time is only implicit (via $\bar{m}(t)$). Therefore, the entire model trajectory (the solution of the ODEs (1)) is defined through the current system state; the time when this state is reached does not influence the future behaviour of the system. The occupancy vector for which the satisfaction relation is checked is therefore denoted $\bar{m}(0)$, and the time intervals in the until formulas are relative. However, all the arguments and algorithms presented in this section can be generalized to models with an explicit time dependence. The next section overviews the algorithms used for checking MFL properties of mean-field models.

5.2 Checking an MFL Property

To check an MFL formula, its parse tree has to be built and all sub-formulas have to be checked recursively. Therefore, the algorithms for checking each individual operator have to be introduced. Checking a given occupancy vector \bar{m} against time-

independent operators is straightforward, which follows directly from Definition 9. However, MFL formulas containing the Until operator can not be checked that easily, since the behaviour of the system (trajectory) influences the result. Therefore, we introduce the notion of the *time validity set* for a given MFL formula, mean-field model and an occupancy vector, as done in the previous section for MF-CSL formulas (see Definition 6). It is easy to see that if the TVS($\mathcal{Y}, \bar{m}, \theta$) contains $t = 0$, the formula holds for \bar{m} . The TVS of a general MFL formula is again built recursively by finding the TVSs of sub-formulas. The computation of the TVS for the time-independent operators is straightforward:

$$\begin{aligned} \text{TVS}(tt, \bar{m}, \theta) &= [0, \theta], \\ \text{TVS}(\text{GAP}, \bar{m}, \theta) &= \{t \in [0, \theta] \mid \text{GAP}(\bar{m}(t)) = 1\}, \\ \text{TVS}(\neg\mathcal{Y}, \bar{m}, \theta) &= [0, \theta] \setminus \text{TVS}(\mathcal{Y}, \bar{m}, \theta), \\ \text{TVS}(\mathcal{Y}_1 \wedge \mathcal{Y}_2, \bar{m}, \theta) &= \text{TVS}(\mathcal{Y}_1, \bar{m}, \theta) \cap \text{TVS}(\mathcal{Y}_2, \bar{m}, \theta). \end{aligned}$$

Computing the TVS for the until operator ($\mathcal{Y}_1 \mathcal{U}^{[a,b]} \mathcal{Y}_2$) (with $0 \leq a < b$) is more challenging. The algorithm described in the following is based on the method of monitoring temporal properties as in Refs. [7, 39]; we refer to these papers for more details and proofs.

To compute the TVS for the until formula $\mathcal{Y} = \mathcal{Y}_1 \mathcal{U}^{[a,b]} \mathcal{Y}_2$ we first find the sets of time intervals where the sub-formulas \mathcal{Y}_1 and \mathcal{Y}_2 hold. Note that both sets may contain multiple intervals. Therefore we denote them as $\text{TVS}(\mathcal{Y}_1, \bar{m}, \theta) = v_1^1 \cup v_1^2 \cup \dots \cup v_1^{n_1}$, and $\text{TVS}(\mathcal{Y}_2, \bar{m}, \theta) = v_2^1 \cup v_2^2 \cup \dots \cup v_2^{n_2}$, respectively.

To calculate $\text{TVS}(\mathcal{Y}, \bar{m}, \theta)$ one must obtain all time intervals where $\mathcal{Y} = \mathcal{Y}_1 \mathcal{U}^{[a,b]} \mathcal{Y}_2$ holds. Hence, we search for time intervals where both \mathcal{Y}_1 and \mathcal{Y}_2 hold, since these are the time intervals, where the validity of the until formula can be confirmed, in case at least one such time interval lies between a and b . Recall that the time interval in the until formula is relative to the starting point. Therefore, to check whether a given vector fulfils the until formula, one must check whether the intersection interval can be reached from the vector within the predefined time interval $[a, b]$. Hence, to directly compute the set of all time points from which the formula can be fulfilled, we shift $\text{TVS}(\mathcal{Y}_1 \cap \mathcal{Y}_2, \bar{m}, \theta)$ backwards, i.e. move the left interval bound back with b and the right with a time units. This is defined for each pair of sub-intervals in $\text{TVS}(\mathcal{Y}_1, \bar{m}, \theta)$ and $\text{TVS}(\mathcal{Y}_2, \bar{m}, \theta)$ as a *backwards shift*, denoted as $\mathcal{BS}^{[a,b]}(v_1^i; v_2^j)$. For each pair of the intervals $(v_1^i; v_2^j)$, the backward shift is computed as follows:

$$\mathcal{BS}^{[a,b]}(v_1^i; v_2^j) = ((v_1^i \cap v_2^j) \ominus [a, b]) \cap v_1^i, \quad (3)$$

where $[x_1, x_2] \ominus [a, b] := [x_1 - b, x_2 - a] \cap [0, \infty)$. This backward shift can be understood as follows (from left to right):

1. The intersection $(v_1^i \cap v_2^j)$ defines all time points where both \mathcal{Y}_1 and \mathcal{Y}_2 are valid.
2. The \ominus -operation (or backwards shift) ensures that:

- a. the earliest starting point is taken such that after at most b time units one can reach a state where \mathcal{Y}_2 holds;
 - b. the latest starting point is taken such that one can still switch to a state in which \mathcal{Y}_2 holds for at least a time units.
3. The intersection with $v_1^{(i)}$ ensures that on the way to the state where \mathcal{Y}_2 holds, \mathcal{Y}_1 always holds.

After the backwards shift is applied to each pair $(v_1^i; v_2^j)$, the resulting intervals are then combined to find the TVS of the overall until formula:

$$\text{TVS}(\mathcal{Y}_1 \mathcal{U}^{[a,b]} \mathcal{Y}_2, \bar{m}, \theta) = \bigcup_{i=1}^{n_1} \bigcup_{j=1}^{n_2} \mathcal{B} \mathcal{S}^{[a,b]}(v_1^i; v_2^j). \quad (4)$$

In practice, only the pairs of intervals which actually intersect must be considered. Given the above, the TVS of any MFL formula can be found. After the TVS of the formula is found, we can validate whether a formula holds for a given initial occupancy vector \bar{m} by checking whether $t = 0$ lies in the TVS. Note that the TVS can also be seen as an independent measure of interest, if one is looking for the time slots where the system satisfies a given property, for a given initial state (as in the previous section). In the following, model-checking MFL formulas is illustrated by an example.

Example 6 We again address the model of Example 1, with the same parameters as given in Example 2. We explain in detail how to calculate the time validity set $\text{TVS}(\mathcal{Y}, \bar{m}(0), \theta)$ for both time-independent and time-dependent formulas, given $\bar{m}(0) = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0.1\}, \{0.4, 0.55, 0.05\})$, and $\theta = 25$. Next, we check whether $0 \in \text{TVS}(\mathcal{Y}, \bar{m}(0), \theta)$, which would indicate that the initial occupancy vector $\bar{m}(0)$ satisfies the formula.

Case A. We first consider the time-independent property, describing the situation in which the fractions of active computers in groups Y and Z are “sufficiently small”, i.e. the fraction of active infected computers in group Y is bounded by 0.015, and the fraction of active infected computers in group Z is at most 0.01:

$$\mathcal{Y}^A = (\text{active}_Y \leq 0.015) \wedge (\text{active}_Z \leq 0.01).$$

To check this property the following steps are taken:

1. The trajectory of the model is obtained by solving the ODEs given $\bar{m}(0)$.
2. The TVS of the sub-formulas $\mathcal{Y}_1^A = (\text{active}_Y \leq 0.015)$ and $\mathcal{Y}_2^A = (\text{active}_Z \leq 0.01)$ are calculated using a root finding procedure for $m_{Y,3}(t) = 0.015$ and $m_{Z,3}(t) = 0.01$:
 - $\text{TVS}(\mathcal{Y}_1^A, \bar{m}(0), \theta) = [6.79; 21.69]$;
 - $\text{TVS}(\mathcal{Y}_2^A, \bar{m}(0), \theta) = [15.14, 25]$ (see Fig. 4a).
3. The TVS of the whole formula then consists of all intervals, where both sub-formulas hold: $\text{TVS}(\mathcal{Y}^A, \bar{m}(0), \theta) = [15.14; 21.69]$ (see Fig. 4b).

Fig. 4 **a** TVS of $\gamma_1^A = (\text{active}_Y \leq 0.015)$ (solid line) and $\gamma_2^A = (\text{active}_Z \leq 0.01)$ (dashed line). **b** Intersection of $\text{TVS}(\gamma_1^A, \bar{m}(0), \theta)$ and $\text{TVS}(\gamma_2^A, \bar{m}(0), \theta)$

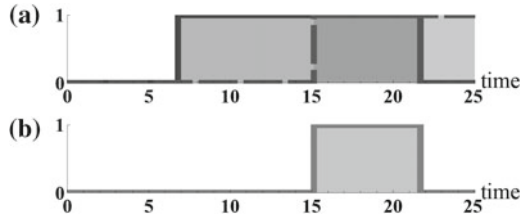
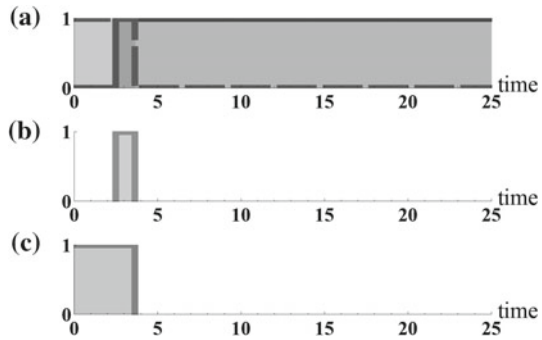


Fig. 5 **a** TVSs of $\gamma_1^B = (\text{active} \leq 0.05)$ (dashed line) and $\gamma_2^B = (\text{uninfected} \leq 0.6)$ (solid line). **b** Intersection of $\text{TVS}(\gamma_1^B, \bar{m}(0), \theta)$ and $\text{TVS}(\gamma_2^B, \bar{m}(0), \theta)$. **c** TVS of $\gamma^B = (\text{active} \leq 0.05) \mathscr{U}^{[0,3]} (\text{uninfected} \leq 0.6)$



4. The validity of the formula for a given initial vector is checked by verifying whether $t = 0$ lies in $\text{TVS}(\gamma^A, \bar{m}(0), \theta)$. It is easy to see that $0 \notin [15.14, 21.69]$, therefore $\bar{m}(0) \notin \gamma^A$.

Case B. We now consider a time-dependent property of the global model, which describes that the fraction of active infected computers in all three groups together (denoted as *active*) remains smaller or equal to 0.05 until within 3 time units the fraction of all uninfected computers becomes less or equal to 0.6. This property ensures that the virus is “quiet enough” and will not be detected until a sufficient number of computers in the system is infected:

$$\gamma^B = (\text{active} \leq 0.05) \mathscr{U}^{[0,3]} (\text{uninfected} \leq 0.6).$$

We first find the time validity sets for this formula and check whether the initial occupancy vector $\bar{m}(0)$ satisfies this property:

1. The time validity sets of the sub-formulas $\gamma_1^B = \text{active} \leq 0.05$ and $\gamma_2^B = \text{uninfected} \leq 0.6$ are calculated using a root finding procedure for $m_{X,3} + m_{Y,3} + m_{Z,3} = 0.05$ and $m_{X,1} + m_{Y,1} + m_{Z,1} = 0.6$; and are given as $\text{TVS}(\gamma_1^B, \bar{m}(0), \theta) = [0; 3.64]$ and $\text{TVS}(\gamma_2^B, \bar{m}(0), \theta) = [2.5; 25]$ (see Fig. 5a).
2. The intersection of $\text{TVS}(\gamma_1^B, \bar{m}(0), \theta)$ and $\text{TVS}(\gamma_2^B, \bar{m}(0), \theta)$ is $[2.5; 3.64]$ (Fig. 5b).
3. To find $\text{TVS}(\gamma^B, \bar{m}(0), \theta)$ the backwards shift is performed as in Eq. (4) $\text{TVS}(\gamma^B, \bar{m}(0), \theta) = [2.5 - 3; 3.64 - 0] = [0; 3.64]$ (see Fig. 5c). Note that the behaviour of the system in the past (before time $t = 0$) is not relevant. Therefore, the lower bound of the TVS is set to zero.

4. Formula $\mathcal{Y}^B = (\text{active} \leq 0.05) \mathcal{U}^{[0,3]}$ (uninfected ≤ 0.6) holds for $\bar{m}(0)$, since $0 \in \text{TVS}(\mathcal{Y}^B, \bar{m}(0), \theta)$.

Wolfram Mathematica [29] was used to calculate the results above. We compared them with results obtained from the Breach toolbox [19], which has been built to check STL properties, confirming that the results coincided.

5.3 Satisfaction Set of an MFL Formula

In this section, we discuss how to compute the complete satisfaction set of an MFL formula, which is formally defined as follows:

Definition 10 (*Satisfaction Set*) Given an MFL formula \mathcal{Y} and a mean-field model \mathcal{M}^O , the **satisfaction set** of an MFL formula consists of *all* occupancy vectors \bar{m} that satisfy \mathcal{Y} :

$$\text{Sat}(\mathcal{Y}) = \{\bar{m} \mid \bar{m} \models \mathcal{Y}\}.$$

The mean-field model has an infinite state-space. Therefore, the computation of the satisfaction set is not straightforward. The ultimate goal is to partition the state-space of the model S^O into two parts: (i) states satisfying a given formula, i.e. $\text{Sat}(\mathcal{Y})$ and (ii) states which do not satisfy \mathcal{Y} . Since exact methods for computing the satisfaction set of an MFL formula are not available, numerical (approximate) algorithms will be discussed in the following, which means that it might not always be possible to partition the state-space into two sets. Hence, a third set, namely, a set of *uncertain states*, may be necessary. In such cases, this third set should be as small as possible. The satisfaction set of a given formula is constructed recursively, by building and combining the satisfaction sets of sub-formulas.

5.3.1 Time-Independent Operators

The computation of satisfaction sets for operators which are not time dependent, is straightforward and does not imply any additional computations. It follows directly from Definition 9 and is formalized as follows:

$$\begin{aligned} \text{Sat}(tt) &= S^o, \\ \text{Sat}(GAP) &= \{\bar{m} \mid GAP(\bar{m}) = 1\}, \\ \text{Sat}(\mathcal{Y}_1 \wedge \mathcal{Y}_2) &= \text{Sat}(\mathcal{Y}_1) \cap \text{Sat}(\mathcal{Y}_2), \\ \text{Sat}(\neg\mathcal{Y}) &= S^o \setminus \text{Sat}(\mathcal{Y}). \end{aligned}$$

5.3.2 The Until Operator

The computation of the satisfaction set of the time-bounded until operator $\Upsilon_1 \mathcal{U}^{[a,b]} \Upsilon_2$ is not trivial and involves additional methods. We discuss three ways that can be used to calculate this set. Two of these methods are directly applicable to the complete MFL formula, and one is only suitable for a single until operator. Hence, the satisfaction sets of the time-independent sub-formulas must be computed separately (see Sect. 5.3.1).

Discretization of the state-space. One of the ways to approximate the satisfaction set of an MFL formula is to discretize the continuous state space and check the MFL formula for each point of the discrete state space obtained using the standard method (see Sect. 5.2). The discretization can be done, for example, by a grid-based approach. However, this approach is computationally intensive and produces only an approximation of the satisfaction set. Moreover, the quality of such an approximation and the computational demand depend directly on the granularity of the grid. Moreover, the complexity of the problem grows with the number of dimensions (local states). Although the method is applicable to any MFL formula, applying it to a model with a large local state-space to obtain high quality approximations is simply not feasible.

Solving two reachability problems. Another way to numerically develop the satisfaction set of a given until formula would be to divide the formula $\Upsilon_1 \mathcal{U}^{[a,b]} \Upsilon_2$ into two *reachability problems*, similar to standard methods, as e.g. in Ref. [38]: (i) starting from the states which satisfy Υ_1 , the trajectory evolves such that only states satisfying Υ_1 are visited during time interval $[0, a]$; (ii) starting from the states which satisfy Υ_1 and are reachable during the first step, the trajectory evolves such that only states satisfying Υ_1 are visited until a state satisfying Υ_2 is reached during time interval $[0, b - a]$. The reachability problems for mean-field models can be solved using techniques proposed in Ref. [8]. Their method partitions the parameter set of the ODE-based model into three sets, namely, (i) S_{goal} , which comprises all states that satisfy the reachability problem, (ii) S_{bad} , including all states which do not satisfy it, and (iii) S_{unc} , which combines all states for which reachability can not (yet) be decided. Instead of partitioning the parameter space, we use the proposed methods to partition the state-space of the mean-field model. Note, however, that this approach is only applicable for a single until operator. To compute the satisfaction set of an until formula, we need to solve the two reachability problems in reverse order. We first find all states from which we can reach an Υ_2 state in at most $(b - a)$ time units, while visiting only Υ_1 states. We denote the set of all these states as S'_{goal} . We then find the set of all states, denoted as S_{goal} , from which we can reach S'_{goal} , while visiting only Υ_1 states. The satisfaction set of the overall until formula then equals S_{goal} .

The general approach of the method given in Ref. [8] is as follows. The reachable set is found using *sensitivity analysis* and the satisfaction set is obtained by using a parameter synthesis algorithm based on *refining partition*, which iteratively refines the state-space of the mean-field model and assigns the subsets obtained to one of the three sets, namely: S_{goal} , S_{bad} , or S_{unc} by checking the reachability problem for this set. Each refinement introduces only subsets that are strictly smaller than the refined

set to guarantee that the process ends. The algorithm is designed to stop when the uncertain set is either empty or smaller than a predefined value.

The Breach Toolbox [19] was used to implement the above algorithm. However, a tool for the automated solution of the reachability problem is not available. Although the numerical algorithms in Ref. [8] can not provide formal guarantees on the correctness of the results, asymptotic guarantees exist. Therefore, results can always be improved by decreasing the tolerance factor in the numerical computations.

Robustness-based method. Another method to obtain the satisfaction set of an arbitrary MFL formula, including nested until operators, partitions the state-space based on refining partition algorithms. However, this approach requires introducing a *quantitative semantics* of MFL. This allows both Boolean and real values as a result of a model-checking algorithm ($\mathbb{R} \cup \{\top, \perp\}$). The result of the model-checking procedure shows that a given occupancy vector satisfies an MFL formula (in case the obtained value is greater than zero), and also estimates the quality of satisfaction. We introduce the quantitative semantics of MFL, similarly to [15], where a quantitative semantics was introduced for STL. For simplicity of notation, we use global atomic properties in the form $f(m_1, m_2 \dots m_K) \geq c$, where $c \in \mathbb{R}$.

Definition 11 (*Quantitative semantics*) Given an MFL formula \mathcal{Y} , a mean-field model \mathcal{M} , and initial occupancy vector \bar{m} , the quantitative semantics $\rho(\mathcal{Y}, \bar{m})$ is defined as follows:

$$\begin{aligned} \neg \top &= \perp, \\ \rho(tt, \bar{m}) &= \top, \\ \rho(GAP, \bar{m}) &= f(m_1, m_2 \dots m_K) - c, \\ \rho(\mathcal{Y}_1 \wedge \mathcal{Y}_2, \bar{m}) &= \min(\rho(\mathcal{Y}_1, \bar{m}), \rho(\mathcal{Y}_2, \bar{m})), \\ \rho(\neg \mathcal{Y}, \bar{m}) &= -\rho(\mathcal{Y}, \bar{m}), \\ \rho(\mathcal{Y}_1 \mathcal{U}^I \mathcal{Y}_2, \bar{m}) &= \sup_{t' \in I} \min\left(\rho(\mathcal{Y}_2, \bar{m}(t')), \inf_{t'' \in [0, t']} \rho(\mathcal{Y}_1, \bar{m}(t''))\right), \end{aligned}$$

where $\bar{m} = \bar{m}(0)$ at time $t = 0$, and $\bar{m}(t')$ is a solution of the ODEs (1) at time $t = t'$ with \bar{m} as the initial condition.

Time and space-time robustness of satisfaction for a quantitative semantics is discussed in Ref. [15], where $\rho(\mathcal{Y}, \bar{m})$ is called a *robustness estimate*. The robustness estimate is found using an inductive procedure for model-checking MFL formulas. Efficient algorithms to find robust estimates are described in Ref. [9], and the Breach toolbox [19] can be directly used for that purpose. Given algorithms to find the robust satisfaction of an MFL formula, the satisfaction set of such formula, $\text{Sat}(\mathcal{Y})$, can be calculated by partitioning the state-space S^O of the mean-field model. The latter can be done using refining partition, as proposed in Refs. [8, 15].

The robustness analysis can also be performed with tools like S-TaLiRo [20] and BIOCHAM [21]. Note also that here no formal guarantees on the correctness of results can be provided. The advantage of the robustness-based method lies in the fact that the procedure is ununiform for any MFL formula, unlike reachability-based methods. Moreover, there are tools available for robustness analysis of the

time series, numerical solutions of the ODEs, or even measured data in the context of satisfaction set development. Finally, more advanced numerical methods may be applied to partition the state space.

6 Relation Between the Two Logics

As discussed in the previous sections, there are two ways to describe properties of the overall mean-field model. One way is reasoning about the fraction of objects satisfying a given local property by checking whether this number meets a given threshold using the logic MF-CSL. Another way is to describe the properties of the whole system, including timed properties, which can be done with the logic MFL. In Sect. 6.1, we discuss the difference between these two logics and argue that both possess value. The possibility of combining both logics is discussed in Sect. 6.2.

6.1 Comparison of MFL and MF-CSL

Table 1 depicts the main differences between the MFL and MF-CSL logics. As previously discussed, both logics are used in order to describe (and check) properties of mean-field models. Moreover, the time validity set can be defined for MF-CSL, while model-checking MFL properties require the computation of TVSSs.

All properties in MF-CSL are based on the structure and labelling of the local model, and *expectation operators* lift these properties to the global level. MFL expresses properties of the global mean-field model independently from the labelling and structure of the local model. A *GAP* can be defined both on the local model structure and labelling, as well as via labelling-independent functions of the occupancy vector \bar{m} . For example, properties such as “there are infected computers in all three groups” can easily be described by MFL, while MF-CSL needs “workarounds” by

Table 1 The MF-CSL logic versus the MFL logic

Property	MF-CSL	MFL
Applicable for mean-field models	+	+
Operates on both local and global levels	+	–
Timed property on the local level	+	–
Timed property on the global level	–	+
Depends on the local labelling	+	–
Uses expectation relations	+	–
Has notion of TVS	+	+
Satisfaction set can be obtained	–/+	+

either introducing different labelling on the local level, or expressing the *infected* properties for each group separately on the local level, then on the global level, and finally combining these properties by concatenation and/or negation.

The largest difference lies in the application of timed properties. Both logics may use the until operator. However, in MFL the until operator is used on the global level, while in MF-CSL only the evolution of an individual random object can be described with the until operator. Approximating the full satisfaction set is possible for MFL properties, as defined in Sect. 5. The calculations on the local level of MF-CSL are, however, quite demanding and the partitioning of the state-space using MF-CSL property as an indicator function is impractical.

For some models, such as models of chemical reactions [41], the behaviour of a random individual (one molecule) is not of interest. Therefore, MF-CSL may not be of interest and only the logic MFL would be applicable. Despite this, there are many systems that can be modelled using the mean-field method, where the behaviour of a random object would still be important, for example in the virus spread models, as discussed in this chapter. Clearly, both logics can be of interest, albeit for different users and different systems. Some properties can be expressed in both logics. However, the majority of properties can only be described using one of the two logics, which explains the necessity of introducing both these logics separately.

6.2 Combination of the Two Logics

We now discuss the combination of the two logics to achieve the greatest expressivity. As described in Sect. 5, a *GAP* can be defined by any Boolean function which, when applied to the model trajectory, produces as output a robust cadlag function. As MF-CSL properties can be interpreted as a Boolean function $S^o \rightarrow \{0, 1\}$, combined properties can be expressed and analysed.

The above can be generalize as follows: in order to describe the combined property of the global mean-field model, a linear combination of the MF-CSL properties is used as a global atomic property:

$$\sum_j a_j \cdot \mathbb{1}_{\psi_j} \bowtie p, \tag{5}$$

where a_j is a real number and $\mathbb{1}_{\psi_j}$ is the indicator function of the MF-CSL property ψ_j being satisfied.

Due to the cadlag restriction on the *GAP* function, the set of MF-CSL formulas that can be used as *GAP* is restricted in order to guarantee decidability, e.g. properties which are only valid at one time point are not allowed for the combination. Apart from this, any MF-CSL formula whose TVS consists of a finite number of sets can be used as *GAP* in MFL. As an example of such a property, we consider the following formula:

$$\mathcal{Y} = (\mathbb{E}_{\leq 0.1} active_Y) \mathcal{U}^{[0,5]} (\mathbb{E}\mathbb{P}_{\leq 0.1} tt U^{[0,3]} infected_Y)$$

This property is useful for a system administrator, who wants to be sure that with the current activity of the anti-malware software not more than 10 % of the computers in group Y are infected and active until within 5 time units a system state is reached, where the probability that a random computer will become infected within 3 time units is less or equal than 0.1.

The example below provides a more detailed explanation on how to check such properties. Note that, the calculation of the satisfaction set of such combined properties may not be practically feasible, due to the high computational costs on the local level of MF-CSL sub-formula.

Example 7 The virus spread model is used in the following example with the parameters as given in Example 4. We will construct a combined property using both logics. Then we find the time validity set for the property obtained given a predefined time interval $\theta = [0, 20]$, and initial distribution

$$\bar{m}(0) = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0, 0.1\}, \{0.4, 0.55, 0.05\}).$$

Finally, we check the combined property against the initial occupancy vector $\bar{m}(0)$. For simplicity, we use the MF-CSL property, described in Example 4 as one of global atomic properties in the combined property:

$$\mathcal{Y}_1 = \mathbb{E}\mathbb{P}_{< 0.3}(uninfected_Y U^{[0,3]} infected_Y).$$

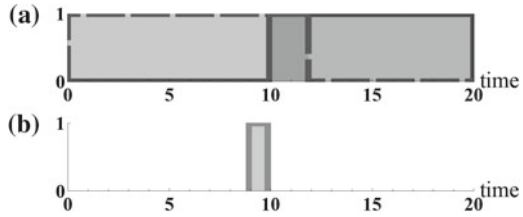
We combine \mathcal{Y}_1 with $\mathcal{Y}_2 = active \leq 0.1$ using the until operator and obtain the following combined formula:

$$\mathcal{Y} = \mathbb{E}\mathbb{P}_{< 0.3}(uninfected_Y U^{[0,3]} infected_Y) \mathcal{U}^{[1,2]} (active \leq 0.1).$$

This property describes a system where the expected probability that a random computer is from group Y and becomes infected within 3 time units is less than 0.3, at all times until within time interval $[1, 2]$ the number of active infected computers in all three groups is less or equal than 0.1.

To check such a combined property we must first find the TVSs of all sub-formulas, including the time validity set of the MF-CSL formula \mathcal{Y}_1 , which equals $TVS(\mathcal{Y}_1, \bar{m}, \theta) = [0, 11.86]$ (see Example 4). The time validity set of the MFL sub-formula $TVS(\mathcal{Y}_2, \bar{m}, \theta)$ is found by solving $m_{X,3} + m_{Y,3} + m_{Z,3} = 0.1$; and equals $TVS(\mathcal{Y}_2, \bar{m}, \theta) = [9.925, 20]$. Figure 6a displays the time validity sets of sub-formulas \mathcal{Y}_1 and \mathcal{Y}_2 . The time validity set of the combined formula is computed as described in Sect. 5.2. First, the intersection of $TVS(\mathcal{Y}_1, \bar{m}, \theta)$ and $TVS(\mathcal{Y}_2, \bar{m}, \theta)$ is found: $TVS(\mathcal{Y}_1 \cap \mathcal{Y}_2, \bar{m}, \theta) = [9.925, 11.86]$. Then, $TVS(\mathcal{Y}_1 \cap \mathcal{Y}_2, \bar{m}, \theta)$ is shifted backwards: $TVS(\mathcal{Y}, \bar{m}, \theta) = [9.925 - 2, 11.86 - 1] = [7.925, 10.86]$ (see Fig. 6b). As one can see, the occupancy vector \bar{m} does not satisfy the combined property \mathcal{Y} , since $0 \notin [7.925, 10.86]$.

Fig. 6 **a** TVSs of Υ_1 (dashed line) and Υ_2 (solid line). **b** TVS of Υ



7 Conclusions

Over the last decade, many systems that consist of a large number of interacting objects have been analysed using the mean-field method. This method avoids the well-known state-space explosion problem, which is encountered in many classical Markovian analysis techniques. However, the mean-field method has primarily been used for classical performance evaluation purposes. In this chapter, we have discussed model-checking algorithms for mean-field models, in order to be able to address non-standard system properties.

In this chapter, we define and motivate two logics, called *Mean-Field Continuous Stochastic Logic* (MF-CSL) and *Mean-Field Logic* (MFL), to describe properties of systems composed of many identical interacting objects. The logic MF-CSL [6] uses local CSL properties as a basis for the global expectation operator. Therefore, it is fully dependent on the structure of the local model. The logic MFL, on the other hand, does not take into account properties of the individual objects and only reasons on the global level. Therefore, time-dependent properties of the global model can be described using MFL, while MF-CSL allows only time-dependent properties on the local level.

The algorithms to check MFL properties against a given occupancy vector and to find the so-called time validity set are based on the methods of monitoring temporal properties as in Refs. [7, 39]. We adapt these methods to check global properties of the mean-field model, and illustrate these algorithms in examples. All computations were done in Wolfram Mathematica and compared against results produced by the Breach Toolbox, which were consistent with one another.

Furthermore, three possible ways to calculate the satisfaction set of an MFL formula were discussed. One of these methods relies on the Boolean semantics of MFL presented and a discretization of the continuous state space. The second method makes use of the existing technique to find the parameters of the model, which satisfy a given *reachability* problem [8] using *sensitivity analysis*. The third method adapts an existing notion of *robustness* [15] of a temporal logic and *sensitivity analysis*. This technique is based on defining a *quantitative semantics* of MFL. The resulting robust estimate is then used as an indicator in order to guide the partitioning algorithm.

The expressivity and applicability of the two logics were also compared in this chapter. Despite the fact that both logics are applicable to mean-field models, both are clearly of interest and can not be fully replaced by the other. Another interesting

insight related to the use of the logics presented is that they can be combined if the global atomic property of the mean-field model is represented by one of the expectation operators. This allows the combination of MF-CSL and MFL properties on both levels, including timed properties. Such properties can be easily checked for a given occupancy vector. However, the satisfaction set development may be even more challenging.

Acknowledgments The work in this chapter has been performed when Anna Kolesnichenko was still at the University of Twente. She has been supported through NWO grant 612.063.918, MAT-MAN (Mean-Field Approximation Techniques for Markov Models), as well as the FP7 Sensation project (see below). Anne Remke has been supported through an NWO VENI grant on Dependability Analysis of Fluid Critical Infrastructures using Stochastic Hybrid Models. Boudewijn Haverkort and Pieter-Tjerk de Boer have been supported through FP7 STREP 318490, Sensation (Self Energy-Supporting Autonomous Computation).

References

1. Kurtz TG (1970) Solutions of ordinary differential equations as limits of pure jump Markov processes. *J Appl Probab* 7(1):49–58
2. Bortolussi L, Hillston J, Latella D, Massink M (2013) Continuous approximation of collective systems behaviour: a tutorial. *Perform Eval* 70(5):317–349
3. Bakhshi R, Cloth L, Fokkink W, Haverkort BR (2009) Mean-field analysis for the evaluation of Gossip protocols. In: *QEST*, IEEE CS Press, pp 247–256
4. Bakhshi R, Endrullis J, Endrullis S, Fokkink W, Haverkort BR (2010) Automating the mean-field method for large dynamic gossip networks. In: *QEST*, IEEE CS Press, pp 241–250
5. Kolesnichenko A, Remke A, de Boer PT, Haverkort BR (2011) Comparison of the mean-field approach and simulation in a peer-to-peer botnet case study. In: *EPEW*, vol 6977. LNCS, Springer, pp 133–147
6. Kolesnichenko A, Remke A, de Boer PT, Haverkort BR (2013) A logic for model-checking mean-field models. In: *DSN/PDF*, IEEE CS Press, pp 1–12
7. Maler O, Nickovic D (2004) Monitoring temporal properties of continuous signals. In: *FOR-MATS*, vol 3253. LNCS, Springer, pp 152–166
8. Donzé A, Clermont G, Legay A, Langmead CJ (2010) Parameter synthesis in nonlinear dynamical systems: application to systems biology. *J Comput Biol* 17(3):325–336
9. Donzé A, Ferrère T, Maler O (2013) Efficient robust monitoring for STL. In: *CAV*, vol 8044. LNCS, Springer, pp 264–279
10. Hillston J (2014) The benefits of sometimes not being discrete. In: *CONCUR*, vol 8704. LNCS, Springer, pp 7–22
11. Bortolussi L, Hillston J (2012) Fluid model checking. In: *CONCUR*, vol 7454. LNCS, Springer, pp 333–347
12. Bortolussi L, Hillston J (2013) Checking individual agent behaviours in markov population models by fluid approximation. In: *SFM*, vol 7938. LNCS, Springer, pp 113–149
13. Bortolussi L, Lanciani R (2013) Model checking Markov population models by central limit approximation. In: *QEST*, vol 8054. LNCS, Springer, pp 123–138
14. Latella D, Loreti M, Massink M (2014) On-the-fly fast mean-field model-checking. In: *TGC*, LNCS, Springer, pp 297–314
15. Donzé A, Maler O (2010) Robust satisfaction of temporal logic over real-valued signals. In: *FORMATS*, vol 6246. LNCS, Springer, pp 92–106
16. Fainekos GE, Pappas GJ (2009) Robustness of temporal logic specifications for continuous-time signals. *Theor Comput Sci* 410(42):4262–4291

17. Rizk A, Batt G, Fages F, Soliman S (2008) On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In: CMSB, vol 5307. LNCS, Springer, pp 251–268
18. Bartocci E, Bortolussi L, Nenzi L, Sanguinetti G (2013) On the robustness of temporal properties for stochastic models. In: HSB, vol 125. EPTCS, Open Publishing Association, pp 3–19
19. Donzé A, Breach A (2010) Toolbox for verification and parameter synthesis of hybrid systems. In: CAV, vol 6174. LNCS, Springer, pp 167–170
20. Annpureddy Y, Liu C, Fainekos G, Sankaranarayanan S (2011) S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: TACAS, vol 6605. LNCS, Springer, pp 254–257
21. Calzone L, Fages F, Soliman S (2006) Biocham: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics* 22(14):1805–1807
22. Chaintreau A, Le Boudec JY, Ristanovic N (2009) The age of gossip: spatial mean field regime. In: ACM SIGMETRICS/Performance, pp 109–120
23. Bobbio A, Griboaldo M, Telek M (2008) Analysis of large scale interacting systems by mean field method. In: QEST, IEEE Computer Society, pp 215–224
24. Darling RWR, Fluid limits of pure jump Markov processes: a practical guide, ArXiv mathematics e-prints [arXiv:arXiv:math/0210109](https://arxiv.org/abs/math/0210109)
25. Darling RWR, Norris JR (2008) Differential equation approximations for Markov chains. *Probab Surv* 5:37–79
26. van Ruitenbeek E, Sanders WH (2008) Modeling peer-to-peer botnets. In: QEST, IEEE CS Press, pp 307–316
27. Kurtz TG (1970) Solutions of ordinary differential equations as limits of pure jump Markov processes. *J Appl Probab* 7(1):49–58
28. Billingsley P (1995) Probability and measure. 3rd edn. Wiley-Interscience
29. Wolfram Research Inc (2010) Mathematica tutorial. <http://www.wolfram.com/mathematica/>
30. Gast N, Gaujal B (2010) A mean field model of work stealing in large-scale systems. In: ACM SIGMETRICS, ACM, pp 13–24
31. Bortolussi L (2011) Hybrid limits of continuous time Markov chains. In: QEST, IEEE Computer Society, pp 3–12
32. Hayden RA (2012) Mean field for performance models with deterministically-timed transitions. In: QEST, IEEE Computer Society, pp 63–73
33. Hayden RA, Horváth I, Telek M (2014) Mean field for performance models with generally-distributed timed transitions. In: QEST, vol 8657. LNCS, Springer, pp 90–105
34. Stefanek A, Hayden RA, Gonagle MM, Bradley JT (2012) Mean-field analysis of Markov models with reward feedback. In: ASMTA, vol 7314. LNCS, pp 193–211
35. Stefanek A, Hayden RA, Bradley JT (2014) Mean-field analysis of hybrid Markov population models with time-inhomogeneous rates. *Ann Oper Res* 1–27
36. Le Boudec JY (2010) The stationary behaviour of fluid limits of reversible processes is concentrated on stationary points. Technical report
37. Benaïm M, Le Boudec JY (2008) A class of mean field interaction models for computer and communication systems. *Perform Eval* 65(11–12):823–838
38. Baier C, Haverkort BR, Hermanns H, Katoen JP (2003) Model-checking algorithms for continuous-time Markov chains. *IEEE Trans Softw Eng* 29(7):524–541
39. Nickovic D, Maler O (2007) AMT: a property-based monitoring tool for analog systems. In: FORMATS, vol 4763. LNCS, Springer, pp 304–319
40. Pnueli A (1977) The temporal logic of programs. In: SFCS, IEEE computer society, pp 46–57
41. Gómez-Marn AM, Hernández-Ortiz JP (2013) Mean field approximation of Langmuir-Hinshelwood CO-surface reactions considering lateral interactions. *J Phys Chem* 117(30):15716–15727

Part III
Checkpointing and Queueing

Standby Systems with Backups

Gregory Levitin and Liudong Xing

Abstract This chapter presents a numerical methodology to model and evaluate reliability, expected mission completion time, and expected total mission cost of 1-out-of- N : G standby sparing systems subject to periodic or non-periodic backup actions. The backups are performed to facilitate effective system recovery in the case of the occurrence of an online operating element failure. The methodology is applicable to dynamic data backup and retrieval times as well as nonidentical system elements with different time-to-failure distributions, different performance, and different standby modes. This chapter also presents applications of the methodology to a set of optimization problems that find the optimal backup distribution and/or element activation sequence, maximizing mission reliability or minimizing expected mission completion time or minimizing total mission cost. Examples are provided to illustrate the presented methodology as well as optimized solutions.

Keywords Even or periodic backup · Uneven or non-periodic backup · Cold standby · Hot standby · Warm standby · Backup distribution · Mission cost · Mission time · Optimization · Element sequencing

Nomenclature

- R Mission reliability or success probability
- C Expected total mission cost
- D Expected mission completion time
- N Number of system elements
- H Number of backups throughout the mission
- M Total number of operations (excluding backups) to be performed during the mission

G. Levitin (✉)
The Israel Electric Corporation, P.O. Box 10, 31000 Haifa, Israel
e-mail: levitin@iec.co.il

L. Xing
University of Massachusetts Dartmouth, 285 Old Westport Road,
Dartmouth, MA 02747, USA
e-mail: lxing@umassd.edu

- T_{\max} Maximum allowed mission time
 Y_{\max} Maximum allowed number of time intervals in the mission
 Y_{\min} Minimum possible number of time intervals in the mission
 w_j, z_j Per time unit operation, standby cost of element j
 v_j, λ_j Replacement cost, time of warm standby element j
 $F_j(t)$ Time-to-failure *cdf* for element j
 G_j Performance (number of operations per unit time) of element j
 d_j Deceleration factor for element j during the standby mode
 Δ Number of operations in each discrete portion of work
 g_j Number of work portions performed by element j in a time unit: $g_j = G_j/\Delta$
 π_h Fraction of the entire mission task that should be performed between the $(h - 1)$ -th and h -th backups
 $\boldsymbol{\pi}$ Backup distribution vector: $\boldsymbol{\pi} = (\pi_1, \dots, \pi_H)$
 B_h Number of operations needed for the h -th backup
 U_h Number of operations needed to retrieve the data stored in the h -th backup procedure
 δ_h Number of work portions needed for the h -th backup: $\delta_h = B_h/\Delta$
 γ_h Number of work portions that should be completed between the $(h - 1)$ -th and h -th backups: $\gamma_h = \pi_h M/\Delta$
 μ_h Number of work portions needed to retrieve data stored by the h -th backup: $\mu_h = U_h/\Delta$
 α_h Number of work portions that should be performed between the beginning of the mission and the end of the h -th backup
 m Total number of work portions needed to accomplish the mission when no failures occur
 $s(k)$ Index of the element that should be initiated, given it is still working, after elements with indices $s(1), \dots, s(k - 1)$ have failed
 h_k Index of the last backup procedure completed by the element sequence $s(1), \dots, s(k - 1)$
 Y_k Index of the time interval when the last element from the sequence $s(1), \dots, s(k - 1)$ fails
 $Q_k(a, b)$ $\Pr\{h_k = a, Y_k = b\}$
 $b(x)$ Number of operations needed to save data generated after performing fraction x of the entire mission task
 $u(x)$ Number of operations needed to retrieve data saved after performing fraction x of the entire mission task
 $[x]$ Integer closest to x
 τ Minimal recognized time interval
 $\psi(t)$ Discretization function: $\psi(t) = [t/\tau]$

1 Introduction

Application of standby sparing techniques abounds in diverse systems, where one or multiple elements are operating and online with additional elements serving as standby spares. When an on-line element fails, it is switched out of operation. The system operation resumes when an available standby element is activated to take over the mission task from the failed element [3, 31, 36]. Examples of standby sparing systems include power [41], storage [11], flight control [14], high performance computing [13], and space mission [35] systems.

Standby sparing effectively enhances a system's reliability and availability. However, this benefit comes with overhead [14, 40]. Particularly in standby systems performing computing related tasks, to enable effective system reconfiguration each online operating element typically performs periodic or non-periodic data backups during its lifetime based on a pre-specified backup policy. These backup operations incur additional mission time as well as additional capital cost for storage. The data backup time is dynamic, which is dependent on the amount of work accomplished since the last backup point or since the beginning of the mission. The overhead also includes the time and cost required for performing replacement procedures. Specifically, in case of an on-line element experiencing a failure, a replacement procedure is initiated to activate an available standby element and to retrieve the previously saved data from the backup storage and then transfer to the newly activated standby element. In addition, before performing the remaining uncompleted task the newly activated standby element needs to re-execute all work portions that have been completed by the failed element since the last successful backup, incurring additional time and cost.

To achieve reliable and cost-effective system operation under limited time and budget constraints a trade-off study among mission reliability, time, and cost is essential. This chapter presents a numerical methodology to facilitate such a trade-off analysis for 1-out-of- N : G standby systems subject to backups. The methodology simultaneously evaluates three mission performance indices (reliability, expected mission time, and expected total mission cost). The method is applicable to dynamic data backup and retrieval times. System elements are not necessarily identical; they can have different time-to-failure distributions, different performance, and different standby modes (cold, hot, and warm). An element in a hot standby mode operates concurrently with the online active element, and is ready to take over the task at any time [14]. Hence, a hot standby element can provide fast system recovery but at the cost of high maintenance or operation overhead because it consumes energy and materials as much as the online element does. An element in a cold standby mode is unpowered and fully isolated from the working stresses. It does not consume any energy or materials until being activated to replace a failed on-line element. Thus, it has the minimal maintenance cost and its failure rate can be assumed to be zero before being activated. However, a cold standby element incurs high startup costs or significant restoration delays [38, 39]. As a more general case, an element in a warm standby mode is partially exposed to working stresses and partially ready to

take over the mission task from the failed element [17, 33]. Both standby cost and restoration delay for a warm standby element are in-between those for a hot standby element and a cold standby element.

This chapter also presents applications of the suggested numerical evaluation method to solving a set of unconstrained and constrained optimization problems including the optimal backup distribution problem, the optimal standby element sequencing problem, and the combined backup distribution and element sequencing optimization problem. The objective of these optimization problems is to maximize mission reliability or to minimize expected mission completion time or to minimize total mission cost of the standby sparing system considered in this work.

2 Relevant Work

Numerous efforts have been expended on formulating and solving optimization problems for different types of standby systems [15]. For example, exact methods of dynamic programming, integer programming, and Lagrange multipliers have been applied to solve the redundancy allocation problem (RAP) for 1-out-of- N : G series-parallel hot standby systems adopting a homogeneous backup strategy [12, 27, 28]. Meta-heuristic methods such as the genetic algorithm (GA), ant colony optimization algorithm, and Tabu search have been applied to solve RAP for K -out-of- N : G series-parallel hot standby systems adopting a heterogeneous backup strategy [6, 7, 10, 30]. As compared to the homogeneous backup where one type of element can be substituted only with the same type of elements, in a heterogeneous backup scheme, one type of element can be substituted by a different type of element with equivalent functionality.

Example works on solving RAP for cold standby systems include an integer programming method [8], and a hybrid algorithm using a GA and fuzzy theory [43]. In [9], the integer programming approach was extended to solve RAP for heterogeneous series-parallel systems with either hot or cold standby elements configured in different parallel subsystems. In [4], the GA was adapted to solve RAP for heterogeneous series-parallel systems with cold and hot standby elements co-existing within one parallel subsystem. In addition, a multi-objective version of the GA [5], and a hybrid intelligent algorithm using the GA, neural networks, and fuzzy theory [42] have been developed to solve RAP for systems with a mix of cold standby and active redundancy technique. Recently, the GA was also applied to solve the optimal element sequencing problem for heterogeneous cold standby systems performing single-phased [18, 19], or multi-phased [20] mission tasks.

Example optimization works for general warm-standby systems include integer programming and GA-based methods for solving RAP of series-parallel warm standby systems in [1, 37]. The GA-based method was also implemented to solve the optimal element sequencing problem for 1-out-of- N : G heterogeneous warm standby systems with perfect [17], or imperfect [21] switching mechanisms.

Despite the extensive research efforts on standby system modeling and optimization, effects of backups were not considered until the recent work [23], where a restricted backup model with even backups, fixed data backup time, and negligible data retrieval time was assumed. In [24], the methodology of [23] was further extended to consider uneven backups as well as dynamic backup and retrieval times [24]. However, the methodologies suggested in [23] and [24] are only applicable to the special class of 1-out-of- N : G cold-standby systems; they cannot be applied to the general, more complex warm-standby systems. There are also studies on optimal backup policies considering different backup techniques for database systems [29, 32] and computer disks [34]; these works do not consider effects of standby sparing.

This chapter presents a numerical methodology that considers effects of uneven backups with dynamic data backup and retrieval times for modeling and analyzing 1-out-of- N : G warm standby systems. Both hot and cold standby systems with even or uneven backups appear as special cases of the proposed model. This chapter also presents formulation and solution for a set of optimization problems relevant to these systems considering mission reliability, expected mission completion time, and expected total mission cost.

3 Standby System Model and Assumptions

There are N nonidentical elements in the considered 1-out-of- N : G standby system, which performs a mission consisting of M operations (i.e., the computational complexity of the mission is M). Each element j is characterized by a specific time-to-failure distribution with cumulative distribution function (cdf) $F_j(t)$ and its performance in number of operations completed per unit time G_j . In case of no failures or backups happening, element j needs time M/G_j to accomplish the entire mission task. At any time, only one system element is online and operating with remaining unfailed elements waiting in different standby modes. Particularly, at the beginning of the mission, according to a prespecified element activation sequence $s(1), \dots, s(N)$, the first element $s(1)$ is activated and online while elements $s(2), \dots, s(N)$ wait in the standby mode. When the online element fails, an available standby element at the beginning of the sequence is activated to replace the failed element and take over the mission task.

During the mission, H data backup actions are performed. The h -th backup is performed when a fraction π_h ($h = 1, \dots, H$) of the entire mission task is completed since the previous backup and it requires B_h operations. The value of B_h is dynamic, depending on the work x completed since the last backup, or since the mission beginning defined by function $B_h = b(x)$. Specifically, if an *incremental* backup technique is used, the amount of data saved during a backup action, and thus the number of operations needed for the backup (i.e., B_h) depends on the amount of work completed since the last successful backup. If a *total* backup technique is used, B_h depends on the amount of total work completed since the beginning of the mission. Thus,

$$B_h = \begin{cases} b(\pi_h), & \text{incremental backup} \\ b\left(\sum_{j=1}^h \pi_j\right), & \text{total backup} \end{cases} \quad (1)$$

Refer to [26] for a study of repairable systems subject to combined total and incremental backups during the mission.

In case of failure of the online operating element, the status of the first remaining standby element $s(j)$ in the sequence is checked. If it fails before it should be activated, then the next element $s(j + 1)$ is checked, and so on until an available standby element is found. Then a replacement or activation process starts immediately, which includes element startup (installing, powering up, and connecting the element to the system) and warming up (preparing the powered element for a typical function, e.g., a hard disk drive should reach its nominal speed before being used). The replacement process can be performed automatically or by technical personnel. During the replacement process, the activated element is exposed to operational stresses. In this work, replacement cost v_j and time λ_j are assumed to be fixed for each element j , which are logically independent from status (success or failure) of the replacement procedure. This assumption is made because the replacement procedure may not be stopped immediately after its failure, and the failed element should be switched off or removed, which takes time and effort. Upon finishing a successful replacement or activation procedure, the newly activated element first retrieves the previously saved data from that backup storage. Equation (2) gives the number of operations U_h required for a retrieval process performed after the h -th backup procedure, which depends on the amount of total work completed before the last successful backup procedure (i.e., the h -th backup).

$$U_h = \begin{cases} 0, & h = 0 \\ u\left(\sum_{j=1}^h \pi_j\right), & h > 0 \end{cases} \quad (2)$$

As illustrated in Fig. 1, each activated standby element, after retrieving the data, starts performing the mission task from the operation immediately following the last successful backup action. A standby element j activated after the h -th backup can successfully perform the next k backups if it does not fail during the time

$$\left(U_h + \sum_{i=1}^k (M\pi_{h+i} + B_{h+i}) \right) / G_j. \quad (3)$$

In the case of no failures occurring at all during the mission, the mission time is

$$\left(M + \sum_{h=1}^H B_h \right) / G_{s(1)}. \quad (4)$$

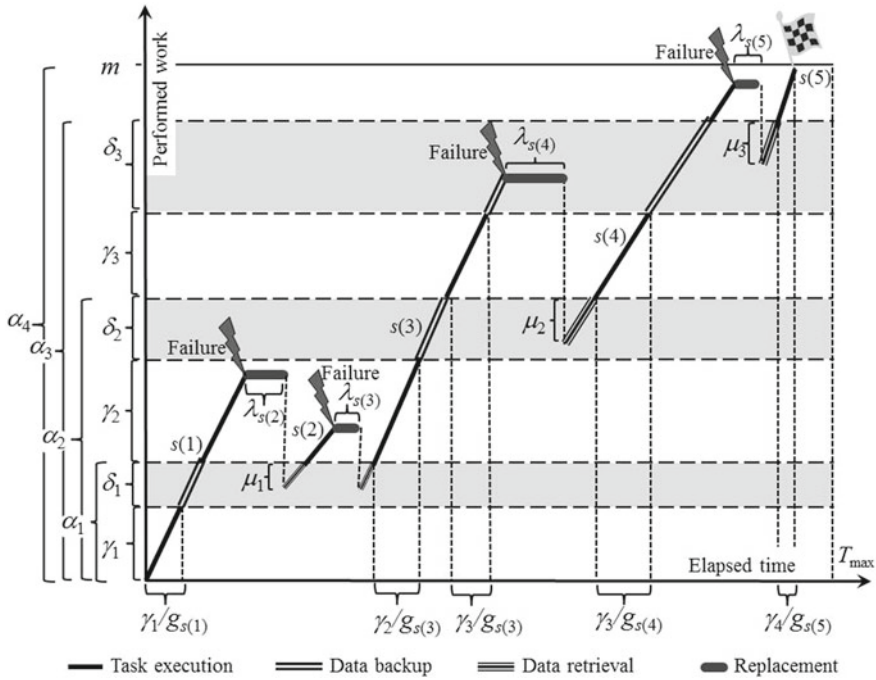


Fig. 1 Example of a successful mission with 3 backups, 4 replacements

Note that this time may not be the minimal possible mission time if the performance of element $s(1)$ is not the highest. For example, consider a case where element $s(1)$ fails immediately after performing the first backup, and element $s(2)$ is activated and completes the remaining part of the mission successfully. In this case, the mission time is

$$(\pi_1 M + B_1) / G_{s(1)} + \lambda_{s(2)} + \left(M(1 - \pi_1) + U_1 + \sum_{h=2}^H B_h \right) / G_{s(2)}. \quad (5)$$

If the inequality in (6) is true, then the mission time of (5) is less than the mission time of (4).

$$G_{s(1)} < G_{s(2)} \left(M(1 - \pi_1) + \sum_{h=2}^H B_h \right) \left(\lambda_{s(2)} G_{s(2)} + M(1 - \pi_1) + U_1 + \sum_{h=2}^H B_h \right)^{-1}. \quad (6)$$

The standby system considered in this work is a real-time system that must complete the entire mission task within a maximum allowed value T_{max} . Otherwise, the mission fails.

The following assumptions are also made in the model.

- (1) The system possesses a perfect fault detection and switching mechanism, and a perfect backup mechanism.
- (2) A replacement procedure can be activated as long as the allowed mission time T_{\max} has not elapsed, even if the remaining time does not allow completing the mission.
- (3) The mission task is executed evenly in time by any operating element.
- (4) The system elements are not repairable during the mission.

4 Evaluation Algorithm of Mission Performance Indices

This section presents detailed derivation of numerical algorithms for evaluating mission reliability, expected mission completion time and expected total mission cost of 1-out-of- N : G warm standby systems subject to uneven backups and dynamic data backup and retrieval times.

4.1 Mission Performance and Time Discretization

The entire mission task $M + \sum_{h=1}^H B_h$ is divided into equal work portions, each containing Δ operations. Equation (7) gives the total number of work portions performed in the entire mission.

$$m = \frac{1}{\Delta} \left(M + \sum_{h=1}^H B_h \right). \quad (7)$$

No element should perform greater than m work portions. In other words, once m work portions are completed, the entire mission is accomplished successfully and the operating element is switched off.

$g_j = G_j/\Delta$ gives the processing speed of element j in number of work portions performed in a time unit. Equations (8) and (9) give the number of work portions performed in the h -th data backup and retrieval procedures, respectively.

$$\delta_h = B_h/\Delta \quad (8)$$

$$\mu_h = U_h/\Delta \quad (9)$$

The number of work portions that should be performed between the $(h - 1)$ -th and h -th backup procedures is

$$\gamma_h = \pi_h M/\Delta. \quad (10)$$

We define $\delta_0 = \mu_0 = \gamma_0 = 0$ since $h = 0$ corresponds to the beginning of the mission.

Equation (11) gives the number of work portions that should be completed when the k -th backup procedure ($1 \leq k \leq H$) is completed (see Fig. 1 for illustration).

$$\alpha_k = \sum_{i=0}^k (\gamma_i + \delta_i) \quad (11)$$

By definition, $\alpha_{H+1} = m$, corresponding to the completion of the entire mission.

For any number of completed work portions i , the number or index of the last successfully completed backup procedure can be obtained as $\varphi(i)$ such that $\alpha_{\varphi(i)} \leq i < \alpha_{\varphi(i)+1}$.

If an element, which is activated after h backups, successfully retrieves backup data (involving μ_h work portions), and then keeps functioning during performing $\sum_{i=1}^k (\gamma_{h+i} + \delta_{h+i}) = \alpha_{h+k} - \alpha_h$ work portions, it completes the k next backup procedures successfully. Thus, the accomplished $\alpha_{h+k} - \alpha_h$ work portions should not be re-performed again; only work portions that follow the last successful backup (or from the beginning of the mission in the case where no backups have been successfully performed yet) should be re-performed.

Consider, for example in Fig. 1 element $s(2)$ that fails after completion of the first backup but before completion of the second backup. Element $s(3)$ is activated after the failure of $s(2)$. First, element $s(3)$ retrieves data saved by the first backup, which requires performing μ_1 work portions and takes $\mu_1/g_{s(3)}$ time intervals. If element $s(3)$ fails after performing more than $\gamma_2 + \delta_2 = \alpha_2 - \alpha_1$ but less than $\gamma_2 + \delta_2 + \gamma_3 + \delta_3 = \alpha_3 - \alpha_1$ work portions, the index of the last backup performed by $s(3)$ is $h = 2$. Thus, the next activated element $s(4)$ should retrieve data saved after the second backup, which requires performing μ_2 work portions and takes $\mu_2/g_{s(4)}$ time intervals. Upon finishing the data retrieval, element $s(4)$ should continue the mission task from a work portion immediately following the second backup. Note that $\mu_h < \gamma_h$, otherwise the data backup is not beneficial.

To discretize time, a time interval τ is introduced and a time period t is measured using an integral number of time intervals through function $\psi(t) = [t/\tau]$, where $[t/\tau]$ represents the integer closest to t/τ . Thus, given the maximal allowed mission time T_{\max} , the maximal number of time intervals in the entire mission is $Y_{\max} = \psi(T_{\max})$.

4.2 Single Element Failure Probability

The cumulative exposure model (CEM) that uses the equivalent age concept [2] is applied to account for effects of different modes (standby, replacement, and operation) during the lifetime of an element. Consider element j that stays in the standby mode for a time period of t_{WS} , then undergoes replacement taking time λ_j and then operates for a duration of t_{OM} . According to CEM, a stress-dependent cumulative exposure time t^* is calculated, where the time element j spends in the standby mode is multiplied by a deceleration factor $d_j < 1$ reflecting lower stresses, that is, $t^* = d_j t_{\text{WS}} + \lambda_j + t_{\text{OM}}$ [2]. Given the *cdf* $F_j(t)$ of element j in the operation mode,

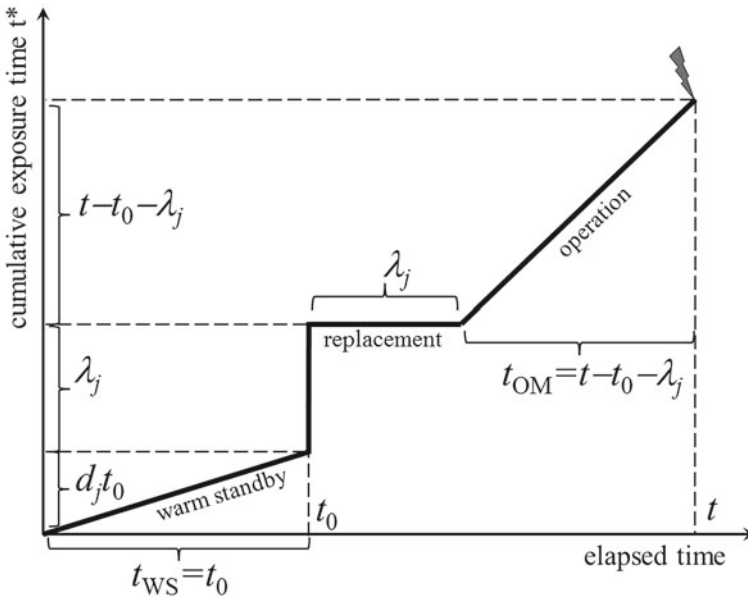


Fig. 2 Determination of the cumulative exposure time t^*

the cumulative time-to-failure distribution function of element j that experiences all three modes is thus $F_j(t^*) = F_j(d_j t_{WS} + \lambda_j + t_{OM})$. If element j should be activated at time t_0 , then its *cdf* at time t is

$$F_j(t^*) = F_j(d_j \min(t, t_0) + \lambda_j \cdot 1(t > t_0) + \max(0, t - t_0 - \lambda_j)). \tag{12}$$

In (12), $t_{WS} = \min(t, t_0)$ is the time spent in the standby mode, $\lambda_j \cdot 1(t > t_0)$ the time in the replacement mode, and $t_{OM} = \max(0, t - t_0 - \lambda_j)$ the time exposed to the operational stresses (see Fig. 2 for an illustration). If element j fails before its activation, then $F_j(t^*) = F_j(d_j t)$.

The probability that element j , which should be activated at time t_0 , fails in an interval between t_1 and $t_2 \geq t_1$ is thus $F_j(t_2^*) - F_j(t_1^*)$, where $t_k^* = d_j \min(t_k, t_0) + \lambda_j \cdot 1(t_k > t_0) + \max(0, t_k - t_0 - \lambda_j)$ for $k = 1, 2$. Notice that the probability that an element fails during a certain time interval x (e.g., from t_0 to $t_0 + x$) depends on the origin of the interval t_0 as expressed in (12).

Equation

$$F_j(d_j t_0 + \lambda_j + (i + 1)/g_j) - F_j(d_j t_0 + \lambda_j + i/g_j) \tag{13}$$

gives the probability that element j activated at t_0 fails after performing exactly i work portions, which is equivalent to the probability that element j fails between $t_0 + \lambda_j + i/g_j$ and $t_0 + \lambda_j + (i + 1)/g_j$.

$F_j(d_j t_0)$ and $F_j(d_j t_0 + \lambda_j) - F_j(d_j t_0)$ are respectively probability that standby element j , which should be activated at t_0 , fails in standby mode, or during the replacement procedure.

Equation

$$1 - F_j(d_j t_0 + \lambda_j + (\mu_h + m - \alpha_h) / g_j). \tag{14}$$

gives the probability that standby element j , which is activated at t_0 after h backups, completes the mission successfully.

4.3 Distribution of Performed Work and Time for an Element Sequence

Let (h_k, Y_k) be a pair of random values, where h_k represents the number or index of the last backup procedure completed by the element sequence $s(1), \dots, s(k)$, and Y_k represents the number or index of the time interval when the last element from this sequence fails. By definition, $(h_0, Y_0) = (0, 0)$ since the mission starts at time 0 and no backups were performed before time 0. The pair (h_k, Y_k) is characterized by its probability mass function (*pmf*), defined by matrix $Q_k = \{Q_k(a, b)\}$ where $Q_k(a, b) = \Pr\{h_k = a, Y_k = b\}$.

Consider element $s(k)$ that is activated in the time interval Y_{k-1} given the index of the last completed backup is h_{k-1} . The element can operate without failure until mission completion, or until T_{\max} when the mission is terminated due to reaching the time limit or deadline. Therefore element $s(k)$ should perform no more than $m - \alpha_{h_{k-1}} + \mu_{h_{k-1}}$ remaining work portions, and no more than $[(T_{\max} - \tau Y_{k-1} - \lambda_{s(k)})g_{s(k)}]$ work portions (corresponding to the remaining mission time). Thus given (h_{k-1}, Y_{k-1}) , Eq. (15) gives the maximum number of work portions that can be performed by element $s(k)$.

$$\zeta_{s(k)}(h_{k-1}, Y_{k-1}) = \min\{m - \alpha_{h_{k-1}} + \mu_{h_{k-1}}, [(T_{\max} - \tau Y_{k-1} - \lambda_{s(k)})g_{s(k)}]\}. \tag{15}$$

The following four cases of $(h_{k-1}, Y_{k-1}) \rightarrow (h_k, Y_k)$ transition are possible.

Case 1: If element $s(k)$, which is activated at time interval Y_{k-1} after the h_{k-1} -th completed backup, fails immediately after executing $i \leq \zeta_{s(k)}(h_{k-1}, Y_{k-1})$ work portions, then

$$h_k = \varphi(\alpha_{h_{k-1}} + \max(0, i - \mu_{h_{k-1}})), Y_k = Y_{k-1} + \psi(\lambda_{s(k)} + i/g_{s(k)}), \tag{16}$$

and the occurrence probability of this event is

$$\begin{aligned} & Q_{k-1}(h_{k-1}, Y_{k-1}) \times (F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)} + (i+1)/g_{s(k)}) \\ & - F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)} + i/g_{s(k)})). \end{aligned} \quad (17)$$

Case 2: If element $s(k)$, which is activated at time interval Y_{k-1} after the h_{k-1} -th backup, does not fail before performing the remaining $\zeta_{s(k)}(h_{k-1}, Y_{k-1})$ work portions, then it is switched off. In this case,

$$h_k = \varphi(\alpha_{h_{k-1}} + \max(0, \zeta_{s(k)}(h_{k-1}, Y_{k-1}) - \mu_{h_{k-1}})), \quad (18)$$

$$Y_k = Y_{k-1} + \psi(\lambda_{s(k)} + \zeta_{s(k)}(h_{k-1}, Y_{k-1})/g_{s(k)}), \quad (19)$$

and the occurrence probability of this event is

$$Q_{k-1}(h_{k-1}, Y_{k-1})(1 - F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)} + (\zeta_{s(k)}(h_{k-1}, Y_{k-1}) + 1)/g_{s(k)})). \quad (20)$$

Case 3: If element $s(k)$ fails in the standby mode before being activated given (h_{k-1}, Y_{k-1}) , then

$$h_k = h_{k-1}, Y_k = Y_{k-1}, \quad (21)$$

and the occurrence probability of this event is

$$Q_{k-1}(h_{k-1}, Y_{k-1})F_{s(k)}(d_{s(k)}Y_{k-1}\tau). \quad (22)$$

Case 4: If element $s(k)$ fails during the replacement procedure given (h_{k-1}, Y_{k-1}) , then

$$h_k = h_{k-1}, Y_k = Y_{k-1} + \psi(\lambda_{s(k)}), \quad (23)$$

and the occurrence probability of this event is

$$Q_{k-1}(h_{k-1}, Y_{k-1})(F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)}) - F_{s(k)}(d_{s(k)}Y_{k-1}\tau)). \quad (24)$$

For example, as illustrated in Fig. 3, for the first activated element $s(1)$ $h_{k-1} = Y_{k-1} = \lambda_{s(1)} = 0$. Thus, if the first element fails in the i -th time interval, $h_1 = \varphi(i)$, $Y_1 = \psi(i/g_{s(1)})$ and $Q_1(h, i) = F_{s(1)}((i+1)/g_{s(1)}) - F_{s(1)}(i/g_{s(1)})$ when $h = \varphi(i)$, $i \leq \min(m, Y_{\max})$ and $Q_1(h, i) = 0$ otherwise.

In summary, having Q_{k-1} and $F_{s(k)}(t)$, Q_k can be obtained using the following iterative algorithm.

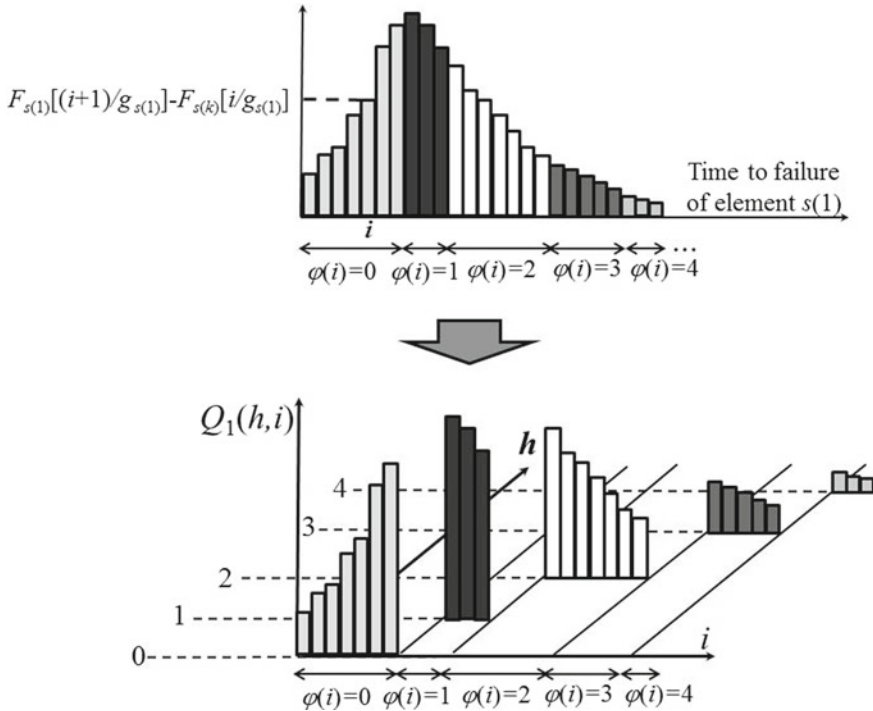


Fig. 3 Obtaining joint pmf $Q_1(h, i)$ based on time-to-failure distribution of element $s(1)$

Algorithm 1.

1. Initialize $Q_k(h, Y) = 0$ for $h = 0, \dots, H+1$; $Y = 0, \dots, Y_{\max}$
2. For $h = 0, \dots, H$:
 - 2.1. For $Y = 0, \dots, Y_{\max} - 1$:
 - 2.1.1. $Q_k(h, Y) = Q_k(h, Y) + Q_{k-1}(h, Y)F_{s(k)}(d_{s(k)}Y\tau)$. //Case 3
 - 2.1.2. $z = \min(Y + \psi(\lambda_{s(k)}), Y_{\max})$;
 - 2.1.3. $Q_k(h, z) = Q_k(h, z) + Q_{k-1}(h, Y)(F_{s(k)}(d_{s(k)}Y\tau + \lambda_{s(k)}) - F_{s(k)}(d_{s(k)}Y\tau))$. //Case 4
 - 2.1.4. $\rho = h$;
 - 2.1.5. For $i = 0, \dots, \zeta_{s(k)}(h, Y)$:
 - 2.1.5.1. If $(\alpha_h + \max(0, i - \mu_h) \geq \alpha_{\rho+1})$ $\rho = \rho + 1$;
 - 2.1.5.2. $z = Y + \psi(\lambda_{s(k)} + i/g_{s(k)})$; $q = d_{s(k)}Y\tau + \lambda_{s(k)}$;
 - 2.1.5.3. $Q_k(\rho, z) = Q_k(\rho, z) + Q_{k-1}(h, Y)(F_{s(k)}(q + (i+1)/g_{s(k)}) - F_{s(k)}(q + i/g_{s(k)}))$; //Case 1
 - 2.1.6. $Q_k(\rho, z) = Q_k(\rho, z) + Q_{k-1}(h, Y)(1 - F_{s(k)}(q + (\zeta_{s(k)}(h, Y) + 1)/g_{s(k)}))$. //Case 2

4.4 Evaluation of Mission Reliability, Expected Completion Time

$Q_k(H + 1, Y)$ represents the probability that the element sequence $s(1), \dots, s(k)$ completes the entire mission at time τY . Notice that for some values of Y , particularly, for $Y < Y_{\min} = \psi (m / \max_{1 \leq j \leq N} g_j)$, $Q_k(H + 1, Y) = 0$. The reason is that the mission time cannot be less than the time required by the fastest element to accomplish the entire mission task without failures.

In Algorithm 1, each Q_k is obtained using $Q_{k-1}(h, Y)$, for h running from 0 to H and Y running from 0 to $Y_{\max} - 1$. $Q_{k-1}(H + 1, Y)$, and $Q_{k-1}(h, Y_{\max})$ are excluded from consideration. The reason is that $Q_{k-1}(H + 1, Y)$ corresponds to a successful completion of the entire mission by elements $s(1), s(2), \dots, s(k - 1)$, and $Q_{k-1}(h, Y_{\max})$ for $h < H + 1$ corresponds to failure of the mission. Under both of these cases, elements $s(k), \dots, s(N)$ are not activated. Hence, $Q_k(H + 1, Y)$ gives the probability that elements $s(1), \dots, s(k)$ successfully complete the mission at time τY given that elements $s(1), \dots, s(k - 1)$ cannot complete the mission: $\Pr\{h_k = H + 1 | h_{k-1} < H + 1 \cup Y_k < Y_{\max}\}$.

The mission reliability is evaluated as a sum of probabilities of mutually exclusive events:

$$R = \Pr\{h_1 = H + 1 | Y_1 < Y_{\max}\} + \Pr\{h_2 = H + 1 | Y_2 < Y_{\max}, h_1 < H + 1\} + \dots + \Pr\{h_N = H + 1 | Y_N < Y_{\max}, h_{N-1} < H + 1\}. \tag{25}$$

Having $Q_k(H + 1, Y)$ for $1 \leq k \leq N$, the mission reliability is

$$R = \sum_{k=1}^N \sum_{Y=Y_{\min}}^{Y_{\max}} Q_k(H + 1, Y), \tag{26}$$

and the conditional expected mission completion time conditioned on a successful mission is

$$D = \frac{\tau}{R} \sum_{k=1}^N \sum_{Y=Y_{\min}}^{Y_{\max}} Y Q_k(H + 1, Y). \tag{27}$$

4.5 Evaluation of Expected Mission Operation Cost

The following gives the operation cost associated with using element $s(k)$ for each of the four cases described in Sect. 4.3.

Case 1:

$$w_{s(k)}i/g_{s(k)} + z_{s(k)}Y_{k-1}\tau + v_{s(k)}. \quad (28)$$

The probability of this event is given in (17). $Y_{k-1} = Y_0 = v_{s(1)} = 0$ for $k = 1$, and thus the operation cost associated with using element $s(1)$ is $w_{s(1)}i/g_{s(1)}$.

Case 2:

$$z_{s(k)}Y_{k-1}\tau + w_{s(k)}\zeta_{s(k)}(h_{k-1}, Y_{k-1})/g_{s(k)} + v_{s(k)}. \quad (29)$$

The probability of this event is given in (20).

Case 3:

$$z_{s(k)} \int_0^{\tau Y_{k-1}} f_{s(k)}(d_{s(k)}t) dt, \quad (30)$$

where $f_{s(k)}(t)$ is the probability density function (*pdf*) of time-to-failure of element $s(k)$. This expression can be approximated as

$$\Theta_{s(k)}(Y_{k-1}) = z_{s(k)}\tau \sum_{i=0}^{Y_{k-1}} i \{F_{s(k)}(d_{s(k)}(i+1)\tau) - F_{s(k)}(d_{s(k)}i\tau)\}. \quad (31)$$

The probability of this event is given in (22).

Case 4:

$$z_{s(k)}Y_{k-1}\tau + v_{s(k)}. \quad (32)$$

The probability of this event is given in (24).

Integrating these four cases, the expected operation cost associated with using element $s(k)$ given (h_{k-1}, Y_{k-1}) for $h_{k-1} < H + 1$ and $Y_{k-1} < Y_{\max}$ (implying the mission is not completed by the previous $k - 1$ elements) is

$$\begin{aligned} E_{s(k)}(h_{k-1}, Y_{k-1}) = & Q_{k-1}(h_{k-1}, Y_{k-1})\{\Theta_{s(k)}(Y_{k-1}) \\ & + (z_{s(k)}Y_{k-1}\tau + v_{s(k)})(F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)}) \\ & - F_{s(k)}(d_{s(k)}Y_{k-1}\tau)) + w_{s(k)}/g_{s(k)} \sum_{i=0}^{\zeta_{s(k)}(h_{k-1}, Y_{k-1})} i\varepsilon(i) \\ & + (z_{s(k)}Y_{k-1}\tau + v_{s(k)}) \sum_{i=0}^{\zeta_{s(k)}(h_{k-1}, Y_{k-1})} \varepsilon(i) \\ & + (z_{s(k)}Y_{k-1}\tau + w_{s(k)}\zeta_{s(k)}(h_{k-1}, Y_{k-1})/g_{s(k)} + v_{s(k)})\vartheta\}, \quad (33) \end{aligned}$$

where

$$\begin{aligned} \varepsilon(i) = & (F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)} + (i + 1)/g_{s(k)}) \\ & - F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)} + i/g_{s(k)})), \end{aligned} \tag{34}$$

$$\vartheta = (1 - F_{s(k)}(d_{s(k)}Y_{k-1}\tau + \lambda_{s(k)} + \zeta_{s(k)}(h_{k-1}, Y_{k-1})/g_{s(k)})). \tag{35}$$

If any element $s(j)$ for $j < k$ completes the mission at time interval $Y \leq Y_{\max}$ or works until the maximum allowed time interval Y_{\max} , then element $s(k)$ is not activated and remains in the standby mode until the end of the mission. Equation (36) gives the expected costs of using element $s(k)$ in this case.

$$\begin{aligned} E_{WS}(s(k)) = & \sum_{i=1}^{k-1} \sum_{Y=0}^{Y_{\max}} Q_i(H + 1, Y) (\Theta_{s(k)}(Y) + \Gamma_{s(k)}(Y)) \\ & + \sum_{i=1}^{k-1} (\Theta_{s(k)}(Y_{\max}) + \Gamma_{s(k)}(Y_{\max})) \sum_{h=0}^H Q_i(h, Y_{\max}). \end{aligned} \tag{36}$$

In (36), $\Theta_{s(k)}(Y)$ and $\Gamma_{s(k)}(Y)$ are expected costs of using element $s(k)$ when it fails, and when it does not fail in the standby mode before the end of the mission respectively, given the mission ends in the time interval Y . $\Theta_{s(k)}(Y)$ can be obtained using (31). Because element $s(k)$ is switched off in interval Y , $\Gamma_{s(k)}(Y)$ is obtained as

$$\Gamma_{s(k)}(Y) = z_{s(k)} (1 - F(d_{s(k)}Y\tau)) Y\tau. \tag{37}$$

The total expected cost of using element $s(k)$ is thus

$$E_{\text{tot}}(s(k)) = E_{WS}(s(k)) + \sum_{h=0}^H \sum_{Y=0}^{Y_{\max}-1} E_{s(k)}(h, Y). \tag{38}$$

The total expected mission operation cost for the standby system with N elements is thus

$$C = \sum_{k=1}^N E_{\text{tot}}(s(k)). \tag{39}$$

4.6 Summary of Evaluation Algorithm

Algorithm 2 gives the pseudo code of the numerical evaluation algorithm for analyzing mission reliability R , expected mission completion time D , and expected mission cost C for 1-out-of- N : G standby systems subject to backups.

Algorithm 2.

1. Given π_h ($h=1, \dots, H$) and functions b, u , determine α_h and μ_h using (1), (2), (8)-(11);
2. Initialize $v_{s(1)} = \lambda_{s(1)} = R = D = C = 0$;
3. For $h=1, \dots, H$: For $Y=0, \dots, Y_{\max}$: Initialize $Q_0(h, Y) = 0$; $Q_0(0, 0) = 1$;
4. For $k=1, \dots, N$: For $Y=0, \dots, Y_{\max}$: Compute $\Theta_k(Y)$ using (31) for given *cdf* $F_k(t)$;
5. For $k=1, \dots, N$:
 - 5.1. Initialize $Q_k(h, Y) = 0$ for $h=0, \dots, H+1$; $Y=0, \dots, Y_{\max}$
 - 5.2. For $h=0, \dots, H$:
 - 5.2.1. For $Y=0, \dots, Y_{\max}-1$:
 - 5.2.1.1. $\Phi = F_{s(k)}(d_{s(k)} Y \tau)$; $\Omega = F_{s(k)}(d_{s(k)} Y \tau + \lambda_{s(k)})$;
 - 5.2.1.2. $Q_k(h, Y) = Q_k(h, Y) + Q_{k-1}(h, Y) \Phi$;
 - 5.2.1.3. $C = C + Q_{k-1}(h, Y) \Theta_{s(k)}(Y)$;
 - 5.2.1.4. $z = \min(Y + \psi(\lambda_{s(k)}), Y_{\max})$;
 - 5.2.1.5. $Q_k(h, z) = Q_k(h, z) + Q_{k-1}(h, Y) (\Omega - \Phi)$;
 - 5.2.1.6. $C = C + (z_{s(k)} Y \tau + v_{s(k)}) Q_{k-1}(h, Y) (\Omega - \Phi)$;
 - 5.2.1.7. $\rho = h$;
 - 5.2.1.8. For $i=0, \dots, \zeta_{s(k)}(h, Y)$:
 - 5.2.1.8.1. If $(\alpha_h + \max(0, i - \mu_h) \geq \alpha_{\rho+1}) \rho = \rho + 1$;
 - 5.2.1.8.2. $z = Y + \psi(\lambda_{s(k)} + i/g_{s(k)})$;
 - 5.2.1.8.3. $\Phi = F_{s(k)}(d_{s(k)} Y \tau + \lambda_{s(k)} + (i+1)/g_{s(k)})$;
 - 5.2.1.8.4. $Q_k(\rho, z) = Q_k(\rho, z) + Q_{k-1}(h, Y) (\Phi - \Omega)$;
 - 5.2.1.8.5. $C = C + (z_{s(k)} Y \tau + v_{s(k)} + w_{s(k)} i/g_{s(k)}) Q_{k-1}(h, Y) (\Phi - \Omega)$;
 - 5.2.1.8.6. $\Omega = \Phi$;
 - 5.2.1.8.7. $Q_k(\rho, z) = Q_k(\rho, z) + Q_{k-1}(h, Y) (1 - \Omega)$;
 - 5.2.1.8.8. $C = C + (z_{s(k)} Y \tau + v_{s(k)} + w_{s(k)} \zeta_{s(k)}(h, Y)/g_{s(k)}) Q_{k-1}(h, Y) (1 - \Omega)$;
 - 5.2.1.9. For $Y = Y_{\min}, \dots, Y_{\max}$:
 - 5.2.1.9.1. $R = R + Q_k(H+1, Y)$;
 - 5.2.1.9.2. $D = D + \tau Y Q_k(H+1, Y)$;
 - 5.2.1.9.3. $C = C + Q_k(H+1, Y) \sum_{j=k+1}^N (\Theta_{s(j)}(Y) + \Gamma_{s(j)}(Y))$;
 - 5.2.2. $C = C + \sum_{h=0}^H Q_k(h, Y_{\max}) \cdot \sum_{j=k+1}^N \Theta_{s(j)}(Y_{\max}) + \Gamma_{s(j)}(Y_{\max})$.

The computational complexity of Algorithm 2 is $O(NHY_{\max}m)$.

5 Examples of Mission Performance Indices Evaluation

Consider a 1-out-of-5 standby clustered computer system using nonidentical processors, which can perform the same computational task. Depending on processor type, their locations (ambient conditions) and exploitation history, the five processors have different performance (computation speeds) and different time-to-failure distributions. Due to the sequential nature of the computational task, when one of the processors works on the mission task, the remaining ones wait in an idle mode or perform other low-priority tasks. Based on a prespecified sequence, the standby processors take over the computational task in the case of failure of the operating processor.

Time-to-failure of each element j follows a Weibull distribution with scale parameter η_j and shape parameter β_j . The exponential distribution appears as a special case of the Weibull distribution with $\beta_j = 1$. Equation (40) gives the *cdf* of the Weibull distribution.

$$F_j(i) = 1 - \exp \left\{ - \left[\Delta i / \eta_j \right]^{\beta_j} \right\}. \tag{40}$$

Table 1 presents values of η_j and β_j , deceleration factor d_j , replacement cost v_j , per time unit standby cost z_j , per time unit operation cost w_j , replacement time λ_j , and processing speed G_j for each element j ($j = 1, 2, 3, 4, 5$).

Table 2 presents values of mission parameters M, H, π_i and T_{\max} . The total backup technique in (1) is assumed for the example. Equation (41) defines the number of operations required for the h -th data backup and retrieval as linear functions of the total amount of work accomplished before the backup. Values of coefficients b_1, b_2, u_1 , and u_2 are given in Table 2.

$$B_h = b_1 + b_2 M \left(\sum_{j=1}^h \pi_j \right), \quad U_h = u_1 + u_2 M \left(\sum_{j=1}^h \pi_j \right). \tag{41}$$

Table 1 Element parameters for the example standby system

Element	η_j	β_j	d_j	v_j	z_j	w_j	λ_j	G_j
1	130	1.0	0.3	60	2.0	9.0	24	110
2	140	1.1	0.5	54	1.7	6.0	15	92
3	165	1.2	0.15	50	1.4	6.0	37	85
4	200	1.0	0.2	50	1.0	5.0	30	78
5	230	1.1	0.4	43	1.2	4.0	20	60

Table 2 Mission parameters for the example standby system

M	T_{\max}	H	π_1	π_2	π_3	u_1	u_2	b_1	b_2
10,000	400	3	0.3	0.3	0.15	100	0.2	100	0.2

The minimal total number of operations required to complete the mission (including $H = 3$ backups) is $M + B_1 + B_2 + B_3 = 13,600$. Thus, the minimal mission time given the fastest element 1 activated at the beginning of the mission completes the mission task successfully is $T_{\min} = (M + B_1 + B_2 + B_3)/G_1 = 13,600/110 = 123.6$.

Assume the five elements are activated in an increasing numerical order, the mission reliability, expected total cost, and expected time obtained using Algorithm 2 are $R = 0.91$, $C = 1866$, and $D = 186.7$, respectively.

5.1 Effect of Maximum Allowed Mission Time T_{\max}

Figure 4 presents mission reliability R , expected mission cost C , and expected mission completion time D as functions of deadline T_{\max} . Values of the rest of the mission parameters are the same as those in Table 2.

For $T_{\max} < T_{\min} = 123.6$, $R = 0$ simply because there is no chance to accomplish the mission in time less than T_{\min} . However, mission cost is nonzero since some elements are activated and have worked until time T_{\max} .

For $T_{\min} < T_{\max} < 161$, R is equal to the probability that the first fastest element completes the mission, and $D = T_{\min}$. The reason is that $T_2 = 161$ is the minimal time needed by the second fastest element 2 to replace element 1 (when element 1 fails at time 0) and to accomplish the mission. Thus, only the first fastest element is able to accomplish the mission in time less than 161.

When T_{\max} becomes greater than the maximal possible time of mission completion 523.6, the value of T_{\max} does not affect R , C , and D any more as shown in Fig. 4.

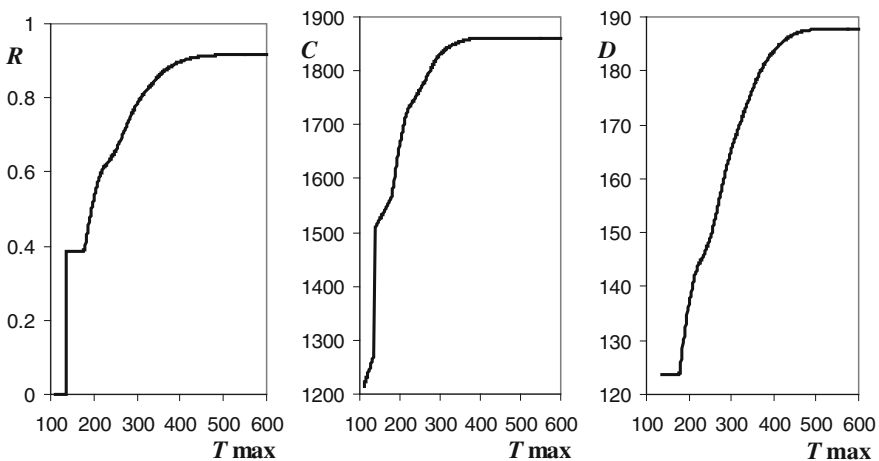


Fig. 4 Effects of T_{\max} on R , C , and D [25]

5.2 Effect of Even Backup Distribution

Assume backups are evenly distributed, i.e., $\pi_i = \pi$ for all i . This section studies effects of even backup distribution indicated by the value of π on the mission performance. Figure 5 presents the number of backups H , mission reliability R , expected cost C , and time D as functions of π . Values of the rest of the mission parameters are the same as those in Table 2.

In this case, $H = \lfloor 1/\pi \rfloor$ represents the greatest integer not exceeding $1/\pi$. Due to two-fold impacts of π on mission performance indices, mission reliability, expected completion time, and expected total cost are non-monotonic functions of π as shown in Fig. 5. Specifically, on the one hand, increasing the value of π leads to less frequent backups, increasing the work to be re-performed when an element fails. On the other hand, an increase in π results in a reduction in the number of backups H during the mission, which can reduce the total time required to complete the entire mission.

Abrupt jumps in functions $R(\pi)$, $C(\pi)$, and $D(\pi)$ take place when the value of H changes at $\pi = 1/y$ with y being an integer, corresponding to the $H = y - 1$ backups throughout the mission. If π is decreased by a negligibly small value, the value of H changes immediately from $y - 1$ to y , causing a sharp increase in the minimal possible mission time. Because the change in π 's value is negligible, there is no considerable variation in work portions that should be re-done in the case of an element failure occurring. Hence, due to the decrease in the value of H the mission reliability increases and the expected mission time and total cost decrease abruptly. In the case of even backups, the maximal mission reliability and the minimal expected mission time and total cost are always achieved when $\pi = 1/(H + 1)$.

For specific examples, $\pi = 0.2499$ implies $H = 4$ backups with the last backup being conducted when 99.96% of the entire mission task is accomplished; $\pi = 0.25$

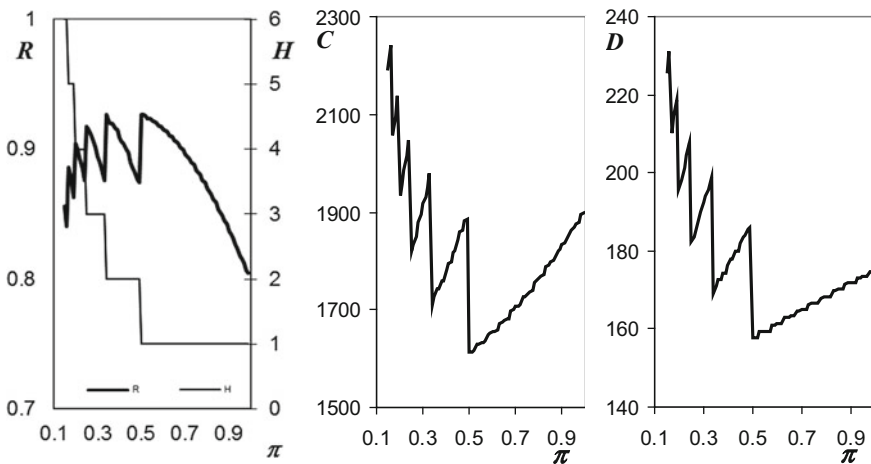


Fig. 5 Effects of π on H , R , C , and D [25]

implies $H = 3$ backups with the last backup being performed when 75% of the entire mission task is accomplished. Performing the fourth backup when the mission is close to finish is ineffective because the probability that the mission fails after this backup is negligible. Therefore, when π increases from 0.2499 to 0.25, the expected mission completion time and total cost drop abruptly while the mission reliability increases drastically.

5.3 Effect of Element Scale Parameter η_j

Figure 6 illustrates mission reliability R , expected cost C , and time D as functions of element j 's scale parameter η_j for $j = 1, 2$, and 4. When η_j for a certain element changes, the rest of the element parameters remain unchanged and their values are taken from Table 1. The same values of mission parameters as in Table 2 are used.

When the first element fails between the completion of the $n - 1$ -th and n -th backup procedures, the next element continues the mission task from the work portion that follows termination of the $n - 1$ -th backup procedure regardless of the time when the first element fails. Thus, the sooner the first element fails, the sooner the standby element activation starts, which decreases the time it spends in the standby mode. The overall exposure time of the next element needed to complete the mission decreases and the total time remaining for this element to complete the mission increases. Thus, the next element has a greater chance to complete the mission. Figure 7 presents two mission scenarios in the “performed work”—“cumulated exposure time” space. In case A, the first element $s(1)$ fails later than in case B, though in both cases it fails

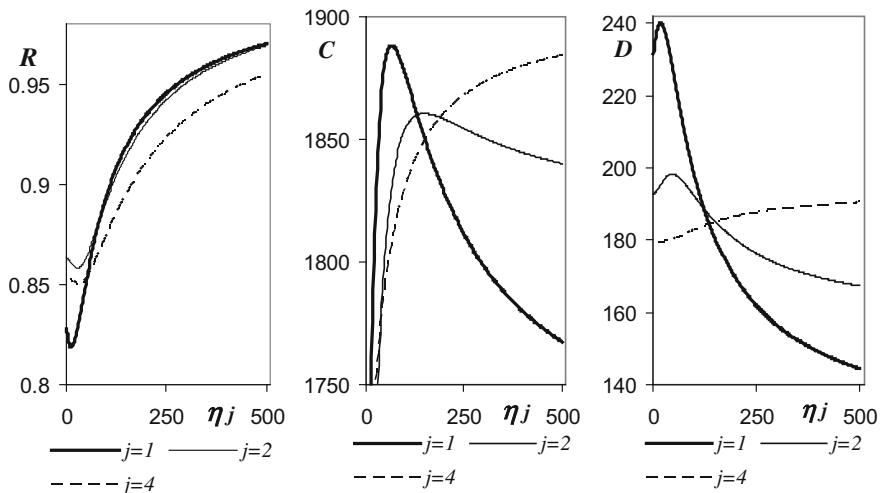


Fig. 6 Effects of η_j on R , C , and D [25]

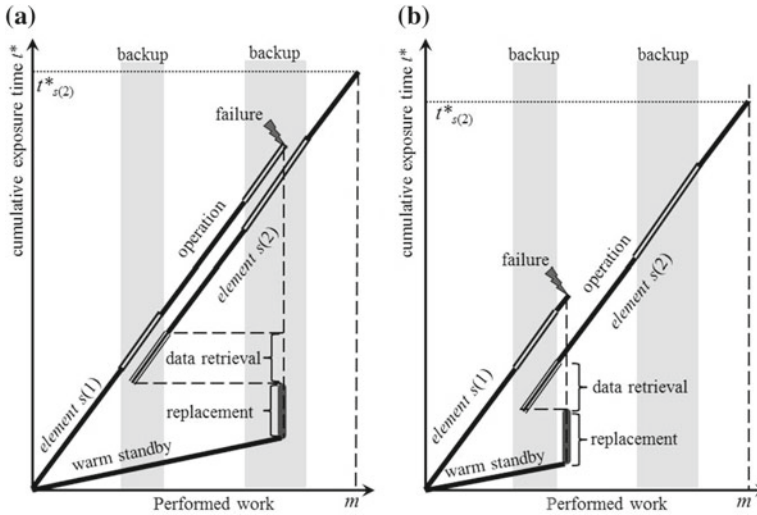


Fig. 7 Two scenarios of mission execution by two elements (**a** late failure of the first element; **b** early failure of the first element)

between completion of the first and the second backups. The cumulative exposure time $t^*_{s(2)}$ of element $s(2)$ in case A is greater than that in case B. Thus, in case A element $s(2)$ has lower chances to complete the mission without failure. Moreover, since in case A element $s(2)$ is activated later (it has to perform the same amount of work as in case B though), the time remaining before the mission termination in case A is less than that of case B and the element may not have enough time to complete the mission in case A. Therefore, when the first element is very unreliable, an increment in its reliability (or its expected failure time) leads to a decrease in the entire mission reliability as well as an increase in the expected mission time and cost. When the reliability of the first element continues to increase, the probability that the first element completes several backups and even completes the entire mission task increases. This change results in a decrease in the expected mission cost and time, and an increase in the mission reliability. Such a non-monotonic functional dependence effect of mission reliability, time, and cost on a single element's reliability (usually referred to as system noncoherency) is more distinguished for elements activated earlier than for elements activated later. The reason for this effect is that the elements that should be activated later have larger chances to fail in the standby mode before being activated if they are more unreliable. Refer to [22] for studies of such noncoherency of standby systems with backups.

6 Standby System Operation Optimization Problems

In this section, application of the presented numerical evaluation algorithm to the solution of several optimization problems is discussed.

6.1 Optimal Backup Distribution Problem

As discussed in Sect. 5.2, mission reliability, expected time, and cost depend on the number of backups H and their distribution $\boldsymbol{\pi}$ in a non-monotonic manner. Hence the optimal backup distribution problem can be formulated for the standby system subject to backups as: Find backup distribution vector $\boldsymbol{\pi}$ that maximizes R (or minimizes C or minimizes D), subject to constraints on the rest of the mission performance indices. Equation (42) presents two examples of such optimization problems.

$$\begin{aligned} \min C(\boldsymbol{\pi}) \text{ s.t. } R(\boldsymbol{\pi}) &\geq R^*, D(\boldsymbol{\pi}) \leq D^*, \\ \max R(\boldsymbol{\pi}) \text{ s.t. } C(\boldsymbol{\pi}) &\leq C^*, D(\boldsymbol{\pi}) \leq D^*. \end{aligned} \quad (42)$$

Using Algorithm 2 in Sect. 4.6, the above optimization problems can be solved using any algorithm applicable to multidimensional optimization. The GA [16] is applied in this work, which generates K numbers π_1, \dots, π_K such that each π_h belongs to interval $(0, 1.1)$. Value of H is decided by conditions $\sum_{h=1}^H \pi_h \leq 1$, $\sum_{h=1}^{H+1} \pi_h > 1$ and can vary from 0 to K ($K = 8$ is used in this work). For each vector $\boldsymbol{\pi}$ generated by GA, $C(\boldsymbol{\pi})$, $R(\boldsymbol{\pi})$, and $D(\boldsymbol{\pi})$ are evaluated using Algorithm 2. The objective function minimized by the GA is

$$\mathcal{E} = \xi_c \max(0, C(\boldsymbol{\pi}) - C^*) + \xi_r \max(0, R^* - R(\boldsymbol{\pi})) + \xi_d \max(0, D(\boldsymbol{\pi}) - D^*) \quad (43)$$

where ξ_c, ξ_r, ξ_d are penalty coefficients. The expected mission cost minimization, mission reliability maximization, and expected mission time minimization problems are special cases of (43) when $\xi_r = \xi_d = C^* = 0$, $\xi_c = \xi_d = 0$ and $R^* = 1$, $\xi_r = \xi_c = D^* = 0$, respectively.

Table 3 presents optimal solutions for the example 1-out-of-5 standby system with elements being activated in an increasing numerical order.

Solutions to the optimal backup distribution problems using different backup times reveal that as the data backup time increases, the optimal value of H always decreases and eventually becomes zero, and the optimal value of H for maximizing R is usually larger than that for minimizing C and D .

Table 3 Optimal backup distribution solutions [25]

Type of problem	R	C	D	H	π_1	π_2
max R	0.935	1644	161.5	2	0.24	0.29
min C	0.922	1560	150.4	1	0.22	–
min D	0.890	1578	148.7	0	–	–
max R s.t. $C < 1600$	0.932	1599	155.8	2	0.15	0.22
min D s.t. $R > 0.91, C < 1580$	0.910	1573	150.4	1	0.14	–
min C s.t. $R > 0.93, D < 155$	0.931	1574	153.2	1	0.37	–

6.2 Optimal Element Activation Sequencing Problem

In the case of nonidentical system elements, the optimal element activation sequencing problem is relevant and is formulated as follows. Find sequence $s(1), s(2), \dots, s(N)$ that maximizes R (minimizes C or D) subject to constraints on the rest of mission performance indices. When N is small, a brute-force enumeration of all possible permutations of numbers $1, 2, \dots, N$ can be used; when N is large, some heuristic algorithms can be used [15, 16] instead. Table 4 gives sample solutions of the optimal activation sequencing problem for the 1-out-of-5 standby system with the parameters given in Tables 1 and 2.

6.3 Optimal Backup Distribution and Element Sequencing Problem

Table 5 presents sample solutions to integrated optimization problems that find combinations of vector π and sequence $s(1), s(2), \dots, s(N)$ maximizing R (minimizing C or D) subject to constraints on the rest of mission performance indices for the example 1-out-of-5 standby system. It can be observed that the fastest elements 1 and 2 tend to be activated first when R or D is the main concern; the inexpensive

Table 4 Optimal element activation sequencing solutions for fixed π

Type of problem	R	C	D	Sequence
max R	0.918	1984	190.0	1, 2, 5, 4, 3
min C	0.897	1810	218.1	4, 2, 5, 3, 1
min D	0.911	1975	185.0	1, 2, 4, 3, 5
min C s.t. $R > 0.91, D < 210$	0.911	1959	206.9	2, 1, 5, 4, 3
max R s.t. $C < 1980, D < 187$	0.916	1965	186.5	1, 2, 4, 5, 3

Table 5 Optimal element activation sequencing and backup distribution solutions

Type of problem	R	C	D	Sequence	H	π_1	π_2
max R	0.940	770	163.9	1, 2, 5, 4, 3	2	0.25	0.27
min C	0.914	1505	175.6	4, 2, 5, 3, 1	1	0.22	–
min D	0.893	1692	147.6	1, 2, 4, 3, 5	0	–	–
min C s.t. $D < 150$, $R > 0.91$	0.917	1677	148.6	1, 2, 4, 3, 5	1	0.17	–
max R s.t. $C < 1680$, $D < 150$	0.917	1677	148.6	1, 2, 4, 3, 5	1	0.17	–

Table 6 Comparison of optimal solutions [25]

Type of problem	Optimal backup	Optimal sequencing	Integrated optimization
max R	0.935 (0.5 %)	0.918 (2.3 %)	0.940
min C	1560 (3.7 %)	1810 (20.3 %)	1505
min D	148.7 (0.7 %)	185.0 (25.3 %)	147.6

element 4 is activated first when C is minimized; and the minimal expected mission time is obtained when no backups are used.

Table 6 compares results of max R , min C , and min D problems when only backup distribution, only element activation sequencing, or both backup distribution and element sequencing are optimized. The relative difference between solutions of each separate optimization problem and corresponding integrated optimization problem is presented within parentheses. It can be seen that the integrated optimization achieves better results than both backup distribution optimization and element activation sequencing optimization, and the backup distribution optimization provides better solutions than the element activation sequence optimization for the cases considered.

7 Conclusion

This chapter presents a numerical method that evaluates mission reliability, expected completion time, and expected total cost of 1-out-of- N : G standby sparing systems subject to backups. The method is applicable to systems with even or uneven backups, cold, hot, and warm standby modes, dynamic data backup and retrieval times, as well as nonidentical system elements. This chapter further presents formulation and solution of a set of optimization problems that is relevant to the optimal design and operation planning of heterogeneous standby systems subject to backups. As demon-

strated through examples, the proposed methodology can identify optimal decisions for system backup and standby policies, promoting reliable and cost-effective operation of real-time standby systems.

References

1. Amari SV, Dill G (2010) Redundancy optimization problem with warm-standby redundancy. In: Proceedings of annual reliability and maintainability symposium, pp 1–6
2. Amari SV, Misra KB, Pham H (2008) Tampered failure rate load-sharing systems: status and perspectives. In: Misra KB (ed) Chapter 20 in handbook of performability engineering. Springer, pp 291–308
3. Amari SV, Pham H, Misra RB (2012) Reliability characteristics of k-out-of-n warm standby systems. *IEEE Trans Reliab* 61:1007–1018
4. Boddu P, Xing L (2013) Reliability evaluation and optimization of series-parallel systems with k-out-of-n: G subsystems and mixed redundancy types. *Proc IMechE Part O, J Risk Reliab* 227(2):187–198
5. Chambari A, Rahmati S, Najafi A, Karimi A (2012) A bi-objective model to optimize reliability and cost of system with a choice of redundancy strategies. *Comput Industr Eng* 63(1):109–119
6. Chen T-C, You P-S (2005) Immune algorithms-based approach for redundant reliability problems with multiple component choices. *Comput Ind* 56(2):195–205
7. Chia LY, Smith AE (2004) An ant colony optimization algorithm for the redundancy allocation problem (RAP). *IEEE Trans Reliab* 53(3):417–423
8. Coit DW (2001) Cold-standby redundancy optimization for non-repairable systems. *IIE Trans* 33:471–478
9. Coit DW (2003) Maximization of system reliability with a choice of redundancy strategies. *IIE Trans* 35(6):535–544
10. Coit DW, Smith AE (1996) Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Trans Reliab* 45(2):254–260
11. Elerath JG, Pecht M (2009) A highly accurate method for assessing reliability of redundant arrays of inexpensive disks (RAID). *IEEE Trans Comput* 58(3):289–299
12. Fyffe DE, Hines WW, Lee NK (1968) System reliability allocation and a computation algorithm. *IEEE Trans Reliab* 17:64–69
13. Hsieh C, Hsieh Y (2003) Reliability and cost optimization in distributed computing systems. *Comput Oper Res* 30:1103–1119
14. Johnson BW (1989) Design and analysis of fault tolerant digital systems. Addison-Wesley (1989)
15. Kuo W, Wan R (2007) Recent advances in optimal reliability allocation. *IEEE Trans Syst Man Cybern Part A Syst Hum* 37(2):143–156
16. Levitin G (2006) Genetic algorithms in reliability engineering. Guest editorial. *Reliab Eng Syst Saf* 91(9):975–976 (2006)
17. Levitin G, Xing L, Dai Y (2013) Optimal sequencing of warm standby elements. *Comput Ind Eng* 65:570–576
18. Levitin G, Xing L, Dai Y (2013) Cold-standby sequencing optimization considering mission cost. *Reliab Eng Syst Saf* 118:28–34
19. Levitin G, Xing L, Dai Y (2013) Sequencing optimization in k-out-of-n cold-standby systems considering mission cost. *Int J Gen Syst* 42(8):870–882
20. Levitin G, Xing L, Dai Y (2014) Minimum mission cost cold-standby sequencing in non-repairable multi-phase systems. *IEEE Trans Reliab* 63(1):251–258
21. Levitin G, Xing L, Dai Y (2014) Mission cost and reliability of 1-out-of-N warm standby systems with imperfect switching mechanisms. *IEEE Trans Syst Man Cybern Syst* 44(9):1262–1271

22. Levitin G, Xing L, Dai Y (2015) Reliability of non-coherent warm standby systems with reworking. *IEEE Trans Reliab* 64(1):444–453
23. Levitin G, Xing L, Johnson BW, Dai Y (2015) Mission reliability, cost and time for cold standby computing systems with periodic backup. *IEEE Trans Comput* 64(4):1043–1057
24. Levitin G, Xing L, Dai Y (2015) Optimal backup distribution in 1-out-of-N cold standby systems. *IEEE Trans Syst Man Cybern Syst* 45(4):636–646
25. Levitin G, Xing L, Dai Y (2015) Heterogeneous 1-out-of-N warm standby systems with dynamic uneven backups. *IEEE Trans Reliab* 64(4):1325–1339
26. Levitin G, Xing L, Zhai Q, Dai Y (in press) Optimization of full vs. incremental periodic backup policy. *IEEE Trans Dependable Secure Comput*, doi:[10.1109/TDSC.2015.2413404](https://doi.org/10.1109/TDSC.2015.2413404)
27. Misra KB (1972) Reliability optimization of a series-parallel system. *IEEE Trans Reliab R-21*(4):230–238
28. Misra KB, Sharma U (1991) An efficient algorithm to solve integer programming problems arising in system-reliability design. *IEEE Trans Reliab* 40(1):81–91
29. Nakagawa T (2007) Shock and damage models in reliability theory, Chapter 9. Springer, London (Springer series in reliability engineering)
30. Onishi J, Kimura S, James RJW, Nakagawa Y (2007) Solving the redundancy allocation problem with a mix of components using the improved surrogate constraint method. *IEEE Trans Reliab* 56(1):94–101
31. Papageorgiou E, Kokolakis G (2010) Reliability analysis of a two-unit general parallel system with warm standbys. *Eur J Oper Res* 201(3):821–827
32. Qian C, Huang Y, Zhao X, Nakagawa T (2010) Optimal backup interval for a database system with full and periodic incremental backup. *J Comput* 5(4):557–564
33. Ruiz-Castro JE, Fernández-Villodre G (2012) A complex discrete warm standby system with loss of units. *Eur J Oper Res* 218(2):456–469
34. Sandoh H, Kaio N, Kawai H (1992) On backup policies for hard computer disks. *Reliab Eng Syst Saf* 37(1):29–32
35. Sinaki G (1994) Ultra-reliable fault tolerant inertial reference unit for spacecraft. In: *Proceedings of the annual rocky mountain guidance and control conference*. Univelt Inc., San Diego, CA, pp 239–248
36. Tannous Q, Xing L, Dugan JB (2011) Reliability analysis of warm standby systems using sequential BDD. In: *Proceedings of the 57th annual reliability and maintainability symposium*, FL, USA
37. Tannous O, Xing L, Peng R, Xie M, Ng SH (2011) Redundancy allocation for series-parallel warm-standby systems. In: *Proceedings of the IEEE international conference on industrial engineering and engineering management*, Singapore
38. Wang C, Xing L, Amari SV (2012) A fast approximation method for reliability analysis of cold-standby systems. *Reliab Eng Syst Saf* 106:119–126
39. Xing L, Tannous O, Dugan JB (2012) Reliability analysis of non-repairable cold-standby systems using sequential binary decision diagrams. *IEEE Trans Syst Man Cybern Part A Syst Hum* 42(3):715–726
40. Yang X, Wang Z, Xue J, Zhou Y (2012) The reliability wall for exascale supercomputing. *IEEE Trans Comput* 61(6):767–779
41. Zhang T, Xie M, Horigome M (2006) Availability and reliability of k-out-of-(M+N): G warm standby systems. *Reliab Eng Syst Saf* 91(4):381–387
42. Zhao R, Liu B (2004) Redundancy optimization problems with uncertainty of combining randomness and fuzziness. *Eur J Oper Res* 157(3):716–735
43. Zhao R, Liu B (2005) Standby redundancy optimization problems with fuzzy lifetimes. *Comput Ind Eng* 49(2):318–338

Reliability Analysis of a Cloud Computing System with Replication: Using Markov Renewal Processes

Mitsutaka Kimura, Xufeng Zhao and Toshio Nakagawa

Abstract Cloud computing is an important infrastructure for many industries. There are increasing needs for such new techniques in data protection, short response time, reduced management cost, etc. A cloud computing system with distributed information and communication processing capabilities has been proposed [1–3] which consists of some intelligent nodes as well as a data center. A short response time in delivering data could be made by using multiple intelligent nodes near each client rather than a data center. In order to protect client data, all of intelligent nodes need to transmit the database content to a data center via a network link, which is called replication. There are two replication strategies which are known as synchronous and asynchronous schemes [4–6]. Using techniques of Markov renewal processes, this chapter summarizes reliability analyses of a cloud computing system with the above two replications, and focuses on some optimization problems to make regular backups of client data from all of the intelligent nodes to a data center.

1 Introduction

Recently, cloud computing has been widely used in managing server and storage on the Internet [7, 8]. It is therefore unsurprising that demands for such new techniques in data protection, early recovery, short response time, and reduced management cost

M. Kimura (✉)

Department of Cross Cultural Studies, Gifu City Women's College,
7-1 Hitoichibakita-machi, Gifu 501-0192, Japan
e-mail: kimura@gifu-cwc.ac.jp

X. Zhao

Department of Mechanical and Industrial Engineering, Qatar University,
Al Tarfa, Doha 2713, Qatar
e-mail: kyokuh@qu.edu.qa

T. Nakagawa

Department of Business Administration, Aichi Institute of Technology,
1247 Yachigusa, Yakusa-cho, Toyota 470-0392, Japan
e-mail: toshi-nakagawa@aitech.ac.jp

© Springer International Publishing Switzerland 2016

L. Fiondella and A. Puliafito (eds.), *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering,
DOI 10.1007/978-3-319-30599-8_15

have increased. A cloud computing system with distributed information and communication processing consists of some intelligent nodes as well as a data center, which manages all of the client information and application software [1–3]. The intelligent node provides application service near clients, enabling the client to enjoy short response times to their requests for service [1]. This also enables high availability as a switching technique to continue the service when an intelligent node breaks down [2]. In the previous research, a messaging scheme based on service dependencies between data centers and clients has been proposed in order to reduce response times experienced by a client [9], and the parallel transmission system with Stream Control Transmission Protocol (SCTP) has been proposed in order to achieve high speed data communication between data centers and clients [3].

It is also important to consider the actions to protect client data in intelligent nodes. That is, it is necessary to consider how to make regular backups of client data from all of the intelligent nodes to a data center. The intelligent node stands by and serves as a backup when another intelligent node breaks down and all of the intelligent nodes need to transmit the database content to a data center via a network link, which is called replication. There are two ways of performing replication, namely synchronous and asynchronous schemes [4–6]. The synchronous scheme guarantees consistency of database content but also imposes a prohibitive cost. The synchronous scheme has lower costs, but it can compromise consistency [6]. This chapter summarizes reliability analyses of a cloud computing system with synchronous and asynchronous replications, using techniques of Markov renewal processes. Markov renewal processes plays an important role in the analysis of probability models with sums of independent nonnegative random variables. Trivedi discussed the renewal theory for a duplex system with two processors and Triple Modular Redundancy (TMR) system with three active units and one spare [10].

In Sect. 2, we introduce the fundamental example of Markov renewal processes [11, 12]. Let us sketch one-unit system, in which an operating unit is repaired at failures and preventive maintenance (PM) is made at suitable times. When the repair and PM are completed, the unit becomes as good as new and begins to operate. The system can be described by a Markov renewal process. In general, a Markov renewal process has the Markovian property in which the future behavior depends only on the present state and not its past history. We mention only the theory of stationary Markov chains with finite-state space for analysis of one-unit system and it is shown that one-step transition probabilities and renewal functions are given in terms of them.

In Sect. 3, we consider a reliability model of a cloud computing system with synchronous and asynchronous replications. We have formulated a stochastic model of a cloud computing system with n intelligent nodes, a data center and a monitor, in which the client service can be made to operate until one of n intelligent nodes break down. The monitor orders each intelligent node to serve the request for each client. That is, the server in the intelligent node provides the application service when a client requests the use of an application. We consider how to make regular backups of client data from all of the intelligent nodes to a data center. When the server in the intelligent node updates the data by requesting from client at times k ,

the monitor orders n intelligent nodes to transmit all of the client data to a data center, and furthermore, it orders n intelligent nodes to receive the update information of the application from a data center. That is, when $k = 1$, synchronous data replication executes from all of the intelligent nodes to a data center. When $k > 1$, asynchronous data replication executes. When one of n intelligent nodes breaks down by a failure such as a disaster or crash failure, the client service is migrated from the intelligent node to another one. We derive the expected number of replications and updating the client data in intelligent nodes before a failure of an intelligent node. Further, by using these expected numbers, we derive the expected cost and discuss an optimal replication interval to minimize it [14]. Next, we discuss an optimal policy to reduce the costs for replications while managing the intelligent nodes. That is, we discuss an optimal number of intelligent nodes to minimize the expected cost.

In Sect. 4, we consider an extended stochastic model of distributed communication processing for a cloud system with n intelligent nodes, in which the service continues to operate until all of the intelligent nodes break down. That is, the client service is migrated from the intelligent node to another one repeatedly whenever one of them breaks down, which continues until n intelligent nodes break down. We assume a server system model is identical to Sect. 3 and derive the expected number of replications and updating the client data in intelligent nodes before failures of n intelligent nodes. Further, we derive the expected cost and discuss an optimal replication interval to minimize it.

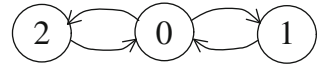
2 Markov Renewal Processes

This section briefly presents Markov renewal processes for a fundamental system with maintenance [13, p. 123]. Consider a simple one-unit system with repair and preventive maintenance (PM). Using one-step transition probabilities and renewal functions of Markov renewal processes, we derive the expected number of repairs and PMs.

2.1 One-Unit System with Repair and PM

This system consists of one operating unit that is repaired at failures and undergoes preventive maintenance at suitable times. It is assumed that the failure time of the unit has a general distribution $F(t)$ with finite mean $1/\lambda$, and the repair time has a general distribution $G_1(t)$ with finite mean $1/\mu_1$ ($0 < \mu_1 < \infty$). Furthermore, when the unit operates at a planned time T ($0 < T \leq \infty$) without failure, its operation is stopped and PM is made. The PM requires the time according to a general distribution $G_2(t)$ with finite mean $1/\mu_2$ ($0 < \mu_2 < \infty$).

Fig. 1 Transition diagram between system states



We form the stochastic model of a one-unit system with repair and PM, and derive the expected number of repairs and PMs, using techniques of Markov renewal processes [13, p. 136]. We define the following states:

- State 0: Unit is operating.
- State 1: Unit is under repair.
- State 2: Unit is under PM.

A transition diagram between system states is shown in Fig. 1.

We define the one-step transition probabilities $Q_{i,j}(t)$ from State i to State j ($i, j = 0, 1, 2$). Moreover, for convenience, we denote that $D(t)$ is a degenerate distribution placing unit mass at T ; i.e., $D(t) \equiv 0$ for $t < T$ and 1 for $t \geq T$. Then, one-step transition probabilities are

$$\begin{aligned}
 Q_{0,1}(t) &= \int_0^t \bar{D}(u) dF(u), & Q_{0,2}(t) &= \int_0^t \bar{F}(u) dD(u), \\
 Q_{1,0}(t) &= G_1(t), & Q_{2,0}(t) &= G_2(t),
 \end{aligned}$$

where $\bar{\Phi}(t) \equiv 1 - \Phi(t)$.

Let $M_{i,j}(t)$ ($i, j = 0, 1, 2$) denote the expected number of visits to State j during $(0, t]$, starting from State i . For instance, $M_{0,2}(t)$ represents the expected number of PMs during $(0, t]$, given that the unit begins to operate at time 0. Then, we have the following renewal equations:

$$\begin{aligned}
 M_{0,0}(t) &= Q_{0,1}(t) * M_{1,0}(t) + Q_{0,2}(t) * M_{2,0}(t), \\
 M_{0,1}(t) &= Q_{0,1}(t) * [1 + M_{1,1}(t)] + Q_{0,2}(t) * M_{2,1}(t), \\
 M_{0,2}(t) &= Q_{0,1}(t) * M_{1,2}(t) + Q_{0,2}(t) * [1 + M_{2,2}(t)], \\
 M_{i,0}(t) &= Q_{i,0}(t) * [1 + M_{0,0}(t)] = G_i(t) * [1 + M_{0,0}(t)] \quad (i = 1, 2), \\
 M_{1,j}(t) &= Q_{1,0}(t) * M_{0,j}(t) = G_1(t) * M_{0,j}(t), \\
 M_{2,j}(t) &= Q_{2,0}(t) * M_{0,j}(t) = G_2(t) * M_{0,j}(t) \quad (j = 1, 2).
 \end{aligned}$$

where the asterisk denotes the pairwise Stieltjes convolution; i.e., $a(t) * b(t) \equiv \int_0^t a(t-u)b(u)$, $\Phi^{(i)}(t)$ ($i = 1, 2, \dots$) denotes the i -fold convolution of any function $\Phi(t)$ and $\Phi^{(i)}(t) \equiv \Phi^{(i-1)}(t) * \Phi(t) = \int_0^t \Phi^{(i-1)}(t-u)d\Phi(u)$, $\Phi^{(0)}(t) \equiv 1$.

Forming the Laplace-Stieltjes (LS) transforms of the above equations,

$$\begin{aligned}
 M_{0,0}^*(s) &= \frac{Q_{0,1}^*(s)Q_{1,0}^*(s) + Q_{0,2}^*(s)Q_{2,0}^*(s)}{1 - Q_{1,0}^*(s)Q_{0,1}^*(s) - Q_{2,0}^*(s)Q_{0,2}^*(s)} \\
 &= \frac{G_1^*(s) \int_0^T e^{-st} dF(t) + G_2^*(s)e^{-sT} \overline{F}(T)}{1 - G_1^*(s) \int_0^T e^{-st} dF(t) - G_2^*(s)e^{-sT} \overline{F}(T)}, \\
 M_{0,1}^*(s) &= \frac{Q_{0,1}^*(s)}{1 - Q_{1,0}^*(s)Q_{0,1}^*(s) - Q_{2,0}^*(s)Q_{0,2}^*(s)} \\
 &= \frac{\int_0^T e^{-st} dF(t)}{1 - G_1^*(s) \int_0^T e^{-st} dF(t) - G_2^*(s)e^{-sT} \overline{F}(T)}, \\
 M_{0,2}^*(s) &= \frac{Q_{0,2}^*(s)}{1 - Q_{1,0}^*(s)Q_{0,1}^*(s) - Q_{2,0}^*(s)Q_{0,2}^*(s)} \\
 &= \frac{e^{-sT} \overline{F}(T)}{1 - G_1^*(s) \int_0^T e^{-st} dF(t) - G_2^*(s)e^{-sT} \overline{F}(T)},
 \end{aligned}$$

where $\Phi^*(s)$ denotes the LS transform of any function $\Phi(t)$, i.e., $\Phi^*(s) \equiv \int_0^\infty e^{-st} d\Phi(t)$ for $Re(s) > 0$.

Furthermore, the limiting values $M_j \equiv \lim_{t \rightarrow \infty} M_{0,j}(t)/t = \lim_{s \rightarrow 0} sM_{0,j}^*(s)$; i.e., the expected numbers of visits to State j per unit of time in the steady-state are

$$\begin{aligned}
 M_0 &= \frac{1}{\int_0^T \overline{F}(T)dt - F(T)/\mu_1 - \overline{F}(T)/\mu_2}, \\
 M_1 &= \frac{F(T)}{\int_0^T \overline{F}(T)dt - F(T)/\mu_1 - \overline{F}(T)/\mu_2}, \\
 M_2 &= \frac{\overline{F}(T)}{\int_0^T \overline{F}(T)dt - F(T)/\mu_1 - \overline{F}(T)/\mu_2}.
 \end{aligned}$$

3 Cloud Computing System Consisting of Intelligent Nodes with Replication

This section formulates a stochastic model of a cloud computing system with n ($n = 1, 2, \dots$) intelligent nodes and a data center. It is shown that there exists an optimal number n^* to minimize the expected cost including management cost for intelligent nodes. The server in the intelligent node provides the application service when a client requests the application. In this model, we make consideration of the data for client, i.e., we consider how to make regular backups of client data from all of the intelligent nodes to a data center. When the server in the intelligent node updates the data by requesting of client at times k ($k = 1, 2, \dots$), the monitor orders n intelligent nodes to transmit all of client data to a data center, that is, the data replication executes from

all of intelligent nodes to a data center. When one of n intelligent nodes breaks down, the client service is migrated from the intelligent node to another one. We derive the expected numbers of replications and updating the client data in intelligent nodes before a failure of an intelligent node, using techniques of a Markov renewal process. In order to reduce replication and loss costs for the client data, it is shown that there exists an optimal replication interval k^* to minimize the expected cost including these costs.

3.1 Reliability Quantities

A cloud computing system consists of a monitor, a data center, and n intelligent nodes as shown in Fig. 2.

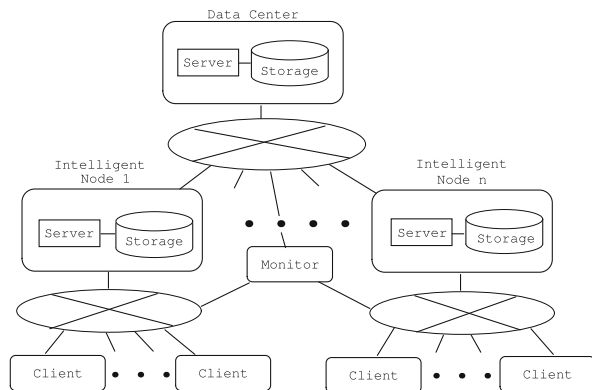
Both a data center and n intelligent nodes consist of identical server and storage, and each intelligent node can serve as backup when another intelligent node breaks down. A data center manages all of the data and application data for client. The monitor orders each intelligent node to serve the request of each client, i.e., the monitor assigns each intelligent nodes to each client. The operation of a cloud computing system is shown in Fig. 3.

The server in the intelligent node provides the application when a client requests the use of an application. The monitor orders n intelligent nodes to transmit all of client data to a data center, and furthermore, orders n intelligent nodes to receive the update information of the application from a data center, after the server in the intelligent node updates the data by requesting of client at times k . When one of n intelligent nodes breaks down by some failures such as a disaster or crash failure, the client service is migrated from the intelligent node to another one.

Then, we formulate the stochastic model as follows:

- (1) A client requests an application whose time has a general distribution $A(t)$. The monitor orders the intelligent node to serve the request of the client. The server

Fig. 2 Outline of a cloud computing system [1, 2]



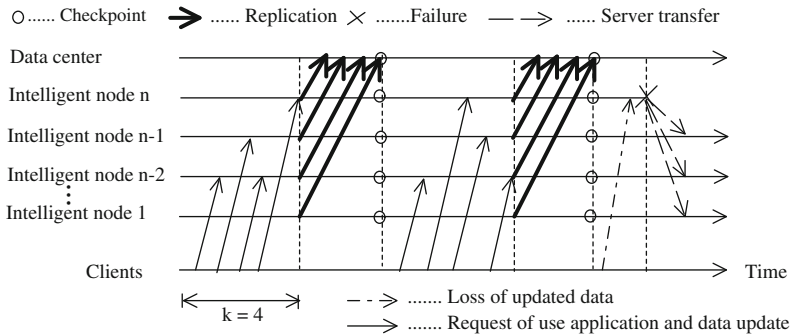
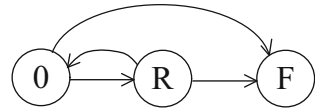


Fig. 3 Operation of a cloud computing system

in the intelligent node updates the storage database when a client requests the data update. The completion of the data update requires the time according to a general distribution $B(t)$.

- (2) The monitor orders n intelligent nodes to transmit all of client data to a data center, and furthermore, it orders n intelligent nodes to receive the update information of application from a data center, after the server in the intelligent node has updated the data by requesting of client at times k . In other words, the replication executes after the server in the intelligent node has updated the client data at times k .
 - (a) The completion of an intelligent node's replication requires the time according to a general distribution $W(t)$. That is, the replication time of n intelligent nodes has a general distribution $W^{(n)}(t)$.
 - (b) If the server in the intelligent node breaks down while the replication is executing, the monitor immediately orders migration of the client service from the intelligent node to another one.
- (3) When one of n intelligent nodes breaks down, the client service is migrated from the intelligent node to another one.
 - (a) The intelligent node breaks down according to an exponential distribution $F(t) = 1 - e^{-\lambda t}$ with finite mean $1/\lambda$ ($0 < \lambda < \infty$). The monitor and a data center do not break down.
 - (b) When the client service is migrated from the intelligent node to another one, the monitor searches for another intelligent node which can serve the request of the client.
 - (i) If the monitor finds another intelligent node which can serve the request of the client, the monitor immediately migrates the client service to the intelligent node.
 - (ii) If the monitor does not find another intelligent node, the monitor immediately prepares the intelligent node to serve the request of the client.

Fig. 4 Transition diagram between system states



Under the above assumptions, we define the following states of the cloud computing system:

- State 0: System begins to operate or restart.
- State R: Replication begins when the server in the intelligent node updates the data at times k ($k = 1, 2, \dots$).
- State F: A client service temporary stops when one of n ($n = 1, 2, \dots$) intelligent nodes breaks down, and the client service migrates from the intelligent node to another one.

The system states defined above form a Markov renewal process [11, 12], where F is an absorbing state. A transition diagram between system states is shown in Fig. 4.

The LS transforms of one-step transition probabilities $Q_{i,j}(t)$ from State i to State j in $[0, t]$ ($i = 0, R, F; j = 0, R, F$) are given by the following equations:

$$\begin{aligned}
 Q_{0,R}^*(s) &= \int_0^\infty e^{-st} \overline{F}(t) dD^{(k)}(t) = [H^*(s + \lambda)]^k, \\
 Q_{0,F}^*(s) &= \int_0^\infty e^{-st} [1 - H^{(k)}(t)] dF(t) = \frac{\lambda}{s + \lambda} \{1 - [H^*(s + \lambda)]^k\}, \\
 Q_{R,F}^*(s) &= \int_0^\infty e^{-st} [1 - W^{(n)}(t)] dF(t) = \frac{\lambda}{s + \lambda} \{1 - [W^*(s + \lambda)]^n\}, \\
 Q_{R,0}^*(s) &= \int_0^\infty e^{-st} \overline{F}(t) dW^{(n)}(t) = [W^*(s + \lambda)]^n,
 \end{aligned}$$

where $H(t) \equiv A(t) * B(t)$. Clearly, $Q_{0,R}^*(0) + Q_{0,F}^*(0) = 1$ and $Q_{R,0}^*(0) + Q_{R,F}^*(0) = 1$.

Expression for expected number of replications before State F , denoted by M_R , is derived as follows: Let $M_{0,R}(t)$ be the expected number of replications before State F in $[0, t]$. Then, we have the following renewal equation:

$$M_{0,R}(t) = Q_{0,R}(t) * Q_{R,0}(t) * [1 + M_{0,R}(t)], \tag{1}$$

and its LS transform $M_R^*(s)$ is

$$M_{0,R}^*(s) = \frac{Q_{0,R}^*(s) Q_{R,0}^*(s)}{1 - Q_{0,R}^*(s) Q_{R,0}^*(s)}. \tag{2}$$

Hence, the expected number M_R before State F is

$$M_R \equiv \lim_{s \rightarrow 0} [M_{0,R}^*(s)] = \frac{1 - X(k, n)}{X(k, n)}, \tag{3}$$

where

$$\begin{aligned} X(k, n) &= 1 - \int_0^\infty \bar{F}(t) dH^{(k)}(t) \int_0^\infty \bar{F}(t) dW^{(n)}(t), \\ &= 1 - [H^*(\lambda)]^k [W^*(\lambda)]^n, \end{aligned} \tag{4}$$

which increases with k from $X(1, n)$ to 1 and increases with n from $X(k, 1)$ to 1.

Similarly, expression for the expected number of updating the client data in intelligent nodes before State F , denoted by M_A , is derived as follows: The expected number $M_A(t)$ in $[0, t]$ is given by following equation:

$$\begin{aligned} M_A(t) &= \sum_{i=1}^k \int_0^t (i-1) H^{(i-1)}(t) * \bar{H}(t) dF(t) \\ &\quad + k \left[\int_0^t \bar{F}(t) dH^{(k)}(t) \right] * \int_0^t [1 - W^{(n)}(t)] dF(t) + Q_{0,R}(t) * Q_{R,0}(t) * M_A(t), \end{aligned} \tag{5}$$

and its LS transform is

$$M_A^*(s) = \frac{\sum_{i=1}^k \left[\int_0^\infty e^{-st} H^{(i)}(t) dF(t) - \int_0^\infty e^{-st} H^{(k)}(t) dF(t) \right] + k \left[\int_0^\infty e^{-st} \bar{F}(t) dH^{(k)}(t) \right] \left[\int_0^\infty e^{-st} [1 - W^{(n)}(t)] dF(t) \right]}{1 - \left[\int_0^\infty e^{-st} \bar{F}(t) dH^{(k)}(t) \right] \left[\int_0^\infty e^{-st} \bar{F}(t) dW^{(n)}(t) \right]}. \tag{6}$$

Hence, the expected number M_A before State F is

$$M_A \equiv \lim_{s \rightarrow 0} [M_{0,A}^*(s)] = \frac{Y(k, n)}{X(k, n)}, \tag{7}$$

where

$$\begin{aligned} Y(k, n) &\equiv \sum_{i=1}^k \left[\int_0^\infty \bar{F}(t) dH^{(i)}(t) - 1 + X(k, n) \right], \\ &= \sum_{i=1}^k \{ [H^*(\lambda)]^i - [H^*(\lambda)]^k [W^*(\lambda)]^n \}, \end{aligned} \tag{8}$$

which increases with k to $H^*(\lambda)/[1 - H^*(\lambda)]$ and increases with n to $\sum_{i=1}^k [H^*(\lambda)]^i$.

3.2 Optimal Policies

We consider two optimal policies to make regular backups of client data from n intelligent nodes to a data center. First, we propose an optimal policy to reduce the loss costs for the client data in the intelligent node. That is, when one of n intelligent nodes breaks down, the client service is migrated from one intelligent node to another one. Then, the backup data is transmitted from a data center to the intelligent node and the interrupted service is restarted. Therefore, the client data before backup is lost. In Sect. 3.2.1, to reduce the loss costs for the client data, we calculate the expected cost before State F and derive an optimal replication interval k^* to minimize it. Next, we propose an optimal policy to reduce the waste of managing the intelligent nodes. In Sect. 3.2.2, to reduce the management costs for the intelligent node, we derive an optimal number n^* to minimize the expected cost. Furthermore, we compute numerically both optimal k^* and n^* to minimize the expected cost.

3.2.1 Optimal Replication Interval k^*

We propose an optimal policy to reduce the waste of costs for replications and the loss costs for the client data in the intelligent node. That is, we calculate the expected cost before State F and derive an optimal interval k^* to minimize it. Let c_A be the loss cost for an updated data, c_R be the cost for a replication and c_M be the cost for managing an intelligent node. We give the expected cost $C_1(k, n)$ as follows:

$$\begin{aligned} C_1(k, n) &\equiv c_R M_R + c_A M_A + c_M n, \\ &= \frac{c_R [1 - X(k, n)] + c_A Y(k, n)}{X(k, n)} + c_M n. \end{aligned} \quad (9)$$

We seek an optimal replication interval k^* ($1 \leq k^* \leq \infty$) to minimize $C_1(k, n)$ for given n ($n \geq 1$). From the inequality $C_1(k+1, n) - C_1(k, n) \geq 0$,

$$\frac{Y(k+1, n) - Y(k, n)}{X(k+1, n) - X(k, n)} X(k, n) - Y(k, n) \geq \frac{c_R}{c_A} \quad (k = 1, 2, \dots). \quad (10)$$

Denoting the left-hand side of (10) by $L_n(k)$,

$$L_n(k) - L_n(k-1) = \left[\frac{Y(k+1, n) - Y(k, n)}{X(k+1, n) - X(k, n)} - \frac{Y(k, n) - Y(k-1, n)}{X(k, n) - X(k-1, n)} \right] X(k, n).$$

If $[Y(k+1, n) - Y(k, n)]/[X(k+1, n) - X(k, n)]$ increases strictly with k and $L_n(\infty) > c_R/c_A$, then there exists a finite and unique minimum k^* which satisfies (10).

In this case,

$$\begin{aligned} \frac{Y(k + 1, n) - Y(k, n)}{X(k + 1, n) - X(k, n)} &= \frac{k[1 - H^*(\lambda)]W^*(\lambda)^n + H^*(\lambda)[1 - W^*(\lambda)^n]}{[1 - H^*(\lambda)]W^*(\lambda)^n} \\ &= \frac{H^*(\lambda)}{1 - H^*(\lambda)} \frac{1 - W^*(\lambda)^n}{W^*(\lambda)^n} + k, \end{aligned}$$

which increases strictly with k from $\frac{[1 - H^*(\lambda)]W^*(\lambda)^n + H^*(\lambda)[1 - W^*(\lambda)^n]}{[1 - H^*(\lambda)]W^*(\lambda)^n} > 1$ to ∞ . Thus, there exists finite and unique minimum $k^*(1 \leq k^* < \infty)$ which satisfies (10). If $\frac{[1 - H^*(\lambda)]W^*(\lambda)^n + H^*(\lambda)[1 - W^*(\lambda)^n]}{[1 - H^*(\lambda)]W^*(\lambda)^n} \geq c_R/c_A$ then $k^* = 1$. Furthermore, if $c_R/c_A \leq 1$ then $k^* = 1$.

Example 1 We compute numerically an optimal interval k^* to minimize $C_1(k, n)$ in (9) when $A(t) \equiv 1 - e^{-\alpha t}$, $B(t) \equiv 1 - e^{-\beta t}$ and $W(t) \equiv 1 - e^{-wt}$, i.e.,

$$H^*(\lambda) = \frac{\alpha\beta}{(\lambda + \alpha)(\lambda + \beta)}, \quad W^*(\lambda) = \frac{w}{\lambda + w}.$$

Suppose that the mean time $1/\beta$ is required for the completion of the data update. It is assumed that the number of intelligent nodes is $n = 2-4$, the mean interval of requests of application is $(1/\alpha)/(1/\beta) = 2-8$, the mean interval in which an intelligent node is down is $(1/\lambda)/(1/\beta) = 1000, 5000$, the mean time required for the replication is $(1/w)/(1/\beta) = 10-30$. Further, we introduce the following costs: The loss cost rate of the cost for updated data to the cost for replication is $c_R/c_A = 5, 10$. Table 1 presents the optimal interval k^* to minimize the expected cost $C_1(k, n)/c_A$.

For example, when $c_R/c_A = 5, n = 2, \beta/\alpha = 2, \beta/w = 10$ and $\beta/\lambda = 1000$, the optimal replication interval is $k^* = 52$. This indicates that k^* increase with β/λ and c_R/c_A . On the other hand, k^* decrease with β/w and roughly decrease with β/α . This shows that we should execute the replication at short intervals when the time required for the use of application is large. Optimal k^* decrease with n , i.e., we should execute the replication at short intervals when the number of intelligent nodes is large.

Next, Fig. 5 draws the expected cost $C_1(k^*, n)/c_A$ and k^* for n when $c_R/c_A = 5, c_M/c_A = 2, \beta/\lambda = 1000, \beta/w = 30$ and $\beta/\alpha = 2$. From this figure, k^* decrease rapidly with n and asymptotically converge to 1, and $C_1(k^*, n)/c_A$ decrease notably with n and converges to $C_1(k^*, n^*)/c_A$ when $n^* = 9$ and $k^* = 1$. It would be enough for the server system to be composed of nine intelligent nodes.

3.2.2 Optimal Number n^*

We propose an optimal policy to reduce the waste of costs for replications and managing the intelligent nodes. That is, we calculate the expected cost and derive an optimal number $n^*(1 \leq n^* \leq \infty)$ to minimize $C_1(k, n)$ for given $k(k \geq 1)$. From the inequality $C_1(k, n + 1) - C_1(k, n) \geq 0$,

Table 1 Optimal interval k^* to minimize $C_1(k, n)/c_A$

n	c_R/c_A	β/w	β/λ					
			1000			5000		
			β/α					
			2	4	8	2	4	8
2	5	10	52	42	33	124	98	74
		20	45	38	30	117	93	72
		30	36	33	28	109	89	69
	10	10	78	63	49	179	141	107
		20	71	58	46	172	137	104
		30	63	54	44	165	132	102
3	5	10	49	40	32	120	96	73
		20	36	33	28	109	89	69
		30	21	25	24	97	82	66
	10	10	75	61	47	176	139	106
		20	63	54	44	165	132	102
		30	50	46	40	154	126	98
4	5	10	45	38	30	117	93	72
		20	26	28	25	101	84	67
		30	3	16	19	85	75	62
	10	10	71	58	46	172	137	104
		20	54	49	41	157	128	100
		30	35	38	36	142	119	95

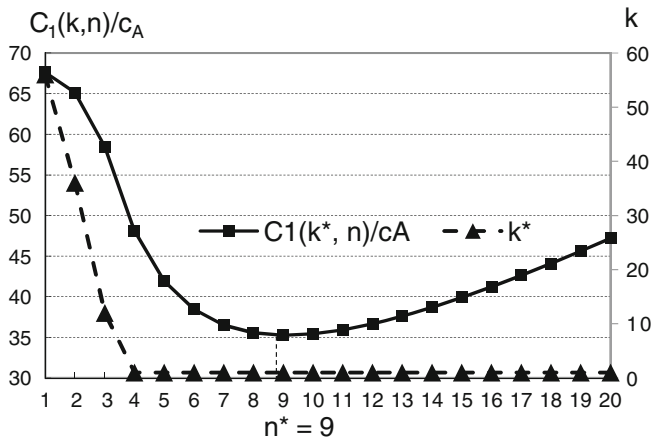


Fig. 5 Optimal cost $C_1(k^*, n)/c_A$ and optimal interval k^* for n when $c_R/c_A = 5, c_M/c_A = 2, \beta/\lambda = 1000, \beta/w = 30$ and $\beta/\alpha = 2$

$$\frac{Y(k, n + 1) - Y(k, n)}{X(k, n + 1) - X(k, n)} X(k, n) - Y(k, n) + \frac{c_M}{c_A} \frac{X(k, n + 1)X(k, n)}{X(k, n + 1) - X(k, n)} \geq \frac{c_R}{c_A} \tag{11}$$

Denoting the left-hand side of (11) by $L_k(n)$,

$$L_k(n) - L_k(n - 1) = \left\{ \left[\frac{Y(k, n + 1) - Y(k, n)}{X(k, n + 1) - X(k, n)} - \frac{Y(k, n) - Y(k, n - 1)}{X(k, n) - X(k, n - 1)} \right] + \frac{c_M}{c_A} \left[\frac{X(k, n + 1)X(k, n)}{X(k, n + 1) - X(k, n)} - \frac{X(k, n)X(k, n - 1)}{X(k, n) - X(k, n - 1)} \right] \right\} X(k, n).$$

If $[Y(k, n + 1) - Y(k, n)]/[X(k, n + 1) - X(k, n)]$ and $X(k, n + 1)X(k, n)/[X(k, n + 1) - X(k, n)]$ increases strictly with n and $L_k(\infty) > c_R/c_A$, then there exists a finite and unique minimum n^* which satisfies (11). In this case,

$$\frac{Y(k, n + 1) - Y(k, n)}{X(k, n + 1) - X(k, n)} = k, \tag{12}$$

$$\frac{X(k, n + 1)X(k, n)}{X(k, n + 1) - X(k, n)} = \frac{\{1 - [H^*(\lambda)]^k [W^*(\lambda)]^{n+1}\} \{1 - [H^*(\lambda)]^k [W^*(\lambda)]^n\}}{[H^*(\lambda)]^k [1 - W^*(\lambda)] [W^*(\lambda)]^n},$$

which increases strictly with n from

$$\frac{\{1 - [H^*(\lambda)]^k [W^*(\lambda)]^2\} \{1 - [H^*(\lambda)]^k W^*(\lambda)\}}{[H^*(\lambda)]^k [1 - W^*(\lambda)] W^*(\lambda)}$$

to ∞ . Thus, there exists a finite and unique minimum n^* ($1 \leq n^* < \infty$) which satisfies (11).

Example 2 We compute numerically an optimal number n^* to minimize $C_1(k, n)$ in (9) when $A(t) \equiv 1 - e^{-\alpha t}$, $B(t) \equiv 1 - e^{-\beta t}$ and $W(t) \equiv 1 - e^{-wt}$. It is assumed that the replication interval is $k = 20, 30$ and for the mean time $1/\beta$ of $B(t)$, the mean interval of requests is $(1/\alpha)/(1/\beta) = 2-6$, the mean interval of an intelligent node being down is $(1/\lambda)/(1/\beta) = 3000, 5000$, the mean time required for the replication is $(1/w)/(1/\beta) = 20, 30$. Further, we introduce the following costs: For the loss cost rate of the cost for an updated data to the cost for replication is $c_R/c_A = 5, 10$, and the cost of managing of an intelligent node is $c_M/c_A = 2, 3$.

Table 2 presents the optimal number n^* to minimize $C_1(k, n)/c_A$. For example, when $c_R/c_A = 5, c_M/c_A = 2$ and $k = 20, \beta/\alpha = 2, \beta/w = 20, \beta/\lambda = 3000$, the optimal number is $n^* = 16$.

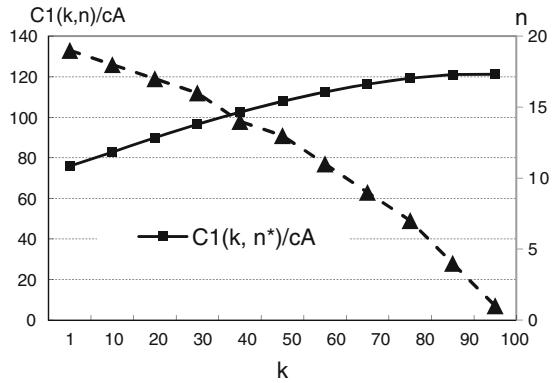
This indicates that n^* increases with β/λ and c_R/c_A . On the other hand, n^* decreases with β/w and decreases with β/α and c_M/c_A , and also, decrease with k . That is, it is only necessary to have fewer intelligent nodes when the replication interval k is large.

Next, Fig. 6 draws the optimal cost $C_1(k, n^*)/c_A$ and optimal number n^* for k when $1/\beta = 1, c_R/c_A = 5, c_M/c_A = 2, \beta/\lambda = 3000, \beta/w = 20$ and $\beta/\alpha = 2$.

Table 2 Optimal number n^* to minimize $C_1(k, n)/c_A$

k	c_R/c_A	c_M/c_A	β/w	β/λ						
				3000			5000			
				β/α						
				2	4	6	2	4	6	
20	5	2	20	16	14	11	22	19	17	
			30	14	12	10	18	17	15	
		3	20	12	10	8	17	15	13	
			30	11	9	8	14	13	12	
		10	2	20	24	22	20	32	30	28
				30	20	19	17	27	25	24
	3		20	19	17	15	26	24	21	
			30	16	15	13	21	20	19	
	30	5	2	20	14	10	7	20	16	13
				30	12	10	7	17	14	12
			3	20	11	7	4	15	12	9
				30	9	7	4	13	11	9
10			2	20	22	19	15	30	27	24
				30	19	16	14	25	23	21
		3	20	17	14	11	24	21	17	
			30	15	13	10	20	18	16	

Fig. 6 Optimal cost $C_1(k, n^*)/c_A$ and optimal number n^* for k when $1/\beta = 1$, $c_R/c_A = 5$, $c_M/c_A = 2$, $\beta/\lambda = 3000$, $\beta/w = 20$ and $\beta/\alpha = 2$



From this figure, n^* decrease rapidly with k and converges to 1, and $C_1(k, n^*)$ increases remarkably with k and tends to $C_1(k^*, n^*)$ when $k^* = 1$ and $n^* = 19$. This indicates that we should execute synchronous replication when the number of intelligent node is $n^* = 19$.

Next, we derive both optimal k^* and n^* to minimize $C_1(k, n)/c_A$ in (9), i.e., we solve the simultaneous equations (10) and (11). It is easily computed that first, we compute k_0 for given n_0 from (10), compute n_1 for given k_0 from (11) and repeat such

Table 3 Optimal interval k^* and number n^* to minimize $C_1(k, n)/c_A$

β/λ	c_R/c_A	c_M/c_A	β/w	β/α					
				2		4		6	
				k^*	n^*	k^*	n^*	k^*	n^*
1000	5	2	10	56	1	44	1	38	1
			20	1	11	42	1	37	1
			30	1	9	1	9	1	9
		3	10	56	1	44	1	38	1
			20	1	9	42	1	37	1
			30	1	7	1	7	35	1
	10	2	10	82	1	65	1	56	1
			20	1	16	63	1	54	1
			30	1	13	1	13	1	13
		3	10	82	1	65	1	56	1
			20	1	13	63	1	54	1
			30	1	10	61	1	53	1
5000	5	2	10	127	1	100	1	85	1
			20	1	25	1	25	83	1
			30	1	20	1	20	1	20
		3	10	127	1	100	1	85	1
			20	1	20	98	1	83	1
			30	1	17	1	17	82	1
	10	2	10	183	1	143	1	122	1
			20	1	35	141	1	120	1
			30	1	29	1	29	1	29
		3	10	183	1	143	1	122	1
			20	1	29	141	1	120	1
			30	1	23	139	1	119	1

computing procedures until $k_i = k_{i+1} = k^*$ and $n_i = n_{i+1} = n^*$. Table 3 presents optimal k^* and n^* to minimize $C_1(k, n)$ when $A(t) \equiv 1 - e^{-\alpha t}$, $B(t) \equiv 1 - e^{-\beta t}$ and $W(t) \equiv 1 - e^{-wt}$.

This table indicates that both n^* and k^* increase with β/λ and c_R/c_A . On the other hand, when β/w is small, $n^* = 1$, that is, we should execute asynchronous replication and should not perform distributed management of client data by multiple intelligent nodes. Conversely, when β/w is large, $k^* = 1$. That is, we should execute synchronous replication and manage client data with multiple intelligent nodes. When β/λ , β/α and c_M/c_A are large, n^* do not depend on β/w , and become constant $n^* = 1$. That is, we should execute asynchronous replication and should not perform distributed management of client data with multiple intelligent nodes. Comparing with Table 1, optimal k^* are close to the value of k^* in Table 1 when β/α is large and β/w is small. On the other hand, by comparison with Table 2, optimal n^* are close to the value of n^* in Table 2 when β/α is small and β/w is large.

4 Cloud Computing System with Intelligent Nodes for Changing at Failure

This section considers an extended stochastic model of a cloud system with n intelligent nodes, in which the service continues to operate until all of intelligent nodes break down. That is, when one of n intelligent nodes breaks down, the client service is migrated from the intelligent node to another one, and the cloud system continues to provide the application service with alternative intelligent nodes until n intelligent nodes break down. We assume the same server system model as Sect. 3, and derive the expected numbers of replications and updating the client data in intelligent nodes before failures of n intelligent nodes. Further, we derive the expected cost and discuss an optimal replication interval k^* to minimize it for given n .

4.1 Reliability Quantities

A cloud computing system consists of a monitor, a data center, and n intelligent nodes as shown in Fig. 2 [1, 14], its operation is shown in Fig. 7.

The server in the intelligent node provides the application when a client requests the use of an application, and updates the data by requesting the client. The replication executes after the server in the intelligent node has updated the client data at times k . Whenever one of intelligent nodes breaks down, the client service is migrated from the intelligent node to another one repeatedly until n intelligent nodes break down. Then, we formulate the stochastic model in the same manner as Sect. 3. The migration of the service requires the time according to a general distribution $G(t)$, and define the following states of a cloud computing system:

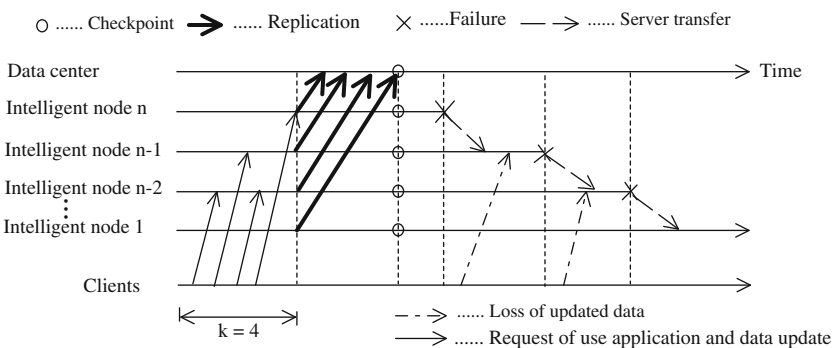
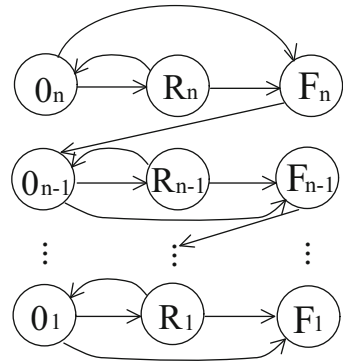


Fig. 7 Operation of a cloud computing system

Fig. 8 Transition diagram between system states



- State 0_i : System begins to operate or restart by normal i ($i = 1, 2, 3, \dots, n$) intelligent nodes.
- State R_i : When the server in the intelligent node updates the data at times k ($k = 1, 2, \dots$), the replication begins.
- State F_i : One of i ($i = 2, 3, \dots, n$) intelligent nodes breaks down.
- State F_1 : All of n intelligent nodes break down.

The system states defined above form a Markov renewal process [10, 12], where F_1 is an absorbing state. A transition diagram between system states is shown in Fig. 8.

The LS transforms of transition probabilities $Q_{\ell,j}(t)$ ($\ell = 0_i, R_i, F_i; j = 0_i, R_i, F_i$) are given by the following equations:

$$\begin{aligned}
 Q_{0_i,R_i}^*(s) &= \int_0^\infty e^{-st} \overline{F}(t) dH^{(k)}(t) = [H^*(s + \lambda)]^k, \\
 Q_{0_i,F_i}^*(s) &= \int_0^\infty e^{-st} [1 - H^{(k)}(t)] dF(t) = \frac{\lambda}{s + \lambda} \{1 - [H^*(s + \lambda)]^k\}, \\
 Q_{R_i,F_i}^*(s) &= \int_0^\infty e^{-st} [1 - W^{(i)}(t)] dF(t) = \frac{\lambda}{s + \lambda} \{1 - [W^*(s + \lambda)]^i\}, \\
 Q_{R_i,0_i}^*(s) &= \int_0^\infty e^{-st} \overline{F}(t) dW^{(i)}(t) = [W^*(s + \lambda)]^i.
 \end{aligned}$$

Clearly, $Q_{0_i,R_i}^*(0) + Q_{0_i,F_i}^*(0) = 1$ and $Q_{R_i,0_i}^*(0) + Q_{R_i,F_i}^*(0) = 1$. The expected number $M_R(t)$ of replications before State F_1 in $[0, t]$ is given by the following equation:

$$\begin{aligned}
 M_R(t) &= M_{0_n,R_n}(t) + H_{0_n,F_n}(t) * G(t) * M_{0_{n-1},R_{n-1}}(t) \\
 &+ H_{0_n,F_{n-1}}(t) * G(t) * M_{0_{n-2},R_{n-2}}(t) + H_{0_n,F_{n-2}}(t) * G(t) * M_{0_{n-3},R_{n-3}}(t) \\
 &+ \dots + H_{0_n,F_2}(t) * G(t) * M_{0_1,R_1}(t),
 \end{aligned} \tag{12}$$

where

$$\begin{aligned}
 M_{0_i, R_i}(t) &\equiv Q_{0_i, R_i}(t) * Q_{R_i, 0_i}(t) * [1 + M_{0_i, R_i}(t)] \quad (i = 1, 2, \dots, n), \\
 H_{0_n, F_j}(t) &\equiv H_{0_n, F_n}(t) * G(t) * H_{0_{n-1}, F_{n-1}}(t) * G(t) * H_{0_{n-2}, F_{n-2}}(t) * G(t) \\
 &\quad * H_{0_{n-3}, F_{n-3}}(t) * G(t) * \dots * H_{0_j, F_j}(t), \quad (j = 2, 3, \dots, n-1), \\
 H_{0_i, F_i}(t) &\equiv Q_{0_i, F_i}(t) + Q_{0_i, R_i}(t) * Q_{R_i, F_i}(t) + Q_{0_i, R_i}(t) * Q_{R_i, 0_i}(t) * H_{0_i, F_i}(t), \\
 &\quad (i = 1, 2, \dots, n).
 \end{aligned}$$

and its LS transform is

$$\begin{aligned}
 M_R^*(s) &= \frac{Q_{0_n, R_n}^*(s) Q_{R_n, 0_n}^*(s)}{1 - Q_{0_n, R_n}^*(s) Q_{R_n, 0_n}^*(s)} + G^*(s)^{n-1} \\
 &\times \sum_{i=2}^n \frac{Q_{0_{i-1}, R_{i-1}}^*(s) Q_{R_{i-1}, 0_{i-1}}^*(s)}{1 - Q_{0_{i-1}, R_{i-1}}^*(s) Q_{R_{i-1}, 0_{i-1}}^*(s)} \left[\prod_{j=i}^n \frac{Q_{0_j, F_j}^*(s) + Q_{0_j, R_j}^*(s) Q_{R_j, F_j}^*(s)}{1 - Q_{0_j, R_j}^*(s) Q_{R_j, 0_j}^*(s)} \right].
 \end{aligned}$$

Hence, the expected number M_R before State F_1 is

$$M_R \equiv \lim_{s \rightarrow 0} [M_R^*(s)] = \sum_{i=1}^n \frac{1 - X(k, i)}{X(k, i)}, \tag{13}$$

where $X(k, i)$ is given in (4).

Similarly, we derive the expected number M_A of updating the client data in intelligent nodes before State F_1 . The expected number $M_A(t)$ in $[0, t]$ is given by following equation:

$$\begin{aligned}
 M_A(t) &= M_{0_n, A_n}(t) + H_{0_n, F_n}(t) * G(t) * M_{0_{n-1}, A_{n-1}}(t) \\
 &\quad + H_{0_n, F_{n-1}}(t) * G(t) * M_{0_{n-2}, A_{n-2}}(t) \\
 &\quad + H_{0_n, F_{n-2}}(t) * G(t) * M_{0_{n-3}, A_{n-3}}(t) + \dots + H_{0_n, F_2}(t) * G(t) * M_{0_1, A_1}(t),
 \end{aligned} \tag{14}$$

where

$$\begin{aligned}
 M_{0_i, A_i}(t) &= \sum_{i=1}^k \int_0^\infty (i-1) H^{(i-1)}(t) * \bar{H}(t) dF(t) \\
 &\quad + k \left[\int_0^\infty \bar{F}(t) dH^{(k)}(t) \right] * \int_0^\infty [1 - W^{(i)}(t)] dF(t) + [Q_{0_i, R_i}(t) * Q_{R_i, 0_i}(t)] * M_{0_i, A_i}(t),
 \end{aligned}$$

and its LS transform is

$$M_A^*(s) = M_{0_n, A_n}^*(s) + G^*(s)^{n-1} \sum_{i=1}^{n-1} M_{0_i, A_i}^*(s) \left[\prod_{j=i+1}^n \frac{Q_{0_j, F_j}^*(s) + Q_{0_j, R_j}^*(s) Q_{R_j, F_j}^*(s)}{1 - Q_{0_j, R_j}^*(s) Q_{R_j, 0_j}^*(s)} \right],$$

where

$$M_{0_i, A_i}^*(s) = \frac{\sum_{j=1}^k [\int_0^\infty e^{-st} H^{(j)}(t) dF(t) - \int_0^\infty e^{-st} H^{(k)}(t) dF(t)] + k \int_0^\infty e^{-st} \bar{F}(t) dH^{(k)}(t) \int_0^\infty e^{-st} [1 - W^{(i)}(t)] dF(t)}{1 - \int_0^\infty e^{-st} \bar{F}(t) dH^{(k)}(t) \int_0^\infty e^{-st} \bar{F}(t) dW^{(i)}(t)}.$$

Hence, the expected number M_A of updating the client data before State F_1 is

$$M_A \equiv \lim_{s \rightarrow 0} [M_A^*(s)] = \sum_{i=1}^n \frac{Y(k, i)}{X(k, i)}, \tag{15}$$

where $Y(k, i)$ is given in (8).

4.2 Optimal Policy

We propose an optimal policy to reduce the waste of costs for replications and the loss costs for updates in an intelligent node. That is, we calculate the expected cost and derive an optimal replication interval k^* to minimize it for given n . Let c_A be the loss cost for an updated data and c_R be the cost for a replication. We give the expected cost $C_2(k, n)$ as follows:

$$C_2(k, n) \equiv c_R M_R + c_A M_A. \tag{16}$$

We seek an optimal replication interval $k^* (1 \leq k^* \leq \infty)$ to minimize $C_2(k, n)$ for given $n (n \geq 1)$. From the inequality $C_2(k + 1, n) - C_2(k, n) \geq 0$,

$$\sum_{i=1}^n L_1(k, i) \left[L_2(k, i) - \frac{c_R}{c_A} \right] \geq 0 \quad (k = 1, 2, \dots), \tag{17}$$

where

$$L_1(k, i) \equiv \frac{1}{X(k, i)} - \frac{1}{X(k + 1, i)},$$

$$L_2(k, i) \equiv \frac{Y(k + 1, i) - Y(k, i)}{X(k + 1, i) - X(k, i)} X(k, i) - Y(k, i).$$

Furthermore,

$$L_2(k, n) - L_2(k - 1, n) = \left[\frac{Y(k + 1, i) - Y(k, i)}{X(k + 1, i) - X(k, i)} - \frac{Y(k, i) - Y(k - 1, i)}{X(k, i) - X(k - 1, i)} \right] X(k, i).$$

If $1/X(k, i) - 1/X(k + 1, i)$ and $[Y(k + 1, i) - Y(k, i)]/[X(k + 1, i) - X(k, i)]$ increases strictly with k and $\sum_{i=1}^n L_1(\infty, i)[L_2(\infty, i) - c_R/c_A] > 0$, then there exists a finite and unique minimum $k^*(1 \leq k^* < \infty)$ which satisfies (17).

In particular, from (4) and (8),

$$\begin{aligned} \frac{1}{X(k, i)} - \frac{1}{X(k + 1, i)} &= \frac{[1 - H^*(\lambda)]W^*(\lambda)^i}{\left[\frac{1}{H^*(\lambda)^k} - W^*(\lambda)^i \right] \left[\frac{1}{H^*(\lambda)^k} - H^*(\lambda)W^*(\lambda)^i \right]} > 0, \\ \frac{Y(k + 1, i) - Y(k, i)}{X(k + 1, i) - X(k, i)} &= \frac{H^*(\lambda)}{1 - H^*(\lambda)} \frac{1 - W^*(\lambda)^i}{W^*(\lambda)^i} + k > 0, \end{aligned}$$

and $\lim_{k \rightarrow \infty} L(k, n) = \infty$. Therefore, there exists a finite and unique minimum $k^*(1 \leq k^* < \infty)$ which satisfies

$$\begin{aligned} &\sum_{i=1}^n \frac{H^*(\lambda)^k [1 - H^*(\lambda)]W^*(\lambda)^i}{[1 - H^*(\lambda)^k W^*(\lambda)^i][1 - H^*(\lambda)^{k+1} W^*(\lambda)^i]} \\ &\times \left\{ \left[\frac{H^*(\lambda)}{1 - H^*(\lambda)} \frac{1 - W^*(\lambda)^i}{W^*(\lambda)^i} \right] [1 - H^*(\lambda)^k W^*(\lambda)^i] + \sum_{j=1}^k [1 - H^*(\lambda)^j] - \frac{c_R}{c_A} \right\} \geq 0, \end{aligned}$$

whose left-hand side increases strictly with n to ∞ . So that, optimal k^* decreases with n to 1.

Example 3 We compute numerically an optimal interval k^* to minimize $C_2(k, n)$ in (16). Suppose that $A(t) \equiv 1 - e^{-\alpha t}$, $B(t) \equiv 1 - e^{-\beta t}$ and $W(t) \equiv 1 - e^{-wt}$. It is assumed that the number of intelligent nodes is $n = 2-4$, the mean interval of requests of application is $(1/\alpha)/(1/\beta) = 2-8$, the mean interval of an intelligent node breaking down is $(1/\lambda)/(1/\beta) = 1000, 5000$, the mean time required for the replication is $(1/w)/(1/\beta) = 10-30$. Further, we introduce the following costs: The loss cost rate of the cost for updated data to the cost for replication is $c_R/c_A = 5, 10$. Table 4 presents the optimal interval k^* to minimize the expected cost $C_2(k, n)/c_A$.

For example, when $c_R/c_A = 5, n = 4, \beta/\alpha = 4, \beta/w = 10$ and $\beta/\lambda = 1000$, the optimal interval is $k^* = 41$. This shows that k^* increases with $1/\lambda$ and c_R/c_A . On the other hand, k^* decreases with $1/\alpha$ and $1/w$. This shows that we should execute the replication at short intervals when the time required for replication is large, and k^* decreases with n . That is, we should execute the replication at short intervals when the number of intelligent nodes is large. By comparison with Table 1, optimal k^* is smaller than then k^* in Table 1. That is, when we assume n intelligent nodes for changing at failure, we should execute the replication at short intervals.

Table 4 Optimal interval k^* to minimize $C_2(k, n)/c_A$

n	c_R/c_A	β/w	β/λ					
			1000			5000		
			β/α					
			2	4	8	2	4	8
2	5	10	54	43	33	126	99	75
		20	49	40	32	121	96	73
		30	43	37	30	115	92	71
	10	10	80	64	49	181	142	107
		20	75	61	47	176	139	106
		30	69	57	46	171	136	104
3	5	10	53	42	33	124	98	74
		20	45	38	31	117	94	72
		30	38	34	28	110	89	69
	10	10	78	63	49	179	141	107
		20	71	59	46	172	137	104
		30	64	54	44	165	133	102
4	5	10	51	41	32	122	97	73
		20	42	36	29	114	91	71
		30	33	31	26	105	86	68
	10	10	77	62	48	178	140	106
		20	68	56	45	169	135	103
		30	59	51	42	160	129	100

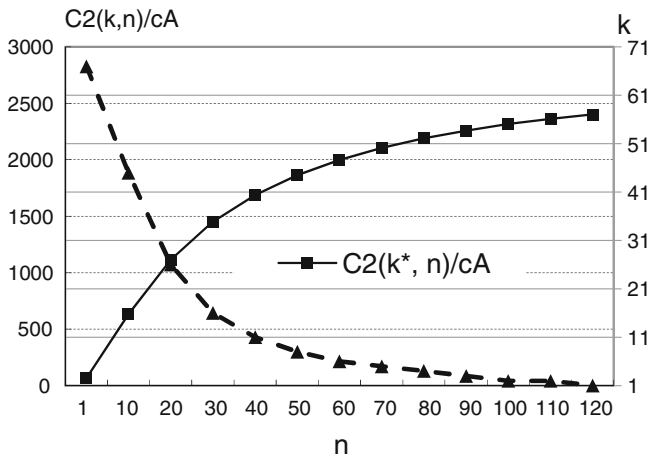


Fig. 9 Optimal cost $C_2(k^*, n)/c_A$ and optimal interval k^* for n when $1/\beta = 1$, $c_R/c_A = 5$, $\beta/\lambda = 1000$, $\beta/w = 10$ and $\beta/\alpha = 2$

Figure 9 shows the optimal cost $C_2(k^*, n)/c_A$ and optimal interval k^* for n when $1/\beta = 1$, $c_R/c_A = 5$, $1/\lambda = 1000$, $1/w = 10$ and $1/\alpha = 2$. From this figure, k^* decrease rapidly with n and tends to 1. $C_2(k, n^*)/c_A$ increases visibly with n and tends to $C_2(k^*, n^*)/c_A$ when $k^* = 67$ and $n^* = 1$. That is, we should execute asynchronous replication when the number of intelligent node is $n^* = 1$.

5 Conclusions

This chapter has analytically studied two reliability models of a cloud computing system consisting of n intelligent nodes with replication in Sect. 3 and for changing at failure in Sect. 4. Furthermore, we have derived the reliability measures by using techniques of Markov renewal processes, and have discussed the optimal policies to minimize the expected cost.

From numerical examples, we have shown that the optimal replication interval decreases as the number of intelligent nodes increases and approaches one when the number of intelligent nodes become a constant value. That is, we should execute replication at short intervals when the number of intelligent nodes is large and should apply synchronous replication when the number of intelligent nodes is a constant value. Further, the optimal number of intelligent nodes decreases as the replication interval increases. That is, it is only necessary to have fewer intelligent nodes when the interval is large.

References

1. Okuno M, Ito D, Miyamoto H, Aoki H, Tsushima Y, Yazaki T (2010) A study on distributed information and communication processing architecture for next generation cloud system. Tech Rep Inst Electron Inf Commun Eng 109(448):241–246, NS2009-204
2. Okuno M, Tsutsumi S, Yazaki T (2011) A study of high available distributed network processing technique for next generation cloud system. Tech Rep Inst Electron Inf Commun Eng 111(8):25–30, NS2011-5
3. Yamada S, Marukawa J, Ishii D, Okamoto S, Yamanaka N (2010) A study of parallel transmission technique with GMPLS in intelligent cloud network. Tech Rep Inst Electron Inf Commun Eng 109(455):51–56, PN2009-95
4. Yamato J, Kan M, Kikuchi Y (2006) Storage based data protection for disaster recovery. J Inst Electron Inf Commun Eng 89(9):801–805
5. VERITAS Software Corporation. In: VERITAS Volume Replication for UNIX Datasheet. http://eval.veritas.com/mktginfo/products/Datasheets/High_Availability/vvr_datasheet_unix.pdf
6. Kimura M, Imaizumi M, Nakagawa T (2011) Reliability characteristics of a server system with asynchronous and synchronous replications. Int J Qual Technol Quant Manag 8(1):45–55
7. Weiss A (2007) Computing in the clouds. netWorker 11:16–25
8. Armbrust M et al (2009) Above the clouds: a Berkeley view of cloud computing. Technical report UCV/EECS-2009-28, University of California at Berkeley
9. Tsutsumi S, Okuno M (2011) A messaging scheme for wide-area distributed applications. Tech Rep Inst Electron Inf Commun Eng 110(448):533–538, NS2010-258

10. Trivedi K (2001) Probability and statistics with reliability, queuing and computer science applications, 2nd edn. Wiley
11. Osaki S (1992) Applied stochastic system modeling. Springer, Berlin
12. Nakagawa T (2011) Stochastic processes with applications to reliability theory. Springer, London
13. Nakagawa T (2005) Maintenance theory of reliability. Springer, London
14. Kimura M, Imaizumi M, Nakagawa T (2014) Reliability modeling of distributed information processing for cloud computing. In: Proceedings of 20th ISSAT international conference on reliability and quality in design, pp 183–187

Service Reliability Enhancement in Cloud by Checkpointing and Replication

Subrota K. Mondal, Fumio Machida and Jogesh K. Muppala

Abstract Virtual machines (VMs) are used in cloud computing systems to handle user requests for service. A user's request cannot be completed if the VM fails. Replication mechanisms can be used to mitigate the impact of VM failures. In this chapter, we are primarily interested in characterizing the failure–recovery behavior of a VM in the cloud under different replication schemes. We use a service-oriented dependability metric called Defects Per Million (DPM), defined as the number of user requests dropped out of a million due to VM failures. We present an analytical modeling approach for computing the DPM metric in different replication schemes on the basis of the checkpointing method. The effectiveness of replication schemes are demonstrated through experimental results. To verify the validity of the proposed analytical modeling approach, we extend the widely used cloud simulator CloudSim and compare the simulation results with analytical solutions.

1 Introduction

Cloud computing is a realization of the endeavor to provide computing as a utility. Users submit their work requests to the *cloud* to be processed and the results are delivered. Typically, users' demands are met by deploying Virtual Machines (VMs) on Physical Machines (PMs) [1]. The proclivity of resources to failures, especially while executing long running computations, renders them vulnerable. Consequently,

S.K. Mondal (✉) · J.K. Muppala
The Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong
e-mail: skmondal@cse.ust.hk

J.K. Muppala
e-mail: muppala@cse.ust.hk

F. Machida
Laboratory for Analysis of System Dependability, Kawasaki, Japan
e-mail: mfumio@ieee.org

a user's request may not be completed successfully. Thus, our goal is to ensure dependability of the cloud in terms of job requests.

Traditional dependability metrics defined from a *system-oriented* perspective [2] like availability, reliability, continuity, and maintainability may not adequately capture the dependability experience from a user's perspective. Since most of the components in a modern cloud computing system are reliable, it is better to focus on the much smaller number of unreliable service events or service defects [3]. These service defects are conveniently normalized as the number of customer demands or user requests (in this chapter, a *request* or *demand* is referred to as a *job*) not served or dropped or lost, per million attempts—referred to as defects per million (DPM) [3–5]. DPM is considered from the perspective of a user's request (i) not being accepted or (ii) not being handled properly within the specified time or deadline. Note that the former is primarily governed by the availability of resources to serve the request, while the latter is governed by whether the allocated resources are available for the entire duration of the job execution until its completion.

In this chapter, we concentrate on the second perspective. We present a new formulation for computing DPM with respect to the violation of the deadline for job completion time by computing the number of requests that are not completed within the deadline. Our DPM analysis is based on the VM provisioning model for the cloud [1], where a VM is deployed on a PM upon arrival of user request. Any failures of VMs or PMs result in the interruption/preemption of job execution. Our model takes into account replication (active as well as passive replication—hot, warm, and cold) [5] of job execution as a viable means to provide sufficient deterrence against failure to complete a user's request. Warm and hot replication schemes can employ checkpointing [6, 7] in order to resume job execution and ensure/maintain/provide service reliability and performance at an optimum level. On the other hand, cold replication schemes do not provide the support for resuming, i.e., no checkpointing is adopted. In this chapter, we use the terminology checkpointing method to formulate the problem whether a system adopts checkpointing or not during the job execution (The formulation of job completion time considering with and without checkpointing is presented in [6], which we refer to as the checkpointing method in this chapter). In order to compare the effectiveness of replication schemes, we also consider the *drop policy* as a baseline case to be compared with. In the drop policy, a job is considered to have been dropped if the system fails before the job completes its execution.

In this chapter, we use the checkpointing method to model the cloud service execution under different replication schemes. The framework for checkpointing method [6] is used to get the job completion time distribution. From this distribution, we can compute the DPM where we count the expected number of jobs that violate the deadline under the given distribution. The different replication schemes: cold, warm, and hot and their corresponding DPM values are compared with each other in our numerical experiments.

The rest of this chapter is organized as follows: we discuss the related work in Sect. 2. In Sect. 3.1, we describe the system model, which depicts the life cycle of a job. The basic concept of replication in a cloud computing system is presented

in Sect. 3.2. A brief discussion on job completion time formulation is presented in Sect. 4. In Sect. 5, we show the models and closed form expressions for DPM computation by employing replication of job execution. Experimental results are shown in Sect. 6. Conclusion and future work are given in the final section.

2 Related Work

Cisco's report [8] presents a simple approach for calculating DPM, multiplying the unavailability by a constant of proportionality. To the best of our knowledge, this is the first attempt at relating the DPM metric to system unavailability.

Johnson et al. [9] state that DPM is a reliability metric that only counts customer demands not served and they referred to DPM as the average number of blocked requests (rejected jobs before service establishment) and cutoff requests (dropped jobs due to outages after service establishment) per million attempted requests.

Trivedi et al. [4] developed a novel method for computing DPM metric which takes into consideration system availability and impact of service application as well as the transient behavior in the failure–repair process. This method takes into account software/hardware failures, different stages of recovery, different phases of job flow, retry attempts, and the interactions between job flow and failure/recovery behavior. This approach is used in [10] for computing DPM in a cloud architecture. *Structure-state process* [11] under different replication schemes is used in [10] for the modeling of cloud service execution. Further, DPM formulation in [10] is shown considering only the failures of virtual resources, but no failure of physical resources is taken into account. This paper extends the work by taking into account these features in the DPM formulation.

In this work, we employ the ideas in [10] for computing DPM using checkpointing method under different replication schemes and show how we can use checkpointing optimally for enhancing service reliability and performance at an optimum level. Checkpointing [6] is one of the approaches empirically/practically used to enable resuming the preempted job. In addition, preemptive resuming of structure-state process theoretically assumes perfect checkpointing (resumes from the failed state and does not take into account the downtime overhead), but usually warm and hot replication follows the underlying principles of *equidistant checkpointing* or *periodical checkpointing* and downtime overhead is taken into account. In this chapter, we take into account these ideas and show the modeling of job execution under different replication schemes using the checkpointing method [6]. On the other hand, if a job has to be restarted/resumed upon failure occurrence, the completion time of the job is prolonged accordingly. If the job completion time exceeds its deadline, then the user refuses the service and job is considered as lost or dropped. We incorporate this idea for computing DPM by modeling checkpointing method under different replication schemes. Further, note that we show the analysis of DPM formulation simply using checkpointing for a cloud service implementation in [12] without considering any replication of execution. We propose to use replication so that if one fails, another

one can takeover shortly. In particular, adoption of checkpointing without replication lacks the contribution in service reliability enhancement if it takes long time for failure–recovery of a system. Further, if we do not use replication, we may loose the saved execution states due to the failure of the standalone machine. Thus, we analyze the service reliability having checkpointing and replication altogether.

In addition, the analysis of the Google Cluster Dataset [13] indicates that single-task jobs occupy 64% of the total jobs. In this chapter, we develop an analytical model with respect to a single-task job and leave the modeling of the other kinds of jobs such as batch tasks, sequential tasks, and mix-mode tasks [13] for future work.

3 Job Execution in the Cloud

In this section, we first present the system model which discusses about resource provisioning and implementing steps of cloud services. Second, we introduce VM replication mechanism and describe how the system can enhance service reliability of job execution by using replication.

3.1 System Model

In cloud computing systems, especially in IaaS cloud, when a request is processed, a pre-built image is used to deploy one or more Virtual Machine instances or a pre-deployed VM may be customized and made available to the requester. VMs are deployed on Physical Machines (PMs) each of which may be shared by multiple VMs. The deployed VMs are provisioned with request specific CPU, RAM, and disk capacity [1]. We assume that all requests are homogeneous (statistically equivalent) and each request is for one VM with request specific CPU core, RAM, and disk capacity. Figure 1 shows the life-cycle of a request as it moves through the system [1]. User requests (i.e., jobs) are submitted to a global resource provisioning decision engine (RPDE) that processes requests on a first-come, first-served (FCFS) basis. The request at the head of the queue is provisioned on a PM if there is any capacity to run a VM on it. If no PM is available, the request is rejected. When a running job

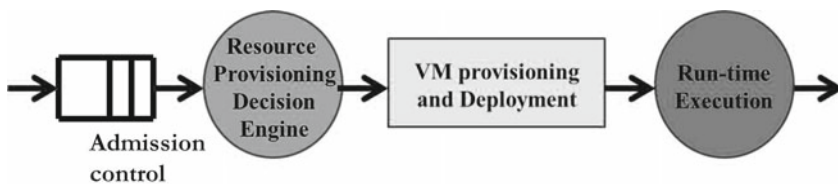


Fig. 1 Request provisioning and servicing steps

exits, the capacity used by that VM is released and becomes available for provisioning and executing the next job [1].

The situations for which a job will be rejected or dropped are as follows:

1. A job is rejected if the global RPDE buffer is full or if there are insufficient capacity of physical and virtual resources, i.e., a job is rejected due to the inadequacy of the resources. In this chapter, we do not show the computation of DPM for this scenario. This will be considered in the future. We show the computation only for run-time execution, i.e., during the actual service time.
2. A job is dropped either by VM failure or by PM failure after starting the execution on the VM.

3.2 Replication in Cloud

A job in a system may be interrupted/preempted due to any failure of the system components. In order to alleviate the impacts of job interruption, replication of job execution gives a viable mean [3, 14, 15]. Further, data replication and software process redundancy are common practices in cloud computing systems [3, 14, 15]. In addition, Bauer et al. [3] presents different types of VM redundancy for enhancing service reliability in cloud. Thus, in a cloud environment, a VM can be replicated for reliability/availability enhancement purpose. When a primary VM fails, a replicated VM can take over the job running on the failed VM by using a failover mechanism. Note that the primary and the replicated (standby) VMs should be deployed on different PMs to prevent underlying hardware from being a single point of failure. The PM on which primary VM is deployed is referred to as the primary PM. Similarly, the PM on which standby VM is deployed is referred to as the standby PM. In this chapter, we address the cloud service implementation under different replication schemes as in [3] in the following way:

- **Cold Replication Scheme:** In this replication scheme, a standby (cold) acts as the backup for the primary. Both the primary and standby VMs are provisioned with request-specific demand. The cold standby is not running when the primary is functioning normally and no job execution state is copied to standby. Traditionally, the standby VM is kept in *suspended* state [3]. In suspended state, the VM and its resources are disabled from executing jobs and the state of the VM and its resources are saved to nonvolatile data storage. Resources may be deallocated. The state is considered enabled but offline. VM instances can be activated when required. Suspended VM instances are sleeping “deeply.” In case the primary fails, the suspended standby is activated immediately and restarts the execution of jobs [3]. This process is usually automated using a cluster manager.
- **Warm Replication Scheme:** In this replication scheme, both the primary and standby VMs are provisioned with request specific demand, but the job is not executed by standby VM [3]. Execution states and data as well (if necessary) are periodically mirrored to standby. Usually, the standby VM is kept in *paused*

state [3]. In paused state, the VM and its resources are disabled from performing tasks; however, the VM and its resources are still instantiated; resources remain allocated. The state is considered temporarily inactive (or quiescent). Paused VM instances can be activated and made available instantly to recover service when necessary. Paused VM instances are sleeping “lightly,” but fewer platform (e.g., CPU) resources are consumed to maintain paused VMs nearly online. In case the primary fails, the standby is activated instantly and resumes the execution of jobs from the last mirrored state. This process is usually automated using a cluster manager. The advantage of this scheme is that it resumes execution after the failover by standby, does not wait for recovery of the failed instance and failure–recovery is done in the background while the job is being executed.

- **Hot Replication Scheme:** In this method, both the primary and standby (redundant active) start executing the same job in parallel. The job execution continues as long as either a primary or a standby is up. If we assume two systems (either PM or VM) do not fail at the same time and we can recover the failed one before encountering another failure, the job never drops in this scheme. In addition, the probability of subsequent failures of the two VMs is negligible [5]. Thus, the job dropping probability is also negligible. Finally, we do not show the formulation for hot replication.

4 Job Completion Time Formulation

We show the job completion time formulation of a single job executed by a VM deployed on a PM. Before computing the job completion time, we introduce some basic concepts to formulate our problem and use the theory developed in [6, 7, 11] to obtain the Laplace–Stieltjes transform (LST) of the job completion time.

We consider the execution of a job with a given work requirement (as measured by the computation time on a failure-free environment with full processing rate) in the presence of failures. We can elaborately express that work requirement is measured in work units, e.g., the number of instructions to be executed as in the CloudSim simulator [16, 17]. Let x be the work processing requirement of the job, $T(x)$ be the amount of time needed to complete the job, and T_d be the deadline for the completion of the job.

Let the cumulative distribution function (CDF) of the job completion time be $F_T(t, x) = P(T(x) \leq t), t \geq 0$. The LST of job completion time transforms $F_T(t, x)$ to a function $\tilde{F}_T(s, x)$ with complex argument s , given by the integral

$$\tilde{F}_T(s, x) = \int_0^{\infty} e^{-st} dF_T(t, x)$$

Assume that interruption/preemption of job execution due to failure, recovery, and restarting/resuming can be represented by a semi-Markov process (SMP) and each state i ($1 \leq i \leq n$) of the SMP represents a specific state of job execution. If a job has failed and successfully restarted (in case not using checkpointing) or resumed (with checkpointing) from failure, then in these circumstances the SMP will transit to a state which is called recovery state, here denoted by G . Assume T_a is the recovery time to state G for the SMP. It is assumed that there is no further failure during the VM recovery period. T_a is a random variable and its cumulative distribution function, $F_{T_a}(t)$, can be easily computed from the failure–recovery SMP model. $F_{T_a}(t) = \pi_G(t)$, where $\pi_G(t)$ is the transient probability that the SMP is in state G at time t and its LST by $\tilde{F}_{T_a}(s)$.

4.1 Without Checkpointing (Conventional Method)

Let H be the time to the first failure after starting job execution, then conditioning on $H = h$, we have

$$T(x)|_{H=h} = \begin{cases} x, & \text{if } h \geq x \\ h + T_a + T(x), & \text{if } h < x \end{cases} \quad (1)$$

If $h \geq x$, then the job will complete in x units of time. If $h < x$, then a failure occurs before the completion of the job. In this case, there is the recovery time T_a after which the job execution is restarted from its beginning, thus, again, requiring x units of uninterrupted processing time to complete. Writing the LST of $T(x)$, we have

$$\tilde{F}_T(s, x)|_{H=h} = \begin{cases} e^{-sx}, & \text{if } h \geq x \\ e^{-sh} \tilde{F}_{T_a}(s) \tilde{F}_T(s, x), & \text{if } h < x \end{cases} \quad (2)$$

Unconditioning on H , we get

$$\begin{aligned} \tilde{F}_T(s, x) &= \int_{h=0}^{\infty} \tilde{F}_T(s, x)|_{H=h} \gamma_{vm} e^{-\gamma_{vm}h} dh \\ &= e^{-(s+\gamma_{vm})x} + \frac{\gamma_{vm} \tilde{F}_{T_a}(s) \tilde{F}_T(s, x) (1 - e^{-(s+\gamma_{vm})x})}{s + \gamma_{vm}} \\ &= \frac{(s + \gamma_{vm}) e^{-(s+\gamma_{vm})x}}{s + \gamma_{vm} (1 - \tilde{F}_{T_a}(s) (1 - e^{-(s+\gamma_{vm})x}))} \end{aligned} \quad (3)$$

where γ_{vm} is the rate of VM failure.

Without checkpointing, the mean job completion time in the presence of failures is given by

$$T_c = \left(\frac{1}{\gamma_{vm}} + T_r\right)(e^{\gamma_{vm}x} - 1) \tag{T.1}$$

It follows directly from the LST in Eq. (3), using the relation $T_c = -\frac{\partial \tilde{F}_T(s,x)}{\partial s} \Big|_{s=0}$, where T_r is the mean time to recovery (MTTR), which follows from $T_r = -\frac{\partial \tilde{F}_{T_d}(s)}{\partial s} \Big|_{s=0}$.

4.2 Adoption of Checkpointing

There are different types of checkpointing strategies. The working principles of the replication scheme in which checkpointing is adopted for/during execution, is similar to the equidistant checkpointing (refer to Sect. 3.2). Thus, we employ the equidistant checkpointing [5–7] strategy herein and review the formulation presented in [6]. Let the total work requirement of the job be divided into n equal parts. Note that there is a checkpoint at the end of each part, except the last one, thus a total of $(n - 1)$ checkpoints. We take into account the possibility of failure during checkpointing. A failure during a checkpoint causes a rollback to the previous checkpoint. Assume that C_d is the checkpoint duration. We denote its CDF by $F_{C_d}(t)$ and LST by $\tilde{F}_{C_d}(s)$.

Each of the first $n - 1$ job segments requires $x/n + C_d$ units of uninterrupted processing time to complete. The execution times of the first $n - 1$ job segments are independent and identically distributed (*i.i.d*) random variables, each is given by $T(x/n + C_d)$. Note that the checkpoint duration C_d is a random variable; however, it is fixed for a given job segment. Following similar steps, as those used in Sect. 4.1, we obtain the following for the LST of $T(x/n + C_d)$

$$\tilde{F}_T(s, x/n + C_d) = \frac{(s + \gamma_{vm})\tilde{F}_{C_d}(s + \gamma_{vm})e^{-(s+\gamma_{vm})x/n}}{s + \gamma_{vm}(1 - \tilde{F}_{T_d}(s)(1 - \tilde{F}_{C_d}(s + \gamma_{vm})e^{-(s+\gamma_{vm})x/n}))}$$

The last job segment requires x/n units of uninterrupted processing time (it does not include a checkpoint) and its execution time is given by $T(x/n)$. The LST of $T(x/n)$ is obtained directly from Eq. (3)

$$\tilde{F}_T(s, x/n) = \frac{(s + \gamma_{vm})e^{-(s+\gamma_{vm})x/n}}{s + \gamma_{vm}(1 - \tilde{F}_{T_d}(s)(1 - e^{-(s+\gamma_{vm})x/n}))}$$

The total job execution time is the sum of n independent random variables (corresponding to the execution times of the n parts). The first $n - 1$ of which are identically distributed and having the LST $\tilde{F}_T(s, x/n + C_d)$, and the last one having the LST $\tilde{F}_T(s, x/n)$. It follows that

$$\begin{aligned}
\tilde{F}_T(s, x) &= \tilde{F}_T(s, x, n) \\
&= \left[\tilde{F}_T(s, x/n + C_d) \right]^{n-1} \left[\tilde{F}_T(s, x/n) \right] \\
&= \left[\frac{(s + \gamma_{vm})e^{-(s+\gamma_{vm})C_d} e^{-(s+\gamma_{vm})x/n}}{s + \gamma_{vm}(1 - \tilde{F}_{T_a}(s)(1 - e^{-(s+\gamma_{vm})C_d} e^{-(s+\gamma_{vm})x/n}))} \right]^{n-1} \\
&\quad \times \left[\frac{(s + \gamma_{vm})e^{-(s+\gamma_{vm})x/n}}{s + \gamma_{vm}(1 - \tilde{F}_{T_a}(s)(1 - e^{-(s+\gamma_{vm})x/n}))} \right]
\end{aligned} \tag{4}$$

With $n - 1$ equally spaced checkpoints, the mean job completion time in the presence of failures is given by

$$T_c = \left(\frac{1}{\gamma_{vm}} + T_r \right) \left[(n - 1)(e^{\gamma_{vm}(x/n + C_d)} - 1) + (e^{\gamma_{vm}x/n} - 1) \right] \tag{T.2}$$

In this formulation, the inversion of the Laplace transform $\tilde{F}_T(s, x)/s$ yields the distribution of the job completion time $F_T(t, x)$,

$$F_T(t, x) = \text{Inverse LST} \left[\frac{\tilde{F}_T(s, x)}{s} \right] \tag{5}$$

5 DPM Computation

DPM is defined as the number of dropped requests per million. A job is dropped when the job is not completed within the deadline T_d . Thus, DPM is defined by

$$\text{DPM} = [1 - F_T(T_d, x)] \times 10^6 \tag{6}$$

In this section, we show the DPM formulation based on job completion time computation for different replication schemes.

5.1 Drop Policy

Assume that when a job arrives, the system is up. The job is considered as dropped if the VM or PM fails before completing the job with the assigned work requirement x . This scheme is referred to as drop policy and the distribution function of job completion time is expressed by a defective distribution

$$F_D = e^{-(\gamma_{vm} + \gamma_{pm})x} \cdot u(t - x)$$

where γ_{vm} and γ_{pm} are the rates of VM and PM failure, respectively. Further, $u(t - x)$ is the unit step function and can be expressed as,

$$u(t - x) = \begin{cases} 1, & t \geq x \\ 0, & t < x \end{cases}$$

Thus, the DPM value in this scheme is computed by,

$$DPM = [1 - F_D] \times 10^6$$

5.2 Cold Replication Scheme

We present the CTMC (it is a special case of an SMP where all the holding times are exponentially distributed) [10, 18] for cold replication scheme as shown in Fig. 2. In this model, state 0 represents the state that both the primary and the standby systems are UP and both the primary and the standby VMs are successfully provisioned where only the primary is executing the job. Any of them can fail with rates γ_{vm} for VM failures and γ_{pm} for PM failures. If the primary VM fails, the system enters state 1, in which it is not able to execute the job, i.e., unavailable. The unavailability in states 1 is not observable until the failure is detected. The failure is detected at the rate δ_{vm} upon which the system enters state 3. In this state, we perform failover, and standby assumes the role of primary (system enters state 5) and restarts the execution. The failover time is assumed to be exponentially distributed with rate τ_c . In the meantime, provisioning of a backup VM is started on another PM at the rate τ_{pv} and the system returns to 0.

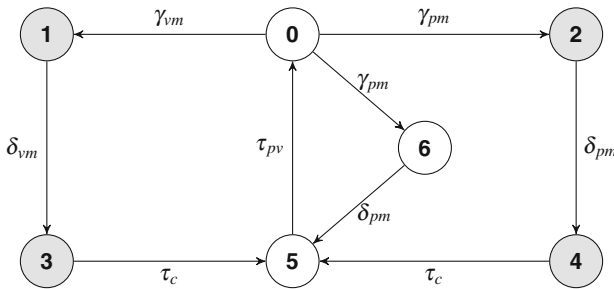
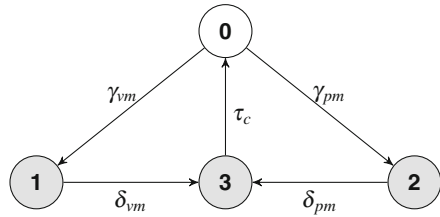


Fig. 2 CTMC model for system with cold replication

Fig. 3 Approximated CTMC model for system with cold replication



The PM on which the primary VM is executing may also fail at the rate γ_{pm} upon which the system goes to state 2. A PM failure (system is in state 2) is detected at the rate δ_{pm} and the system transits to state 4. As in the case of primary VM failure, the standby is switched to primary (system is in state 5). A new backup is started with rate τ_{pv} on an available node to take the system back to state 0. It accumulates the time for the allocation of physical and virtual resources and the provisioning of the VM.

Note that in this model, the standby VM does not execute the job and no execution state is copied from primary to it. It is kept ready so that it can immediately take the role of primary if there is any failure of primary. Standby VM is in deep sleep mode where the probability of VM failure in this mode is negligible. Thus, no standby VM failure is considered; only the underlying standby PM failure causes the standby VM to fail.

In case of standby PM failure, the model traverses in a similar manner as in the case of primary PM failure except the transition of switchover. If it fails, the system transits to state 6, and after detecting the failure system is in state 5. In state 5, a new backup is started on an available node and the system returns to state 0.

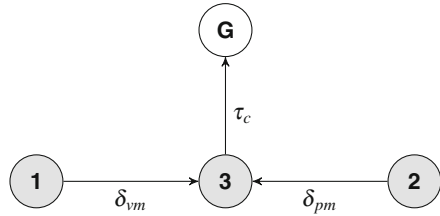
For the completion time distribution we reduce the model shown in Fig. 2, because standby PM failure–recovery and backup provisioning do not have any impact for executing jobs. Thus, we use the approximated CTMC model for the cold replication scheme as shown in Fig. 3.

In this scheme, a failed job is restarted upon recovery, thus checkpointing is not adopted.

5.2.1 Formulation Without Checkpointing

The CTMC model shown in Fig. 4 shows the state transitions after a failure (either VM or PM) has occurred. State G is a recovery (or absorption) state, which represents the state in which the job has been successfully restarted from failure. Let T_a be the time to recover to state G for this CTMC. It is assumed that there is no further failure during recovery period. Suppose a failure occurs at time $t = 0$, hence the CTMC of Fig. 4 has just entered state 1 if it is the failure of VM or entered state 2 if it is the failure of PM. Since the state G is the absorbing state of the CTMC, the time to absorption can be represented by $F_{T_a}(t) = \pi_G(t)$. With probability $\gamma_{vm}/(\gamma_{vm} + \gamma_{pm})$,

Fig. 4 Failure–recovery model for cold replication



the time distribution is $HYP0(\delta_{vm}, \tau_c)$, while with probability $\gamma_{pm}/(\gamma_{vm} + \gamma_{pm})$, the time distribution is $HYP0(\delta_{pm}, \tau_c)$. Therefore,

$$F_{T_a}(t) = \frac{\gamma_{vm}}{\gamma_{vm} + \gamma_{pm}} \left(1 - \frac{\tau_c}{\tau_c - \delta_{vm}} e^{-\delta_{vm}t} + \frac{\delta_{vm}}{\tau_c - \delta_{vm}} e^{-\tau_c t} \right) + \frac{\gamma_{pm}}{\gamma_{vm} + \gamma_{pm}} \left(1 - \frac{\tau_c}{\tau_c - \delta_{pm}} e^{-\delta_{pm}t} + \frac{\delta_{pm}}{\tau_c - \delta_{pm}} e^{-\tau_c t} \right), \quad t \geq 0 \quad (7)$$

where $\gamma_{vm}/(\gamma_{vm} + \gamma_{pm})$ is the probability of VM failure, and the segment associated with it is the failure–recovery time distribution of VM. We can derive the mean time to recovery for a VM failure from this segment of distribution. Similarly, $\gamma_{pm}/(\gamma_{vm} + \gamma_{pm})$ is the probability of PM failure, and the segment associated with it is the failure–recovery time distribution of PM. Mean time to recovery from a PM failure can be derived from this segment of distribution as well.

We represent the job completion time distribution, $\tilde{F}_T(s, x)$ using Eq. (3) by

$$\tilde{F}_T(s, x) = \frac{(s + \gamma_{vm} + \gamma_{pm})e^{-(s+\gamma_{vm}+\gamma_{pm})x}}{s + (\gamma_{vm} + \gamma_{pm})(1 - \tilde{F}_{T_a}(s)(1 - e^{-(s+\gamma_{vm}+\gamma_{pm})x}))} \quad (8)$$

where $\tilde{F}_{T_a}(s)$ can be derived from Eq. (7).

Now, we can represent the mean job completion time, T_c by,

$$T_c = \left(\frac{1}{\gamma_{vm} + \gamma_{pm}} + T_r \right) (e^{(\gamma_{vm}+\gamma_{pm})x} - 1) \quad (9)$$

where the mean time to recovery, T_r is

$$T_r = \frac{\gamma_{vm}}{\gamma_{vm} + \gamma_{pm}} \left(\frac{1}{\delta_{vm}} + \frac{1}{\tau_c} \right) + \frac{\gamma_{pm}}{\gamma_{vm} + \gamma_{pm}} \left(\frac{1}{\delta_{pm}} + \frac{1}{\tau_c} \right)$$

Applying T_r in Eq. (9), we can get the value of mean job completion time in the cold replication. If the completion time exceeds the deadline, then we compute the deadline violating probability, $1 - F_T(T_d, x)$ by Eq. (5). Afterwards, we substitute the value in Eq. (6) and get the DPM value.

5.3 Warm Replication Scheme

Figure 5 shows the CTMC for warm replication scheme. State 0 represents the state that both the primary and the standby systems are UP and both the primary and the standby VMs are successfully provisioned where only the primary is executing the job. In addition, the execution states and data (if necessary) are periodically mirrored to standby. Any of the VMs or PMs can fail with rates γ_{vm} for VM failures and γ_{pm} for PM failures. If the primary VM fails, the system enters state 1, in which it is not able to execute the job, i.e., unavailable. The failure is detected at the rate δ_{vm} upon which the system enters state 3. In this state, we perform failover, and standby assumes the role of primary (system enters state 5). The failover time is assumed to be exponentially distributed with rate τ_w . In the meantime, provisioning of a backup VM is started on another PM at the rate τ_{pv} and the system returns to state 0.

The PM on which the primary VM is executing may fail at the rate γ_{pm} upon which the system goes to state 2 and is unavailable. After detection with rate δ_{pm} , the system transits to state 4. In state 4 (as in the case of primary VM failure), the standby PM is switched to primary (system is in state 5). A new backup is started on an available node to take the system back to state 0.

In case of standby VM failure, upon which the system is in state 6, the primary is still executing the job. After the standby failure detection (system is in state 5), a VM is provisioned on another PM with rate τ_{pv} and the system returns to state 0.

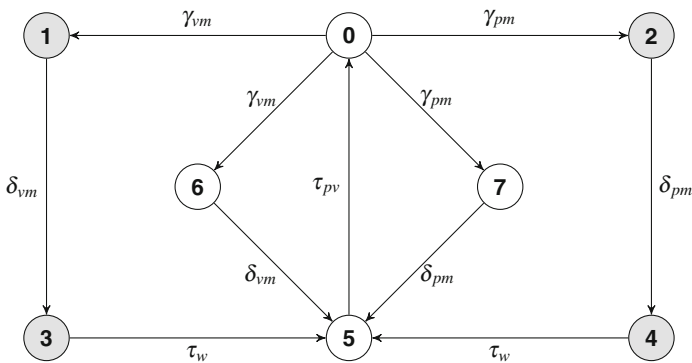
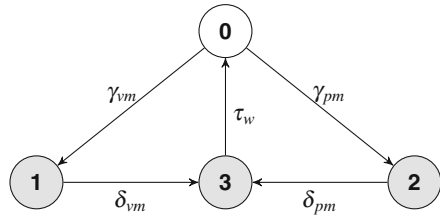


Fig. 5 CTMC model for system with warm replication

Fig. 6 Approximated CTMC model for system with warm replication



In case of standby PM failure, the model traverses in a similar manner as in the case of primary PM failure except the transition of switchover. If it fails, system transits to state 7, and after detecting the failure system is in state 5. In state 5, a new backup is started on an available node and the system returns to state 0.

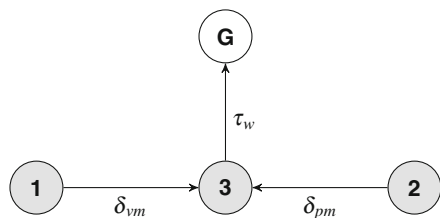
For the completion time distribution, we reduce the model shown in Fig. 5, because standby failure–recovery and backup provisioning do not have any impact for executing jobs. Thus, we use the approximated CTMC model for the warm replication scheme as shown in Fig. 6.

In this scheme, a failed job is resumed from the last saved checkpoint state upon recovery.

5.3.1 Formulation with Checkpointing

The CTMC model shown in Fig. 7 shows the state transitions after a failure (either VM or PM) has occurred. state G is a recovery (or absorption) state, which represents the state in which the job has been successfully resumed from failure. Let T_a be the time to recover to state G for this CTMC. It is assumed that there is no further failure during recovery period. Suppose a failure occurs at time $t = 0$, hence the CTMC of Fig. 7 has just entered state 1 if it is the failure of VM or entered state 2 if it is the failure of PM. Since the state G is the absorbing state of the CTMC, the time to absorption can be represented by $F_{T_a}(t) = \pi_G(t)$. With probability $\gamma_{vm}/(\gamma_{vm} + \gamma_{pm})$, the time distribution is $HYP0(\delta_{vm}, \tau_w)$, while with probability $\gamma_{pm}/(\gamma_{vm} + \gamma_{pm})$, the time distribution is $HYP0(\delta_{pm}, \tau_w)$. Therefore,

Fig. 7 Failure–recovery model for warm replication



$$F_{T_a}(t) = \frac{\gamma_{vm}}{\gamma_{vm} + \gamma_{pm}} \left(1 - \frac{\tau_w}{\tau_w - \delta_{vm}} e^{-\delta_{vm}t} + \frac{\delta_{vm}}{\tau_w - \delta_{vm}} e^{-\tau_w t} \right) + \frac{\gamma_{pm}}{\gamma_{vm} + \gamma_{pm}} \left(1 - \frac{\tau_w}{\tau_w - \delta_{pm}} e^{-\delta_{pm}t} + \frac{\delta_{pm}}{\tau_w - \delta_{pm}} e^{-\tau_w t} \right), \quad t \geq 0 \quad (10)$$

With equidistant checkpointing, we represent the job completion time distribution, $\tilde{F}_T(s, x)$ using Eq. (4.2) by

$$\begin{aligned} \tilde{F}_T(s, x, n) &= \left[\frac{(s + \gamma_{vm} + \gamma_{pm})e^{-(s+\gamma_{vm}+\gamma_{pm})C_d} e^{-(s+\gamma_{vm}+\gamma_{pm})x/n}}{s + (\gamma_{vm} + \gamma_{pm})(1 - \tilde{F}_{T_a}(s)(1 - e^{-(s+\gamma_{vm}+\gamma_{pm})C_d} e^{-(s+\gamma_{vm}+\gamma_{pm})x/n}))} \right]^{n-1} \\ &\times \left[\frac{(s + \gamma_{vm} + \gamma_{pm})e^{-(s+\gamma_{vm}+\gamma_{pm})x/n}}{s + (\gamma_{vm} + \gamma_{pm})(1 - \tilde{F}_{T_a}(s)(1 - e^{-(s+\gamma_{vm}+\gamma_{pm})x/n}))} \right] \end{aligned} \quad (11)$$

where $\tilde{F}_{T_a}(s)$ can be derived from Eq. (10).

Now, we can represent the mean job completion time, T_c by,

$$T_c = \left(\frac{1}{\gamma_{vm} + \gamma_{pm}} + T_r \right) \left\{ (n-1)(e^{(\gamma_{vm}+\gamma_{pm})(x/n+C_d)} - 1) + (e^{(\gamma_{vm}+\gamma_{pm})x/n} - 1) \right\} \quad (12)$$

where the mean time to recovery, T_r is

$$T_r = \frac{\gamma_{vm}}{\gamma_{vm} + \gamma_{pm}} \left(\frac{1}{\delta_{vm}} + \frac{1}{\tau_w} \right) + \frac{\gamma_{pm}}{\gamma_{vm} + \gamma_{pm}} \left(\frac{1}{\delta_{pm}} + \frac{1}{\tau_w} \right)$$

Applying T_r in Eq. (12), we can get the value of mean job completion time in the warm replication. If the completion time exceeds the deadline, then we compute the deadline violating probability, $1 - F_T(T_d, x)$ by Eq. (5). Afterwards, we substitute the value in Eq. (6) and get the DPM value.

6 Experiments

To verify the validity of our proposed analytical modeling approach, we extend CloudSim [16, 17] for our experiments. The following sections outline the experimental settings. We then compare the analytical modeling approach with simulation in terms of job completion time. Furthermore, the results of sensitivity analysis of parameter values are presented.

6.1 *CloudSim Extension*

CloudSim [16, 17] is a widely used extensible simulation framework that supports modeling of virtual resource allocation, job scheduling, and other functionalities. We use CloudSim in the following way to support our experiments:

- A data center network is constructed to connect the host servers that can deploy more than one VMs.
- VM or PM failure, and recovery events are triggered. An event can be generated according to a specified distribution. The failure event data and the recovery event data can be saved to a file so the experiment can be repeated.
- A checkpoint state is generated, transferred, and stored based on the checkpointing method and replication scheme. This module is extensible.
- A job is resumed from a failure based on the checkpoint state and replication scheme. If there is no accessible checkpoint state, it restarts the job from the beginning.

6.2 *Experimental Setup*

We construct a data center network in CloudSim in which each host server can deploy four VMs individually. We configure each host server and each VM in the following way:

- The miph (millions of instructions per hour) of each host server is 4,000,000, the disk size is 100 GB, the memory size is 4 GB, and the bandwidth is 4000 bps.
- The miph of each VM is 1,000,000, and the disk size is 25 GB, the memory size is 1 GB, and the bandwidth is 1000 bps.

6.3 *Model Parameterization*

Table 1 shows the default parameter values used in the experiments. Two different sets of values are used. The values of Set1 proposed by us as well as with reference to [1, 3], and Set2 by [1, 3, 19]. Note that we take the reference of Fig. 5.15 [3] for considering the mean time for VM recovery and provisioning in different replication schemes. The work processing rates of every individual states of CTMC are measured by (unit/h) where 1 means 1,000,000 miph which is equal to the miph of each VM in the UP state.

Table 1 Default value of parameters

Parameters	Description	Values	
		Set1	Set2
$1/\gamma_{vm}$	Mean time to failure of VM (MTTF_VM)	48 h	240 h
$1/\gamma_{pm}$	Mean time to failure of PM (MTTF_PM)	192 h	720 h
$1/\delta_{vm}$	Mean time for failure detection of VM	2 s	2 s
$1/\delta_{pm}$	Mean time for PM failure detection of PM	5 s	5 s
$1/\tau_{rb}$	Mean time for VM recovery by rebooting	5 min	5 min
$1/\tau_{pv}$	Mean time for VM provisioning in a new PM	15 min	30 min
a	Coverage factor of VM recovery by rebooting	0.80	0.80
$1/\tau_c$	Mean time for failover in cold replication	2 min	2 min
$1/\tau_w$	Mean time for failover in warm replication	1 min	1 min
C_n	Number of checkpoints ($n - 1$)	3	3
C_d	Checkpoint duration	1 min	1 min
x	Amount of work requirements	50 unit	300 unit

6.4 Numerical Results

In this section, we show the numerical results for job completion time, CDF of job completion time, and DPM of the models analyzed in Sect. 5. Note that we show the computation for both sets of parameter values side by side. Any deviations from these parameter values are explicitly noted.

6.4.1 Comparison Between Simulative Solution and Analytical Modeling Solution

Note that in our simulation, the sample size for every individual execution is 100,000.

In Figs. 8 and 9 we show the comparison of mean job completion time between simulative solution and analytic modeling solution in cold and warm replication schemes, respectively. We show the computation with different work requirements

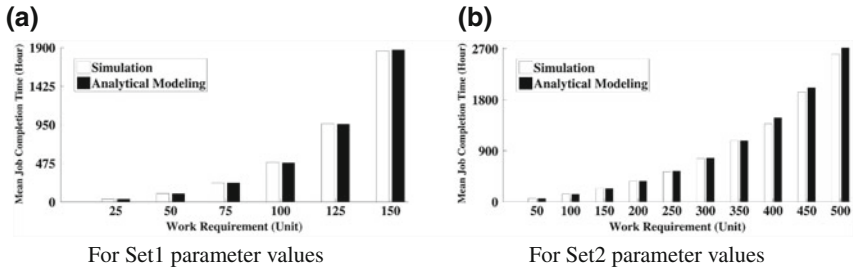


Fig. 8 Comparison of mean job completion time between simulative solution and analytic modeling solution in cold replication for different work requirements

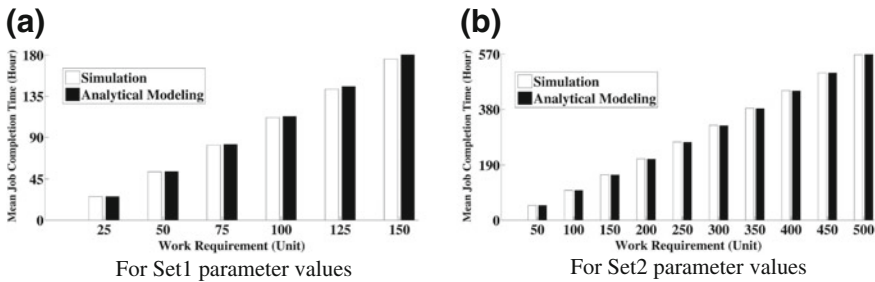


Fig. 9 Comparison of mean job completion time between simulative solution and analytic modeling solution in warm replication for different work requirements

Table 2 95 % confidence interval of job completion time computed by simulation

Work requirement	95 % confidence interval of job completion time (h)					
	Cold replication (Set1)			Warm replication (Set1)		
	Lower limit	Mean	Upper limit	Lower limit	Mean	Upper limit
25	35.1995	35.3057	35.4119	25.7373	25.7448	25.7524
50	102.8789	103.3031	103.7273	52.9143	52.9356	52.9569
75	233.8017	234.9451	236.0886	81.5254	81.5646	81.6038
100	484.2511	486.8565	489.4619	111.6056	111.6663	111.7271
125	956.1904	961.6283	967.0663	142.5971	142.6783	142.7594
150	1840.1	1850.9	1861.7	175.4717	175.5823	175.693

for both sets of parameter values. As can be seen, the difference in completion time between these two approaches is negligible when the work requirement is less than or around the MTTF (either VM or PM which one is smaller, here MTTF of VM is smaller than PM, and it is 48 h in Set1, 240 h in Set2), but beyond that the difference increases gradually.

In Table 2, we show the 95 % confidence interval of job completion time computed by simulation in cold and warm replication schemes for the first set of parameter values. The lower limits and the upper limits are in close agreement with the means.

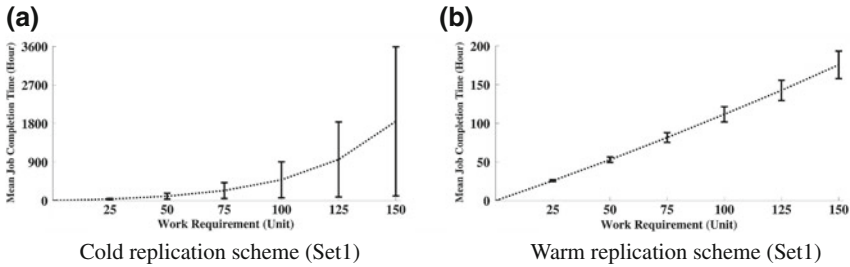


Fig. 10 Mean job completion time and its standard deviation (STD) obtained from simulation

The difference between upper limit and lower limit increases gradually as the work requirement increases.

Figure 10 shows the mean job completion time and its standard deviation obtained from simulation in cold and warm replication schemes. We show the computation for the first set of parameter values. The standard deviations are plotted in both sides around the mean values by filled rectangles. In cold replication, mean job completion time as well as its standard deviation increases rapidly with gradual increase in work requirement. On the other hand, in warm replication, completion time as well as its standard deviation increases gradually as the work requirement increases. Further, Fig. 11 compares the mean job completion times by the different replication schemes. As can be seen, the job completion time in cold replication is comparatively much higher than in warm replication scheme, because in this scheme we need to restart the job every time upon the occurrence of a failure and rises rapidly especially if the work requirement is greater than the MTTF (either VM or PM which one is smaller, here MTTF of VM is smaller than PM, and it is 48 h in Set1, 240h in Set2). On the other hand, in warm replication we can resume the job execution from the last saved checkpoint state after recovery and the time for activating the paused standby VM is comparatively quite short [3]. Consequently, the mean job completion time with warm replication is much shorter than cold replication case regardless of the amount of work requirements.

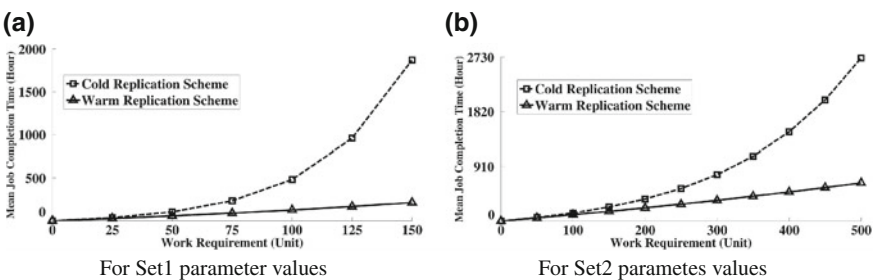


Fig. 11 Mean job completion time for different work requirements

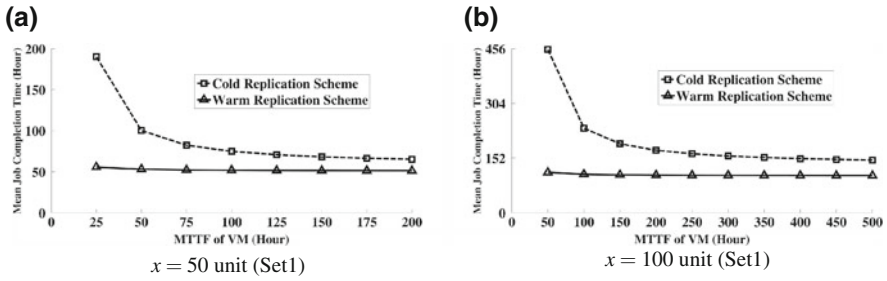


Fig. 12 Mean job completion time by varying the MTTF of VM

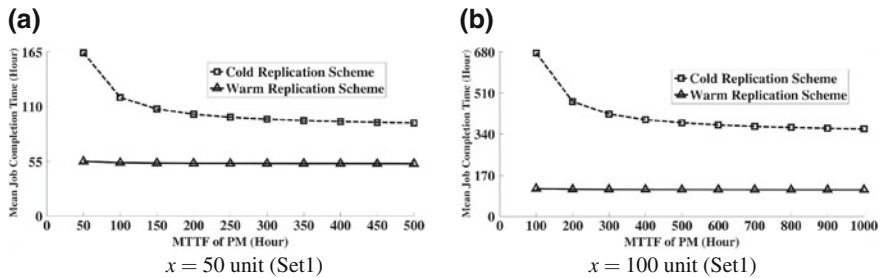


Fig. 13 Mean job completion time by varying the MTTF of PM

In Fig. 12, we show the mean job completion time by varying the MTTF of VM. We show the computation for two different work requirements, $x = 50$ unit, and $x = 100$ unit in the replication schemes considering the first set of parameter of values. Mean job completion time in warm replication scheme improves a little as the MTTF increases. On the other hand, in cold replication scheme, the completion time decreases gradually as MTTF increases. This is because, in warm replication scheme, we can resume the execution of the failed job from the last saved checkpoint state upon recovery, thereby the job completion does not vary significantly. On the other hand, in cold replication, we need to restart the failed job upon recovery, and as the MTTF increases, the frequency of failure decreases, consequently resulting in the improvement of completion time.

Similar to Fig. 12, we show the mean job completion time by varying the MTTF of PM in Fig. 13. The graphs behave similarly as in Fig. 12.

Figure 14 plots the mean job completion time versus work requirement for the different number of checkpoints in warm replication scheme. As can be seen, with the increase in the number of checkpoints, the job completion time gradually improves, but beyond a certain number of checkpoints, the improvement becomes marginal. Further, when the work requirement value is less than or around the MTTF (as discussed above—refer to Fig. 11), then we get almost the same output by any number of checkpointing value. In general, increasing the number of checkpoints increases the operational overhead that also increases complexity. Thus, a cloud service provider

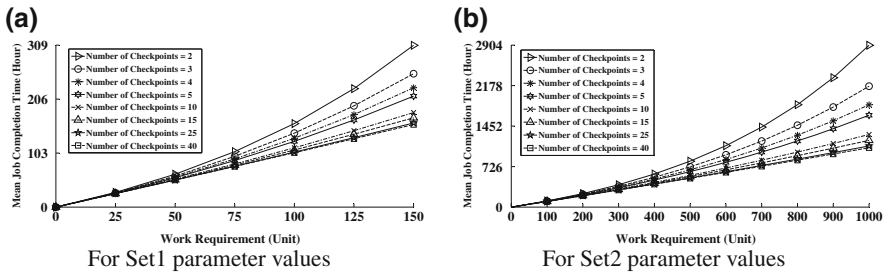


Fig. 14 Mean job completion time versus work requirement for different number of checkpoints in warm replication

needs to design the frequency of checkpoints in consideration with the trade-off between the impact on job completion time and its overhead.

6.4.2 Job Completion Time Distribution

In Fig. 15, we show the CDF of job completion time in the replication schemes. If the system (either VM or PM) does not encounter any failures for 50h, the job execution completes at $t = 50$ (work requirement is 50 units for the computation in Fig. 15a). Similarly, if the system (either VM or PM) does not encounter any failures for 300h, the job execution completes at $t = 300$ (work requirement is 300 units for the computation in Fig. 15b). Further, the probability of job completion in cold replication scheme approaching 1 is slower than in warm replication scheme, because we cannot save the execution states of jobs, but in warm replication scheme we can save the execution states using checkpointing, and resume the execution from the last saved checkpoint state upon failure–recovery. Since the job completion time distribution in the drop policy is defective, the probability does not reach 1.

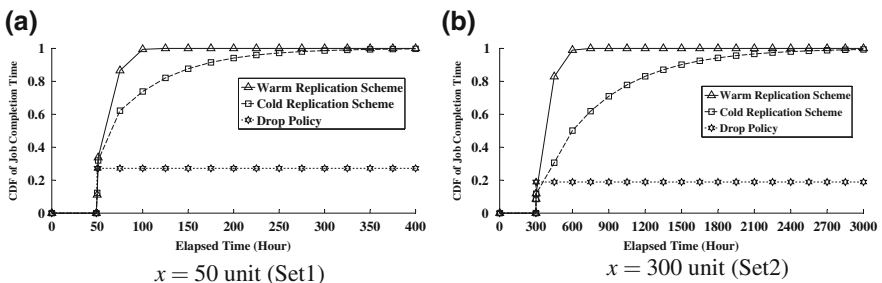


Fig. 15 CDF of job completion time

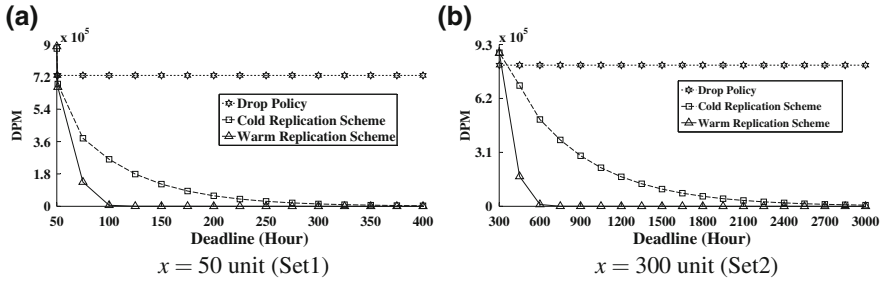


Fig. 16 DPM versus deadline

Table 3 DPM in cold and warm replication

Required DPM	Minimum deadline (h)	
	Cold replication	Warm replication
DPM < 20	6225	887
DPM < 15	6400	890
DPM < 10	6655	938
DPM < 5	7000	963
DPM < 1	7950	1029

6.4.3 Defects per Million (DPM)

Figure 16a shows the DPM by varying the values of deadline $T_d \geq 50$. Similarly, Fig. 16b shows the DPM by varying the values of deadline $T_d \geq 300$. As can be seen, DPM values decrease with longer deadline T_d . Note that in the drop policy the DPM values are not influenced by the deadline. Further, the minimum deadlines to achieve the required DPM are summarized in Table 3. If we need to make the DPM value less than 20, the deadline needs to be set to 6225 h at minimum in cold replication scheme. In warm replication scheme, the minimum deadline to guarantee the required DPM value is less than one seventh of that in cold replication scheme. Note that the computation in Table 3 is shown considering the second set of parameter values only.

7 Conclusion and Future Work

In this chapter, we developed analytical models of job execution through VM for cloud computing systems. The method to compute the DPM based on the checkpointing method for job execution in different replication schemes is provided. The analytical modeling approach was validated with simulation using CloudSim. The impact of a different number of checkpoints was considered to identify a suitable

value. In the numerical examples, we showed the effectiveness of replication schemes on DPM as well as job completion time in the cloud computing systems.

A future direction is to extend this work for nonhomogeneous jobs with different priorities and batch-task jobs. Further, in future work, we will take into account the computation of DPM due to the limited queue (buffer) as well as inadequacy of resources (e.g., VM or PM).

References

1. Ghosh R, Longo F, Naik VK, Trivedi KS (2013) Modeling and performance analysis of large scale iaas clouds. *Future Gener Comput Syst* 29(5):1216–1234
2. Avizienis A, Laprie J-C, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans Dependable Secure Comput* 1(1):11–33
3. Bauer E, Adams R (2012) Reliability and availability of cloud computing. Wiley-IEEE Press
4. Trivedi K, Wang D, Hunt J (2010) Computing the number of calls dropped due to failures. In: IEEE 21st international symposium on software reliability engineering (ISSRE). IEEE, pp 11–20
5. Mondal S, Yin X, Muppala J, Alonso Lopez J, Trivedi K (2015) Defects per million computation in service-oriented environments. In: IEEE transactions on services computing, vol 8, no 1, pp 32–46
6. Nicola V (1994) Checkpointing and the modeling of program execution time, series of mem-
oranda informatica. University of Twente, Department of Computer Science and Department
of Electrical Engineering. <http://books.google.com.hk/books?id=b4pzHAAACAAJ>
7. Duda A (1983) The effects of checkpointing on program execution time. *Inf Process Lett* 16:221–229
8. Systems C (2004) Introduction to performance management
9. Johnson C, Kogan Y, Levy Y, Saheban F, Tarapore P (2004) Voip reliability: a service provider's perspective. *IEEE Commun Mag* 42(7):48–54 July
10. Mondal SK, Muppala JK, Machida F, Trivedi KS (2014) Computing defects per million in cloud caused by virtual machine failures with replication. In: IEEE 20th Pacific Rim international symposium on dependable computing (PRDC). IEEE, pp 161–168
11. Kulkarni VG, Nicola VF, Trivedi KS (1987) The completion time of a job on multimode systems. *Adv Appl Probab* 19:932–954
12. Mondal SK, Muppala JK (2014) Defects per million (dpm) evaluation for a cloud dealing with vm failures using checkpointing. In: 44th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, pp 672–677
13. Di S, Kondo D, Cappello F (2013) Characterizing cloud applications on a google data center. In: 42nd international conference on parallel processing (ICPP). IEEE, pp 468–473
14. Ben-Yehuda O, Schuster A, Sharov A, Silberstein M, Iosup A (2012) Expert: Pareto-efficient task replication on grids and a cloud. In: IEEE 26th international parallel distributed processing symposium (IPDPS), pp 167–178
15. Burgener E (2009) Replication and cloud computing are inseparable. Infostor. http://www.infostor.com/index/articles/display/6599123393/articles/infostor/backup-and_recovery/cloud-storage/replication-and_cloud.html
16. Calheiros R, Ranjan R, De Rose C, Buyya R (2009) Cloudsim: a novel framework for modeling and simulation of cloud computing infrastructures and services. [arXiv:0903.2525](https://arxiv.org/abs/0903.2525)
17. Calheiros R, Ranjan R, Beloglazov A, De Rose C, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw: Pract Exper* 41(1):23–50

18. Mondal SK, Muppala JK (2014) Energy modeling of different virtual machine replication schemes in a cloud data center. In: IEEE international conference on internet of things (iThings), and green computing and communications (GreenCom), IEEE and cyber, physical and social computing (CPSCom). IEEE, pp 486–493
19. Machida F, Nicola VF, Trivedi KS (2014) Job completion time on a virtualized server with software rejuvenation. *ACM J Emerg Technol Comput Syst (JETC)* 10(1):10

Linear Algebraic Methods in RESTART Problems in Markovian Systems

Stephen Thompson, Lester Lipsky and Søren Asmussen

Abstract A task with ideal execution time ℓ is handled by a Markovian system with features similar to the ones in classical reliability. The Markov states are of two types, UP and DOWN, such that the task only can be processed in an UP state. Upon entrance to a DOWN state, processing is stopped and must be restarted from the beginning upon the next entrance to an UP state. The total task time $X = X_r(\ell)$ (including restarts and pauses in failed states) is investigated with particular emphasis on the expected value $\mathbb{E}[X_r(\ell)]$, for which an explicit formula is derived that applies for all relevant systems. In general, transitions between UP and DOWN states are interdependent, but simplifications are pointed out when the UP to DOWN rate matrix (or the DOWN to UP) has rank one. A number of examples are studied in detail and an asymptotic exponential form $\exp(\beta_m \ell)$ is found for the expected total task time $\mathbb{E}[X(\ell)]$ as $\ell \rightarrow \infty$. Also, the asymptotic behavior of the total distribution, $H_r(x|\ell) \rightarrow \exp(-x\gamma(\ell))$, as $x \rightarrow \infty$ is discussed.

1 Background

For some systems, failure is rare enough that it can be ignored or dealt with as an afterthought. For many systems, failure is so common that the design choice of how to deal with it may have a significant impact on the performance of the system. Many papers discuss methods of failure recovery and analyze their complexity in

S. Thompson (✉) · L. Lipsky
Department of Computer Science and Engineering, 371 Fairfield Way,
Unit 4155, University of Connecticut, Storrs, CT 06269, USA
e-mail: sat03002@engr.uconn.edu

L. Lipsky
e-mail: lester@engr.uconn.edu

S. Asmussen
Department of Mathematics, Aarhus University,
Ny Munkegade, 8000 Aarhus C, Denmark
e-mail: asmus@math.au.dk

one or more metrics, like restartable processors in [6], or Stage Checkpointing in [7], etc. There are many specific and distinct failure recovery schemes, but they can be grouped into three broad classes

RESUME, also referred to as *preemptive resume* (prs);
REPLACE, also referred to as *preemptive repeat different* (prd);
RESTART, also referred to as *preemptive repeat identical* (pri).

For *RESUME*, when the system fails, it is repaired and then the job continues where it is left off. For *REPLACE*, after being repaired, the system selects a new job from the same distribution. Under *RESTART* the system must start the job from the beginning, necessarily repeating everything that had been done previously.

The first two classes are well understood and discussed in detail in the literature. The analysis of the distribution of completion time when the policy is *RESUME* or *REPLACE* was carried out by Kulkarni et al. [11]. The task distribution for the *RESTART* policy was defined and examined in Kulkarni et al. [12]. The work in [3, 8, 11, 12] clearly suggested that if the task time, L , with distribution function, $F(t)$ and density $f(t)$ had a Matrix Exponential (ME) distribution, then the resulting total service time, X , with distribution $H(x)$ and density $h(x)$ for the *RESUME* and *REPLACE* policies could also be represented by ME distributions.

But the *RESTART* policy resisted detailed analysis. All they could show was that it definitely was *not* ME. However, by assuming that the failures are iid one can derive the Laplace transform of the total time distribution, $H(x)$, from which the mean time, $\mathbb{E}[X]$, can be found. By numerically taking the inverse Laplace transform (see Jagerman [10]), Chimento and Trivedi [5] (following a model proposed by Castillo [4]) were able to find the *RESTART* time distribution for a few cases, but for a limited range ($x \leq 3\mathbb{E}[X]$). However, the properties of $H(x)$ for large x were still unknown. Asmussen et al. [1, 15], showed that if the task time distribution, $F(t)$, has finite support, $H(x)$ is asymptotically exponential. But if its support is infinite, then $H(x)$ decays slower than any exponential. Furthermore, if the tail of the failure distribution decays exponentially at rate β and F decays at exponential rate μ then the tail of H is essentially of order $1/x^\alpha$ where $\alpha = \mu/\beta$. To be precise, let

$$\mu_m = \sup \left\{ \mu \mid \int_0^\infty e^{\mu t} f(t) dt < \infty \right\} \quad (1)$$

also

$$\alpha = \sup \left\{ \rho \mid \int_0^\infty x^\rho h(x) dx < \infty \right\} \quad (2)$$

then $\alpha = \mu_m/\beta$. More recently, Asmussen [2] extended this to non-exponential failure distributions. Most recently [16], the present authors found an expression for the exponential tail that also includes dependence between repair and failure times.

2 Introduction

In this paper, we take a linear algebraic approach and model the *RESTART* problem via a finite-state Markov chain that is also relevant in *Dependability Theory* (see e.g., [17]). We partition the state space into Up and Down states so that the transitions between the two sets generate a *Markov Renewal Process* (MRP). The times spent alternately in the two sets are not necessarily independent or even identically distributed. We then show how this relates to many results in Dependability theory. (See e.g., [4, 16, 17] for details.)

Next, we focus on the *RESTART* process where we are primarily interested in the total time to complete a single job, $\mathbb{E}[X_r(\ell)]$ (the subscript, r , indicates inclusion of all repair times). In this case, the system stays in the UP states for a time $U < \ell$, or the job ends, where ℓ is the time the job would have taken if there had been no failures. If the job fails, then it goes to Down states until repair is complete, and then returns to the Up states to try again. Since the job must ultimately complete, this is not an MRP, but can be called a *Truncated Markov Renewal Process* (TMRP). We then derive the mean total time under *RESTART* $\mathbb{E}[X_r(\ell)]$. This includes not only the repair times but the dependence between successive failures and repairs. After all, there may be several ways that a system could fail, and several degrees of repair. The matrix expression given for $\mathbb{E}[X_r(\ell)]$ is amenable to computation using standard software. We then show how this can be extended to systems where ℓ comes from some distribution, $F(\ell)$.

We provide several examples, including systems with (hot or cold) backup servers, including online repair, and also consider systems where *RESTART* can begin after only one server has been repaired. Some of these examples have non-exponential failure and repair distributions. Finally, we discuss $\gamma(\ell)$, the exponential tail for $H_r(x|\ell)$, and how the formula for its computation simplifies under certain conditions.

We adhere to the notation that upper-case boldfaced characters (except for the *expectation symbol* \mathbb{E}) are matrices, lower-case bold faced characters are vectors, where the “'” symbol denotes column vector.

3 Properties of the Intensity Matrix \mathbf{Q}

The intensity matrix, \mathbf{Q} is also called the *infinitesimal generator* by Gross and Harris [9], the *transition rate matrix* by Lipsky [13], as well as other names. Some of this material may be found, perhaps from a different point of view in, e.g., [1, 2, 8, 13].

Consider a finite state Markovian system with c exponential states, where \mathbf{M} is a diagonal matrix such that $(\mathbf{M})_{ii}$ is the departure rate of state i , and \mathbf{P}_{ij} is the probability the system will transition from $i \rightarrow j$ after leaving i . Let $\mathbf{\epsilon}'$ be a column vector of dimension c with all 1's. Assume that \mathbf{P} is irreducible, and since the row sums of \mathbf{P} add up to 1, we can write $\mathbf{P}\mathbf{\epsilon}' = \mathbf{\epsilon}'$. From the transition and rate matrices, \mathbf{P} and

\mathbf{M} , the generating matrix, $\mathbf{Q} = \mathbf{M}(\mathbf{P} - \mathbf{I})$ can be constructed. Clearly, \mathbf{Q} has the property $\mathbf{Q} \boldsymbol{\varepsilon}' = \mathbf{o}'$, where \mathbf{o}' is a column vector of all 0's.

The state space is partitioned into two sets, E_u and E_d of dimension c_u and c_d , respectively, ($c_u + c_d = c$), which we call the *UP* states and the *DOWN*, (or *repair*) states. \mathbf{Q} can be written as

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{uu} & \mathbf{Q}_{ud} \\ \mathbf{Q}_{du} & \mathbf{Q}_{dd} \end{bmatrix} = \begin{bmatrix} -\mathbf{B}_u & \mathbf{B}_u \mathbf{Y}_u \\ \mathbf{B}_d \mathbf{Y}_d & -\mathbf{B}_d \end{bmatrix}. \tag{3}$$

Since \mathbf{P} is irreducible, it follows that \mathbf{B}_u and \mathbf{B}_d have inverses, then $\mathbf{Y}_u = \mathbf{V}_u \mathbf{Q}_{ud}^{-1}$ and $\mathbf{Y}_d = \mathbf{V}_d \mathbf{Q}_{du}$. Obviously, $\mathbf{B}_v = -\mathbf{Q}_{vv}$ and $\mathbf{V}_v = \mathbf{B}_v^{-1}$ for $v \in \{u, d\}$. Because $\mathbf{Q} \boldsymbol{\varepsilon}' = \mathbf{o}'$, we have

$$\mathbf{Q}_{uu} \boldsymbol{\varepsilon}'_u + \mathbf{Q}_{ud} \boldsymbol{\varepsilon}'_d = \mathbf{o}'_u \quad \text{or} \quad -\mathbf{B}_u \boldsymbol{\varepsilon}'_u + \mathbf{B}_u \mathbf{Y}_u \boldsymbol{\varepsilon}'_d = \mathbf{o}'_u$$

and

$$\mathbf{Q}_{dd} \boldsymbol{\varepsilon}'_d + \mathbf{Q}_{du} \boldsymbol{\varepsilon}'_u = \mathbf{o}'_d \quad \text{or} \quad -\mathbf{B}_d \boldsymbol{\varepsilon}'_d + \mathbf{B}_d \mathbf{Y}_d \boldsymbol{\varepsilon}'_u = \mathbf{o}'_d,$$

from which it follows that

$$\mathbf{Y}_u \boldsymbol{\varepsilon}'_d = \boldsymbol{\varepsilon}'_u \quad \text{and} \quad \mathbf{Y}_d \boldsymbol{\varepsilon}'_u = \boldsymbol{\varepsilon}'_d.$$

$\boldsymbol{\varepsilon}'_u$ and $\boldsymbol{\varepsilon}'_d$ are column vectors of all 1's, of dimension c_u and c_d , respectively, while \mathbf{o}'_u and \mathbf{o}'_d are column vectors of all 0's. The \mathbf{Y} matrices contain the correlation for transitions between the UP and DOWN states.

The \mathbf{B} matrices generate distributions for the time spent in each subsystem. Assuming that the process started in state $j \in E_u$, then

$$[\bar{\mathbf{G}}_u(t)]_{ji} := [\exp(-t \mathbf{B}_u)]_{ji} \quad (i, j \in E_u)$$

is the probability that the system is still UP and in state $i \in E_u$ at time t . Also, define the auxiliary matrix

$$\mathbf{G}_u(t) := \mathbf{I}_u - \bar{\mathbf{G}}_u(t).$$

The matrix

$$[\mathbf{g}_u(t)]_{jk} := [\bar{\mathbf{G}}_{u(t)} \mathbf{B}_u \mathbf{Y}_u]_{jk} \quad (j \in E_u, \quad k \in E_d)$$

is the probability density that the system will fail at time t , and transfer to state $k \in E_d$. Note that it can also be written as $\mathbf{g}_u(t) = \exp(-t \mathbf{B}_u) \mathbf{Q}_{ud}$. Then,

$$\mathbf{Y}_u = \int_0^\infty \mathbf{g}_u(t) dt,$$

¹For clarity, note that in general the boldfaced subscripts identify the entire matrix, not its components. The components are identified by the notation, $(\mathbf{Q}_{ud})_{ij}$, $i \in E_u$, and $j \in E_d$.

showing that $[\mathbf{Y}_u]_{jk}$ is the probability that $k \in E_d$ is the first state in E_d after a period in E_u starting from $j \in E_u$.

In a similar fashion, we define

$$[\tilde{\mathbf{G}}_d(x)]_{km} := [\exp(-x \mathbf{B}_d)]_{km} \quad (k, m \in E_d)$$

and

$$[\mathbf{g}_d(x)]_{kj} := [\tilde{\mathbf{G}}_d(x) \mathbf{B}_d \mathbf{Y}_d]_{kj} \quad (j \in E_u, k \in E_d).$$

Let us suppose that the system starts in an UP state with initial probability vector \mathbf{p}_o , and let

$$U_1, R_1, U_2, R_2, \dots, U_n, R_n, \dots$$

be the random variables denoting the time spent sequentially in UP and DOWN states. The complementary distribution for the first time to failure is

$$\tilde{H}_{U_1}(t) := \Pr[U_1 > t] = \mathbf{p}_o \tilde{\mathbf{G}}_u(t) \boldsymbol{\varepsilon}'_u = \mathbf{p}_o \exp(-t \mathbf{B}_u) \boldsymbol{\varepsilon}'_u$$

Expressions of this form are often called *Matrix Exponential* (ME) distributions. As is well-known, such functions can be written as a sum of exponentials times polynomials (See, e.g., Chap. 3 of Lipsky [13]),

$$\tilde{H}_{U_1}(t) = \sum_{k=1}^{r_u} q_k(t \beta_k) e^{-t \beta_k}$$

where β_k is an eigenvalue of \mathbf{B}_u , q_k is a polynomial of degree one less than the multiplicity of the k th eigenvalue, and $r_u \leq c_u$ is the number of roots. The roots must all have positive real parts, and in particular β_m , the minimum eigenvalue, must be real. Then for large t ,

$$\tilde{H}_{U_1}(t) \sim q_m(t \beta_m) e^{-t \beta_m}. \tag{4}$$

The density and distribution functions are

$$h_{U_1}(t) = \mathbf{p}_o \tilde{\mathbf{G}}_u(t) \mathbf{B}_u \boldsymbol{\varepsilon}'_u = \mathbf{p}_o \exp(-t \mathbf{B}_u) \mathbf{B}_u \boldsymbol{\varepsilon}'_u,$$

and

$$H_{U_1}(t) = 1 - \tilde{H}_{U_1}(t) = \Pr[U_1 \leq t] = \mathbf{p}_o \mathbf{G}_u(t) \boldsymbol{\varepsilon}'_u.$$

The moments of the distribution can also be calculated by using the well-known formula

$$\mathbf{E}[U_1^n] = n! \mathbf{p}_o \mathbf{V}_u^n \boldsymbol{\varepsilon}'_u,$$

where $\mathbf{V}_u = \mathbf{B}_u^{-1}$.

The repair times depend on the previous failure process. Consider that the system goes from failure to repair to failure indefinitely. The joint distribution for the first failure and repair cycle is given by

$$h_{U_1 R_1}(t, x) = \mathbf{p}_o \bar{\mathbf{G}}_u(t) \mathbf{B}_u \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \mathbf{Y}_d \boldsymbol{\varepsilon}'_u \tag{5}$$

If we integrate over t , we get

$$h_{R_1}(x) := \int_0^\infty h_{U_1 R_1}(t, x) dt = \mathbf{p}_o \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d$$

where we used

$$\int_0^\infty \bar{\mathbf{G}}_u(t) \mathbf{B}_u dt = \mathbf{I}_u.$$

From this we see that $\mathbf{p}_o \mathbf{Y}_u$ is the initial vector for the start of the first repair epoch, averaged over all possible t , the initial failure time.

As with U_1 the moments of R_1 are

$$\mathbb{E}[R_1^n] = n! \mathbf{p}_o \mathbf{Y}_u \mathbf{V}_d^n \boldsymbol{\varepsilon}'_d$$

where $\mathbf{V}_d = \mathbf{B}_d^{-1}$. In general

$$h_{U_n}(t) = \mathbf{p}_o (\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \bar{\mathbf{G}}_u(t) \mathbf{B}_u \boldsymbol{\varepsilon}'_u, \tag{6}$$

$$h_{U_n R_n}(t, x) = \mathbf{p}_o (\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \bar{\mathbf{G}}_u(t) \mathbf{B}_u \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d,$$

and

$$h_{R_n}(x) = \mathbf{p}_o (\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d.$$

$$h_{R_n U_{n+1}}(x, t) = \mathbf{p}_o (\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \mathbf{Y}_d \bar{\mathbf{G}}_u(t) \mathbf{B}_u \boldsymbol{\varepsilon}'_u.$$

More complicated orderings can be written down without difficulty.

In any case, notice that

$$\lim_{n \rightarrow \infty} (\mathbf{Y}_u \mathbf{Y}_d)^{n-1} = \boldsymbol{\varepsilon}'_u \mathbf{p}_u,$$

where \mathbf{p}_u is the left eigenvector of $\mathbf{Y}_u \mathbf{Y}_d$ with eigenvalue 1, and normalized so that $\mathbf{p}_u \boldsymbol{\varepsilon}'_u = 1$. That is,

$$\mathbf{p}_u (\mathbf{Y}_u \mathbf{Y}_d) = \mathbf{p}_u. \tag{7}$$

Therefore,

$$h_{U_\infty}(t) := \lim_{n \rightarrow \infty} h_{U_n}(t) = \mathbf{p}_u \bar{\mathbf{G}}_u(t) \mathbf{B}_u \boldsymbol{\varepsilon}'_u.$$

Similarly,

$$h_{R_\infty}(x) := \lim_{n \rightarrow \infty} h_{R_n}(x) = \mathbf{p}_u \mathbf{Y}_u \bar{\mathbf{G}}_{d(x)} \mathbf{B}_d \mathbf{e}'_d.$$

Also, define $\mathbf{p}_d := \mathbf{p}_u \mathbf{Y}_u$. It follows directly that $\mathbf{p}_u = \mathbf{p}_d \mathbf{Y}_d$, and $\mathbf{p}_d = \mathbf{p}_d \mathbf{Y}_d \mathbf{Y}_u$.

As n grows large, $[h_{U_n}(x) - h_{U_{n+1}}(x)] \rightarrow 0$. So, the U_n 's become identically distributed, *but* they are *not* necessarily independent. We show this by finding the *auto-covariance*, *lag-k* coefficient, namely

$$\text{Cov}(U, U_{+k}) := \lim_{n \rightarrow \infty} \text{Cov}(U_n, U_{n+k}) = \lim_{n \rightarrow \infty} \{\mathbf{E}[U_n U_{n+k}] - \mathbf{E}[U_n] \mathbf{E}[U_{n+k}]\} =$$

$$\mathbf{p}_u \mathbf{V}_u (\mathbf{Y}_u \mathbf{Y}_d)^k \mathbf{V}_u \mathbf{e}'_u - \mathbf{p}_u \mathbf{V}_u \mathbf{e}'_u \mathbf{p}_u \mathbf{V}_u \mathbf{e}'_u = \mathbf{p}_u \mathbf{V}_u [(\mathbf{Y}_u \mathbf{Y}_d)^k - \mathbf{e}'_u \mathbf{p}_u] \mathbf{V}_u \mathbf{e}'_u.$$

It is possible that $\text{Cov}(U, U_{+k}) \neq 0$, and in fact, $\text{Cov}(U, U_{+1}) \neq 0$ unless $(\mathbf{Y}_u \mathbf{Y}_d) = \mathbf{e}'_u \mathbf{p}_u$.² We explore that next, and give examples later.

3.1 What if \mathbf{Y}_u and/or \mathbf{Y}_d Is of Rank 1?

We have seen that \mathbf{Y}_u and \mathbf{Y}_d govern the dependence between epochs, even nonsequential ones. Now, suppose that both \mathbf{Y}_u and \mathbf{Y}_d are of rank 1. Then, they can be written as

$$\mathbf{Y}_u = \mathbf{e}'_u \mathbf{p}_d \quad \text{and} \quad \mathbf{Y}_d = \mathbf{e}'_d \mathbf{p}_u$$

where $\mathbf{p}_u \mathbf{e}'_u = \mathbf{p}_d \mathbf{e}'_d = 1$. In this case, $h_{U_1}(t)$ is the same as before, but

$$h_{U_n}(t) = \mathbf{p}_u \exp(-t \mathbf{B}_u) \mathbf{B}_u \mathbf{e}'_u, \quad \text{for } n > 1$$

and

$$h_{R_n}(t) = \mathbf{p}_d \exp(-t \mathbf{B}_d) \mathbf{B}_d \mathbf{e}'_d \quad \text{for } n \geq 1.$$

So, we can state the following theorem.

Theorem 1 *If \mathbf{Y}_u and \mathbf{Y}_d are both of rank 1, then all UP and DOWN periods are independent, all repair periods are iid, and except perhaps for the first UP period, all failure times are iid. The system then reduces to a two-server loop, with failure distribution given by $h_{U_n}(u)$ (except for $n = 1$) and repair time distribution given by $h_{R_n}(y)$. \square*

²Of course, even if all the Cov coefficients were 0, that would not be sufficient to have independence. Just one nonzero coefficient proves dependence.

If only one of the \mathbf{Y}_a 's, ($a \in \{u, d\}$) is of rank 1, then only certain joint distributions are iid. For instance, suppose that $\mathbf{Y}_d = \boldsymbol{\epsilon}'_d \mathbf{p}_u$, then (5) becomes

$$h_{U_n R_n}(t, x) = \mathbf{p}_1 \exp(-t\mathbf{B}_u) \mathbf{B}_u \mathbf{Y}_u \exp(-x\mathbf{B}_d) \mathbf{B}_d \boldsymbol{\epsilon}'_d$$

for $n > 1$. (For $n = 1$ replace \mathbf{p}_u with \mathbf{p}_o .)

Similarly, suppose that $\mathbf{Y}_u = \boldsymbol{\epsilon}'_u \mathbf{p}_d$, then

$$h_{R_n U_{n+1}}(x, t) = \mathbf{p}_d \exp(-x \mathbf{B}_d) \mathbf{B}_d \mathbf{Y}_d \exp(-t\mathbf{B}_u) \mathbf{B}_u \boldsymbol{\epsilon}'_u.$$

Since the right-hand side of both expressions are independent of n , we state this as another theorem.

Theorem 2 *If $\mathbf{Y}_d = \boldsymbol{\epsilon}'_d \mathbf{p}_u$ then $h_{U_n R_n}(t, x)$ is independent of n and the cycle times, $U_n + R_n$, are iid (except, perhaps, for $n = 1$). similarly, if $\mathbf{Y}_u = \boldsymbol{\epsilon}'_u \mathbf{p}_d$ then $h_{R_n U_{n+1}}(x, t)$ is independent of n and the cycle times, $R_n + U_{n+1}$, are iid. \square*

The Theorem is still true even if neither \mathbf{Y}_u nor \mathbf{Y}_d is of rank 1, but their product is (e.g., $\mathbf{Y}_u \mathbf{Y}_d = \boldsymbol{\epsilon}'_d \mathbf{p}_u$).

4 Relation to Dependability Theory

Before moving on to the major objective of this paper, we touch on the broad subject of *Dependability Theory* which has come to be the umbrella name for *Reliability*, *Availability* and *Checkpointing* as well as other subjects including *RESTART*. There have been innumerable papers and books written on the subject, which we will not even attempt to summarize here. One might start with the books by Trivedi [17], and Rubino and Sericola [14], to name just two. Our purpose is to link our linear algebraic approach to that of the broader field.

4.1 Reliability Theory

Reliability theory involves, among other things, the *Mean Time To Failure* (MTTF). In many applications it is critical to avoid the failure states for as long as possible. The time to repair takes a back place. As an extreme example, it is of no use to be told that *Your parachute can be repaired as soon as you land* after you've jumped out of an airplane. But still, there are other applications where both MTTF and MTTR (*Mean Time To Repair*) are important.

A standard technique for computing this is to just consider

$$MTTF_1 := \mathbf{E}[U_1] = \mathbf{p}_o \mathbf{V}_u \boldsymbol{\epsilon}'_u.$$

But this is only the first UP time. Another way is to find the solution to

$$\boldsymbol{\Pi} \boldsymbol{Q} = \mathbf{o},$$

where $\boldsymbol{\Pi} = [\boldsymbol{\pi}_u, \boldsymbol{\pi}_d]$. This translates to the simultaneous equations

$$\boldsymbol{\pi}_u \mathbf{B}_u = \boldsymbol{\pi}_d \mathbf{B}_d \mathbf{Y}_d,$$

$$\boldsymbol{\pi}_d \mathbf{B}_d = \boldsymbol{\pi}_u \mathbf{B}_u \mathbf{Y}_u,$$

yielding the eigenvector equations

$$\boldsymbol{\pi}_u \mathbf{B}_u = \boldsymbol{\pi}_u \mathbf{B}_u \mathbf{Y}_u \mathbf{Y}_d,$$

$$\boldsymbol{\pi}_d \mathbf{B}_d = \boldsymbol{\pi}_d \mathbf{B}_d \mathbf{Y}_d \mathbf{Y}_u.$$

But we have already seen from (7) that $\mathbf{p}_u \mathbf{Y}_u \mathbf{Y}_d = \mathbf{p}_u$, so $c_u \mathbf{p}_u = \boldsymbol{\pi}_u \mathbf{B}_u$, or

$$\boldsymbol{\pi}_u = c_u \mathbf{p}_u \mathbf{V}_u, \quad \text{and} \quad \boldsymbol{\pi}_d = c_d \mathbf{p}_u \mathbf{Y}_d \mathbf{V}_d = c_d \mathbf{p}_d \mathbf{V}_d,$$

where c_u and c_d are normalization constants. The steady-state times, then, are (no subscript)

$$MTTF := \lim_{n \rightarrow \infty} \mathbf{E}[U_n] = \mathbf{p}_u \mathbf{V}_u \boldsymbol{\epsilon}'_u, \quad \text{and} \quad MTTR = \mathbf{p}_d \mathbf{V}_d \boldsymbol{\epsilon}'_d.$$

In fact, their distributions are given by $h_{U_\infty}(x)$ and $h_{R_\infty}(t)$, as given above.

We have actually already derived the pdf's for the n th failure times and repair times, namely $h_{U_n}(x)$ and $h_{R_n}(t)$. In particular, we can write down the mean times for each epoch,

$$MTTF_n = \mathbf{p}_0 (\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \mathbf{V}_u \boldsymbol{\epsilon}'_u,$$

and

$$MTTR_n = \mathbf{p}_0 (\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \mathbf{Y}_u \mathbf{V}_d \boldsymbol{\epsilon}'_d.$$

In many examples, \mathbf{Y}_u and \mathbf{Y}_d are of rank 1, in which case, all the $MTTF$'s are equal.

4.2 Availability

In the systems where parallel processing goes on, it may be important to know how many processors are available (i.e., functional) at any time. Let a_i be the number of processors available when the system is in state $i \in E_u$, and define the diagonal matrix, $[\mathbf{A}]_{ii} = a_i$. Let N_n be the random variable denoting the expected number of

processors that are available during the n th UP interval, then

$$\mathbb{E}[N_n] = \frac{\mathbf{p}_o(\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \mathbf{V}_u \mathbf{A} \boldsymbol{\varepsilon}'_u}{\mathbf{p}_o(\mathbf{Y}_u \mathbf{Y}_d)^{n-1} \mathbf{V}_u \boldsymbol{\varepsilon}'_u}.$$

Its range is $0 < \mathbb{E}[N_n] \leq \max[a_i]$. If one is dealing with servers with different service rates, then a_i is the sum of the service rates of the servers that are functional when the system is in state $i \in E_u$. Then $\mathbb{E}[N_n]$ is the expected processing power available during the n th UP cycle. In this paper, we assume that the base service rate is 1.0.

5 Application to RESTART

The previous sections describe infinite looping. Now, consider a system that must execute a job that takes a time denoted by the random variable, L . For now, assume that the job takes a fixed time, ℓ (i.e., $L = \ell$). Under *RESTART* the process ends whenever the time spent in E_u exceeds ℓ . This is a *Transient Markov Renewal Process* (the process must end eventually). The equations above must be modified to accommodate this.

5.1 Time in E_u Cannot Exceed ℓ

All the equations up to (5) do not involve ℓ , but now $\bar{H}_{U_1}(t)$ must be modified to

$$\bar{H}_{U_1}(t) = \begin{cases} \mathbf{p}_o \bar{\mathbf{G}}_u(t) \boldsymbol{\varepsilon}'_u & \text{for } t \leq \ell \\ 0 & \text{for } t > \ell \end{cases}.$$

Clearly, if the system would have failed in a time $t > \ell$ then the process ends with the job completion. In order to move on, we must have $t < \ell$. The conditional probability distribution for failure to occur by time $t \leq \ell$ is

$$H_{U_1}(t | \ell) := \Pr[U_1 \leq t | U_1 \leq \ell] = \frac{\mathbf{p}_o \mathbf{G}_u(t) \boldsymbol{\varepsilon}'_u}{\mathbf{p}_o \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u} = \frac{H_{U_1}(t)}{H_{U_1}(\ell)}.$$

Clearly, in order to reach the n th repair cycle, the system has to fail n times first. Equation(5) is modified to

$$h_{U_1 R_1}(t, x | \ell) = \frac{\mathbf{p}_o \bar{\mathbf{G}}_u(t) \mathbf{B}_u \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d}{\mathbf{p}_o \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u},$$

from which it follows [using $\int_0^\ell \exp(t\mathbf{B}_u)\mathbf{B}_u dt = \mathbf{I}_u - \exp(\ell\mathbf{B}_u) = \mathbf{G}_u(\ell)$],

$$h_{R_1}(x | \ell) = \frac{\mathbf{p}_0 \mathbf{G}_u(\ell) \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d}{\mathbf{p}_0 \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u} = \frac{\mathbf{p}_0 \mathbf{Y}_u(\ell)}{\mathbf{p}_0 \mathbf{Y}_u(\ell) \boldsymbol{\varepsilon}'_d} \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d,$$

where we have defined

$$\mathbf{Y}_u(\ell) := \mathbf{G}_u(\ell) \mathbf{Y}_u.$$

Then, in general,

$$h_{U_n}(t | \ell) = \frac{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \bar{\mathbf{G}}_u(t) \mathbf{B}_u \boldsymbol{\varepsilon}'_u}{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u} \quad (8)$$

and

$$h_{R_n}(x | \ell) = \frac{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{Y}_u(\ell) \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d}{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u}. \quad (9)$$

These are the density functions for U_n and R_n , given that at least n failures occur, and the denominators are the probabilities that they occur. The conditional densities for two consecutive events are now [compare with the equations following (5)]

$$h_{U_n R_n}(t, x | \ell) = \frac{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \bar{\mathbf{G}}_u(t) \mathbf{B}_u \mathbf{Y}_u \bar{\mathbf{G}}_d(x) \mathbf{B}_d \boldsymbol{\varepsilon}'_d}{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u}$$

and

$$h_{R_n U_{n+1}}(x, t | \ell) = \frac{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{Y}_u(\ell) \bar{\mathbf{G}}_d(x) \mathbf{B}_d \mathbf{Y}_d \bar{\mathbf{G}}_u(t) \mathbf{B}_u \boldsymbol{\varepsilon}'_d}{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^n \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u}.$$

In all these equations, we see that the n th repair interval begins with the initial probability vector:

$$\mathbf{p}_{R_n} := \frac{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{Y}_u(\ell)}{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{Y}_u(\ell) \boldsymbol{\varepsilon}'_u}. \quad (10)$$

Similarly the n th UP period begins with:

$$\mathbf{p}_{U_n} := \frac{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1}}{\mathbf{p}_0 [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \boldsymbol{\varepsilon}'_u}. \quad (11)$$

Note that the function, $X = \mathbf{p}_{U_n} \bar{\mathbf{G}}_u(t) \mathbf{B}_u \mathbf{Y}_u \boldsymbol{\varepsilon}'_u$, is defective (the system can't fail if $t > \ell$), so to satisfy the conditional probability that it will fail, the function must be divided by the probability of failure (i.e., be normalized to 1). That is, divide by $\int_0^\ell X dt = \mathbf{p}_{U_n} \mathbf{Y}_u(\ell) \boldsymbol{\varepsilon}'_u$. This yields the expressions given.

Observe that for any invertible matrix, \mathbf{B} with inverse, \mathbf{V} ,

$$\int_0^\ell t \exp(-t \mathbf{B}) \mathbf{B} dt = [\mathbf{I} - (\mathbf{I} + \ell \mathbf{B}) \exp(-\ell \mathbf{B})] \mathbf{V}.$$

We use this to find expressions for mean times for U_n and R_n , which are

$$\mathbb{E}[U_n] = \int_0^\ell t h_{U_n}(t|\ell) dt = \frac{\mathbf{p}_o [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \{ \mathbf{I}_u - (\mathbf{I}_u + \ell \mathbf{B}_u) \bar{\mathbf{G}}_u(\ell) \} \mathbf{V}_u \boldsymbol{\epsilon}'_u}{\mathbf{p}_o [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{G}_u(\ell) \boldsymbol{\epsilon}'_u},$$

and

$$\mathbb{E}[R_n] = \int_0^\infty x h_{R_n}(x|\ell) dx = \frac{\mathbf{p}_o [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{Y}_u(\ell) \mathbf{V}_d \boldsymbol{\epsilon}'_d}{\mathbf{p}_o [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{G}_u(\ell) \boldsymbol{\epsilon}'_u}.$$

Alternatively, their sum can be calculated with

$$\mathbb{E}[U_n + R_n] = \int_0^\ell \left[\int_0^\infty (t+x) h_{U_n R_n}(t, x|\ell) dx \right] dt.$$

This yields the sum of the two previous equations.

For any square matrix, \mathbf{Z} ,

$$\lim_{n \rightarrow \infty} \mathbf{Z}^n = \sigma^n \mathbf{v}' \mathbf{u},$$

where σ is the largest eigenvalue in magnitude³ of \mathbf{Z} (i.e., $\text{spr}[\mathbf{Z}] = \sigma$)⁴ and \mathbf{u} , \mathbf{v}' are the corresponding left and right eigenvectors ($\mathbf{u}\mathbf{Z} = \sigma \mathbf{u}$ and $\mathbf{Z}\mathbf{v}' = \sigma \mathbf{v}'$). Now, let us look at (10) and (11), where $[\mathbf{Y}_u(\ell) \mathbf{Y}_d]^n$ is replaced by $\sigma^n \mathbf{v}' \mathbf{u}$ (σ , \mathbf{u} and \mathbf{v}' depend on ℓ).

$$\mathbf{p}_R(\ell) := \lim_{n \rightarrow \infty} \mathbf{p}_{R_n} = \lim_{n \rightarrow \infty} \frac{\sigma^{n-1} \mathbf{p}_o \mathbf{v}' \mathbf{u} \mathbf{Y}_u(\ell)}{\sigma^{n-1} \mathbf{p}_o \mathbf{v}' \mathbf{u} \mathbf{G}_u(\ell) \boldsymbol{\epsilon}'_u} = \frac{\mathbf{u} \mathbf{Y}_u(\ell)}{\mathbf{u} \mathbf{G}_u(\ell) \boldsymbol{\epsilon}'_u},$$

and

$$\mathbf{p}_U(\ell) := \lim_{n \rightarrow \infty} \mathbf{p}_{U_n} = \lim_{n \rightarrow \infty} \frac{\sigma^{n-1} \mathbf{p}_o \mathbf{v}' \mathbf{u}}{\sigma^{n-1} \mathbf{p}_o \mathbf{v}' \mathbf{u} \boldsymbol{\epsilon}'_u} = \frac{\mathbf{u}}{\mathbf{u} \boldsymbol{\epsilon}'_u}. \tag{12}$$

So, $\mathbf{p}_U(\ell) \mathbf{Y}_u(\ell) \mathbf{Y}_d = \sigma(\ell) \mathbf{p}_U(\ell)$. Observe that \mathbf{u} , \mathbf{v}' , \mathbf{p}_R and \mathbf{p}_U all depend on ℓ . Also, \mathbf{p}_U is not the same as \mathbf{p}_u defined in (7), but

$$\lim_{\ell \rightarrow \infty} \mathbf{p}_U(\ell) = \mathbf{p}_u.$$

³Strictly speaking, this is only true if σ is unique. If it is a multiple root then the formula is more complicated.

⁴ $\text{spr}[\mathbf{X}] = \text{Spectral Radius} :=$ Largest eigenvalue in magnitude of matrix \mathbf{X} .

Also note that σ has a physical interpretation.

$$\sigma(\ell) = \mathbf{p}_U(\ell) \mathbf{Y}_u(\ell) \mathbf{Y}_d \boldsymbol{\varepsilon}'_u.$$

Since

$$\mathbf{Y}_u(\ell) \mathbf{Y}_d \boldsymbol{\varepsilon}'_u = \mathbf{G}_u(\ell) \mathbf{Y}_u \mathbf{Y}_d \boldsymbol{\varepsilon}'_u = \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u,$$

it follows that

$$\sigma = \mathbf{p}_U(\ell) \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u. \tag{13}$$

which is the probability that the system will fail at least one more time when n is large. It follows from (4) that as ℓ becomes large,

$$1 - \sigma = \mathbf{p}_U(\ell) \tilde{\mathbf{G}}_u(\ell) \boldsymbol{\varepsilon}'_u \sim q(\ell \beta_m) e^{-\ell \beta_m}$$

where q is a polynomial of degree one less than the multiplicity of β_m .

5.2 Mean Time Under RESTART

To get the mean total time for the process to finally finish, we must calculate

$$\mathbf{E}[X_r(\ell)] = \sum_{n=1}^{\infty} \Pr[N = n] \left[\sum_{j=1}^n \mathbf{E}[U_j + R_j] \right] + \ell,$$

where N is the rv for the number of failures before success. The sums can be interchanged using (for any x_{nj})

$$\sum_{n=1}^{\infty} \sum_{j=1}^n x_{nj} = \sum_{j=1}^{\infty} \sum_{n=j}^{\infty} x_{nj},$$

so this can be written as (after changing $j \rightarrow n$)

$$\mathbf{E}[X_r(\ell)] = \sum_{n=1}^{\infty} \Pr[N \geq n] \mathbf{E}[U_n + R_n] + \ell.$$

But $\Pr[N \geq n] = \mathbf{p}_o [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u$, so

$$\mathbf{E}[X_r(\ell)] = \sum_{n=1}^{\infty} \mathbf{p}_o [\mathbf{Y}_u(\ell) \mathbf{Y}_d]^{n-1} \{ [\mathbf{I}_u - (\mathbf{I}_u + \ell \mathbf{B}_u) \tilde{\mathbf{G}}_u(\ell)] \mathbf{V}_u \mathbf{Y}_u + \mathbf{Y}_u(\ell) \mathbf{V}_d \} \boldsymbol{\varepsilon}'_d + \ell.$$

The formula for the geometric sum lets us write

$$\sum_{n=1}^{\infty} [\mathbf{Y}_u(\ell)\mathbf{Y}_d]^{n-1} = [\mathbf{I}_u - \mathbf{Y}_u(\ell)\mathbf{Y}_d]^{-1},$$

so (after some manipulation) we get the following formula, which we state as a theorem.

Theorem 3 *Let $X_r(\ell)$ be the total time required (including repair time) under RESTART for a job of length ℓ to complete, then*

$$\mathbb{E}[X_r(\ell)] = \mathbf{p}_o[\mathbf{I}_u - \mathbf{Y}_u(\ell)\mathbf{Y}_d]^{-1}\mathbf{G}_u(\ell)[\mathbf{V}_u + \mathbf{Y}_u\mathbf{V}_d\mathbf{Y}_d]\boldsymbol{\varepsilon}'_u, \tag{14}$$

It also follows that

$$\mathbb{E}[N(\ell)] = \mathbf{p}_o[\mathbf{I}_u - \mathbf{Y}_u(\ell)\mathbf{Y}_d]^{-1}\mathbf{G}_u(\ell)\boldsymbol{\varepsilon}'_u,$$

where $N(\ell)$ is the number of failures before completion. This holds for any system that can be described by a \mathbf{Q} matrix as described by (3).

For the simplest case (a single server with exponential failure rate $\mathbf{B}_u = \beta$, and a single repairman with repair rate $\mathbf{B}_d = \lambda$, in which case $\mathbf{Y}_u = \mathbf{Y}_d = \boldsymbol{\varepsilon}'_u = \boldsymbol{\varepsilon}'_d = \mathbf{I}_u = 1$.) this reduces to the well-known formula

$$\mathbb{E}[X_r(\ell)] = (e^{\beta\ell} - 1) \left[\frac{1}{\beta} + \frac{1}{\lambda} \right]. \tag{15}$$

□

The above equation is shown in Fig. 1 for various values of λ . Although this is the simplest of systems, it is typical of all RESTART systems. As with all such systems, $\mathbb{E}[X_r(\ell)] = \ell$ when $\beta = 0$. Here, the slope at that point is $(\ell^2/2 + \ell/\lambda)$. But for more complicated systems (e.g., those with redundant servers) the slope can even be 0. But as the failure probability increases, $\mathbb{E}[X_r(\ell)]$ grows unboundedly large at a rate that is at least exponential (in this case, as $e^{\beta\ell}$). In fact, as shown by [1, 2], when averaged over all ℓ (see Sect. 5.3) it can become infinite even though the failure probability is finite. So, these models can be most useful in separating those failure rates for which $\{\mathbb{E}[X_r(\ell)] - \ell\}$ is acceptable from those for which $\{\mathbb{E}[X_r(\ell)] - \ell\}$ is so large that the mode of operation must be changed. More examples are given in Sect. 6.

5.2.1 Mean Residual Times

Often it is of interest to estimate how long it will take for a job to complete, given that it has already been running for a time, x . Formally, the mean time for this is

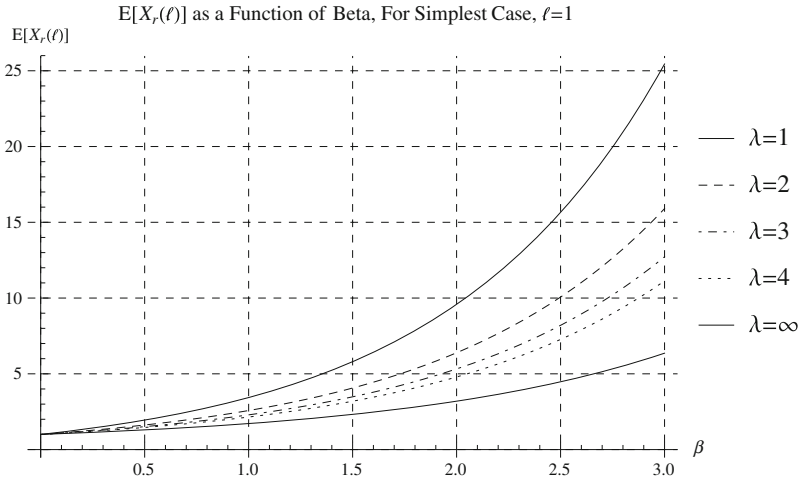


Fig. 1 $\mathbb{E}[X_r(\ell)]$ as a function of β , from Eq. (15), For various values of λ , and $\ell = 1$. This figure is typical of the performance of all systems under *RESTART*. When $\beta = 0$, $\mathbb{E}[X_r(\ell)] = \ell$. As β increases, the curves grow exponentially

$$\mathbb{E}[X_r(\ell | x)] := \frac{\int_0^\infty t h(t + x | \ell) dt}{\bar{H}(x | \ell)},$$

where $h(x | \ell)$ is the pdf for the time for a job of length ℓ to complete under *RESTART*. This is not possible to compute since we do not know what $h(x | \ell)$ is, but we can get a good idea if we know how many failures have already occurred. Then, the vector for starting another cycle is \mathbf{p}_{U_n} given by (11). Let $X_r(\ell | n)$ be the total time remaining at the start of the n^{th} cycle. Then, from (14)

$$\mathbb{E}[X_r(\ell | n)] = \mathbf{p}_{U_n} [\mathbf{I}_u - \mathbf{Y}_u(\ell) \mathbf{Y}_d]^{-1} \mathbf{G}_u(\ell) [\mathbf{V}_u + \mathbf{Y}_u \mathbf{V}_d \mathbf{Y}_d] \mathbf{e}'_u.$$

As $n \rightarrow \infty$ replace \mathbf{p}_{U_n} with \mathbf{p}_U from (12) and get

$$\mathbb{E}[X_r(\ell | \infty)] = \mathbf{p}_U [\mathbf{I}_u - \mathbf{Y}_u(\ell) \mathbf{Y}_d]^{-1} \mathbf{G}_u(\ell) [\mathbf{V}_u + \mathbf{Y}_u \mathbf{V}_d \mathbf{Y}_d] \mathbf{e}'_u.$$

But, from (12) $\mathbf{p}_U(\ell)$ is a left eigenvector of $\mathbf{Y}_u(\ell) \mathbf{Y}_d$ with eigenvalue σ , so

$$\mathbb{E}[X_r(\ell | \infty)] = \frac{1}{1 - \sigma} \mathbf{p}_U(\ell) \mathbf{G}_u(\ell) [\mathbf{V}_u + \mathbf{Y}_u \mathbf{V}_d \mathbf{Y}_d] \mathbf{e}'_u.$$

From (13), it follows that $(1 - \sigma) = \mathbf{p}_U(\ell) \exp(-\ell \mathbf{B}_u) \mathbf{e}'_u$, so

$$\mathbb{E}[X_r(\ell | \infty)] = \frac{1}{\mathbf{p}_U(\ell) \exp(-\ell \mathbf{B}_u) \mathbf{e}'_u} \mathbf{p}_U(\ell) \mathbf{G}_u(\ell) [\mathbf{V}_u + \mathbf{Y}_u \mathbf{V}_d \mathbf{Y}_d] \mathbf{e}'_u. \tag{16}$$

[noting that $\mathbf{G}_u(\ell) \rightarrow \mathbf{I}_u - e^{-\ell\beta_m}\mathbf{D}(\beta_m\ell)$, and that β_m is the minimum eigenvalue of \mathbf{B}_u]⁵

$$\mathbb{E}[X_r(\ell | \infty)] \sim \frac{e^{\ell\beta_m}}{p(\ell\beta_m)} [\mathbf{p}_u(\mathbf{V}_u + \mathbf{Y}_u\mathbf{V}_d\mathbf{Y}_d)\boldsymbol{\varepsilon}'_u] - O(1). \tag{17}$$

A simple coupling argument, that we omit, shows that this is also true for $\mathbb{E}[X_r(\ell)]$.

5.3 Job Time Is Random

Instead of considering a job requiring a fixed time ℓ , in this section, we consider the performance of a large set of jobs taken from some distribution, $F(t)$, with r.v. L . Thus, $F(t) = \Pr[L \leq t]$. The average over all the samples is given by

$$\mathbb{E}[X_r] := \int_0^\infty \mathbb{E}[X_r(t)]f(t) dt. \tag{18}$$

It has been shown in (17) that $\mathbb{E}[X_r(t)]$ goes to ∞ as $\exp(t\beta_m)$. Therefore $f(t)$ must go to 0 faster than $\exp(-t\beta_m)$ in order for the integral to be finite. From (1) μ_m must be greater than β_m . If $\mu_m = \infty$ (which includes functions that are 0 above some finite value of t) then there is no problem, and if $\mu_m = 0$ ($f(t)$ is *sub-exponential*) there is no hope. As stated previously, if $0 < \mu_m < \infty$ ($f(t)$ has an exponential tail) then $H_{X_r}(x)$ (the distribution for the total time of the *RESTART* process) is *power-tailed* with parameter $\alpha = \mu_m/\beta_m$. That is,

$$\lim_{x \rightarrow \infty} x^\rho \bar{H}_{X_r}(x) = \begin{cases} 0 & \text{for } \rho < \alpha \\ \infty & \text{for } \rho > \alpha \end{cases}.$$

Therefore, it follows that $\mathbb{E}[X_r(t)] = \infty$ if $\alpha \leq 1$, that is, if $\mu_m \leq \beta_m$. In fact, if $2\mu_m \leq \beta_m$ then $\mathbb{E}[X_r^2(t)] = \infty$, implying that the variance of $H_{X_r}(x)$ is infinite. What implication does this have for performance? Suppose that the failure distribution (as represented by \mathbf{B}_u) depends on a single parameter, β , in such a way that β_m increases monotonically (not necessarily linearly) with β . Then a performance curve (as exemplified by Fig. 1) will become infinite at that point where $\beta_m(\beta) = \mu_m$. Furthermore, one can expect the system to become unstable when $\beta_m(\beta) \geq \mu_m/2$, for then X_r has infinite variance.

There is a problem evaluating the integral in Eq. (18) because the expression in (14) is not in general conducive to analytic integration. Therefore, numerical procedures must be used. This is discussed in Sect. 6.5 with a numerical example as shown in Fig. 3.

⁵ $\mathbf{D}(\ell\beta_m)$ is a matrix whose components are polynomials of degree less than or equal to the degree of $p(\ell\beta_m)$. Therefore, $\mathbf{D}(\ell\beta_m)/p(\ell\beta_m) \sim O(1)$.

6 Some Examples

We set up the Q matrix for various systems, including redundant servers, which thereby yield non-exponential failure distributions, and variations in how the system is restarted. We also consider non-exponential repair time distributions.

6.1 Simplest Case: Single Exponential Server/Repair

Here, we have a simple exponential server with failure rate, β , and a single exponential repair stage with rate λ . The Q matrix is

$$-Q = \begin{bmatrix} \beta & | & -\beta \\ \hline -\lambda & | & \lambda \end{bmatrix}.$$

The various matrices and vectors trivialize to

$$Y_u = Y_d = p_u = p_d = \epsilon'_u = \epsilon'_d = I_u = I_d = 1.$$

Also, $B_u = \beta$, $B_d = \lambda$ and Eq. (14) reduces to (15) in Theorem 3, and is plotted in Fig. 1 in Sect. 5.2.

6.2 Two Exponential Servers/'r' Exponential Repair Servers

Here, we have two structurally different systems. In the first, only one of the servers performs useful work. The other either: ($b = 1$) sits idle, and starts up when the other fails (*cold backup*); or ($b = 2$) duplicates the first one's work, and takes over if it fails (*hot backup*). In the latter case the backup server can fail while waiting. It could also happen that the process of bringing up the backup server fails with probability, p . We will assume this is the case only for cold backup.

We also allow for *on-line* repair to occur, with the same distribution as *off-line* repair (system is down). When both servers are down, they can be repaired in parallel ($r = 2$) or one at a time ($r = 1$).

We allow for one other variation. When the system is DOWN (both servers have failed and are being repaired), it can restart when: (1) either one is repaired, or, (2) must wait for both to be repaired.

In all cases, we assume that failures for each server are exponentially distributed with rate β . The mean time to repair a given server is $1/\lambda$. We consider non-exponential repair distributions in Sect. 6.3

6.2.1 RESTART Begins Immediately When One Is Repaired

The system is DOWN only when both servers are down, and when one of them is repaired the system shifts to the UP state. There are two up states (both up, one up and one down), and one down state (both servers down). For notation, we use

$$\begin{aligned} \uparrow &:= \text{Server is functional} \\ \downarrow &:= \text{Server is down} \end{aligned}$$

The states are

$$E_u = \{\uparrow\uparrow, \uparrow\downarrow\}$$

$$E_d = \{\downarrow\downarrow\}$$

The Q matrix follows:

$$-Q = \left[\begin{array}{cc|c} b\beta & -(1-p)b\beta & -pb\beta \\ -\lambda & \lambda + \beta & -\beta \\ \hline 0 & -r\lambda & r\lambda \end{array} \right].$$

The matrices, B_u and B_d are obvious.

$$Y_d = p_u = [0 \ 1], \quad Y_u = \varepsilon'_u = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \text{and} \quad p_d = \varepsilon'_d = [1].$$

These arrays can be inserted into (14). One can either use matrix operators directly, or find the analytic expression for $\exp(-\ell B_u)$ since it is a two-dimension matrix. For higher dimensional matrices, it is probably best to stay with the matrix form, since MATLAB (and other software) calculate matrices efficiently and easily.

The results of such a calculation are shown in Fig. 2 where $\mathbf{E}[X_r(\ell)]$ is plotted as a function of β for four different system configurations. This set of curves is typical for the systems presented here. It is not our goal to give detailed performance analyses of these systems, but only to show what might be studied using the equations derived. Nonetheless, certain features are general. For example, if $p = 0$ (perfect transition to the backup server), the slope of the curves at $\beta = 0$ is 0. The value of $p = 0.1$ was assigned to cold backup ($b = 1$) because such systems are less likely to transition smoothly when the primary server fails. Thus their slopes at $\beta = 0$ are positive. But, as β increases each curve crosses the corresponding curve with the same value of r , reflecting the ultimate cost of hot backup. After all, the hot backup server can fail even while waiting. And as might be expected, the systems with two repairmen outperform the ones with one repairman. Of course this does not include the monetary cost of having a second repairman constantly available. Ultimately all of the curves grow exponentially as $\exp(\ell\beta_m)$, where β_m is the smallest eigenvalue of B_u , which is always positive and real (Note that β_m is different for each system.).

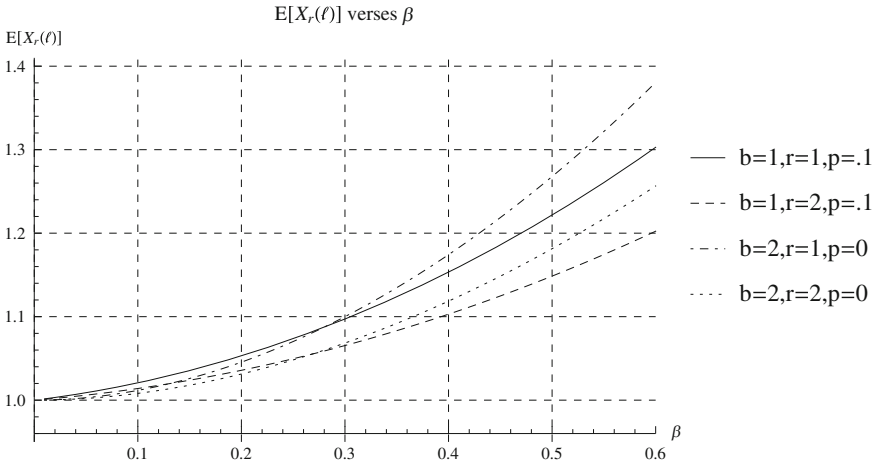


Fig. 2 $\mathbf{E}[X_r(\ell = 1)]$ as a function of β for both cold and hot backup, and both 1 and 2 repairmen with repair rate $\lambda = 1$. $p = 0.1$ for cold backup. In this example, the system restarts as soon as one of the servers is repaired

6.2.2 RESTART Begins only When both Servers Are Functional

Here, there are again two up states, when both servers are up, and when one is up and one is down, given that it got there from the 2 up state. There are now 2 down states, when both are down and when one is down and the other is up, given that it got there from the 2 down state. The states are $E_u = \{\uparrow\uparrow, \uparrow\downarrow\}$ and $E_d = \{\downarrow\downarrow, \downarrow\uparrow\}$. The \mathbf{Q} matrix is now

$$-\mathbf{Q} = \left[\begin{array}{cc|cc} b\beta & -(1-p)b\beta & -pb\beta & 0 \\ -\lambda & \lambda + \beta & -\beta & 0 \\ \hline 0 & 0 & r\lambda & -r\lambda \\ -\lambda & 0 & 0 & \lambda \end{array} \right].$$

Again, \mathbf{B}_u and \mathbf{B}_d are clear, and

$$\mathbf{Y}_d = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} [1 \ 0] = \mathbf{e}'_d \mathbf{p}_u = \mathbf{Y}_u = \mathbf{e}'_u \mathbf{p}_d.$$

Just like the previous examples this system has the property that both \mathbf{Y}_u and \mathbf{Y}_d are of rank 1. Note that in this case $\mathbf{p}_u = \mathbf{p}_o$ (the system always starts or restarts with both servers functional), so all the intervals are iid, including U_1 .

6.3 Non-exponential Repair Times

In this section, we present examples of systems where at least one of the \mathbf{Y} 's is of rank greater than 1. We will do this by requiring that the repair is matrix exponential. This is rather easy to do for those nodes where only one server is being repaired. But when two or more devices are active (e.g., hot backup or when one is up and one is down during the UP time), then the node must be represented by a product space of the two (or more) distributions. If the two devices are identical (during hot backup, or two repairmen) the problem is somewhat simplified by using a so-called *reduced product space*. There may be other complications, however, that require individual treatment.

6.3.1 Erlangian-2 Repair Times

We can probably find a general formulation, but for this example, we assume that the repair time is Erlangian-2, i.e.,

$$\bar{R}(t) = (1 + 2\lambda t)e^{-2\lambda t},$$

where the mean repair time is the same as before for a single server, namely, $1/\lambda$. We examine two cases: (1) *RESTART* only begins when both servers are repaired; (2) *RESTART* begins as soon as one server is repaired.

(a) Both Must Be Repaired Before System Can RESTART

We must have eight states to describe this system. For notation, we use

- \uparrow := Server is functional
- \downarrow_1 := Server is down and in phase 1 of repair
- \downarrow_2 := Server is down and in phase 2 of repair.

Then, the set of UP states is

$$E_u = \{\uparrow\uparrow, \uparrow\downarrow_1, \uparrow\downarrow_2\}.$$

The set of DOWN states is:

$$E_d = \{\downarrow_1\downarrow_1, \downarrow_1\downarrow_2, \downarrow_2\downarrow_2, \downarrow_1\uparrow, \downarrow_2\uparrow\}.$$

The states $\{\uparrow\downarrow_1, \uparrow\downarrow_2\}$ seem to occur twice, once in E_u and once in E_d . They are actually different states, since once the system fails it must wait until *both* servers are repaired before returning to the UP state. We assume that the repaired machine cannot fail while in a DOWN state, but can fail while in an UP state, but only in *hot backup* ($b = 2$).

The \mathbf{Q} matrix, (the states are ordered as listed in E_u and then E_d),

$$-\mathbf{Q} = \left[\begin{array}{ccc|ccc} b\beta & -(1-p)b\beta & 0 & -pb\beta & 0 & 0 & 0 & 0 \\ 0 & 2\lambda + \beta & -2\lambda & -\beta & 0 & 0 & 0 & 0 \\ -2\lambda & 0 & 2\lambda + \beta & 0 & -\beta & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 2r\lambda & -2r\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2r\lambda & -2(r-1)\lambda & -2\lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 2r\lambda & 0 & -2r\lambda \\ 0 & 0 & 0 & 0 & 0 & 0 & 2\lambda & -2\lambda \\ -2\lambda & 0 & 0 & 0 & 0 & 0 & 0 & 2\lambda \end{array} \right]$$

Remember, 2λ is the rate at which a single stage is repaired, $r \in \{1, 2\}$ is the number of repairmen available, and $b \in \{1, 2\}$ is cold, or hot backup.

Note the factor $(r - 1)$ in the matrix. In leaving the state $\{\downarrow_1 \downarrow_2\}$, two things can happen. Either the machine in its first phase of repair goes to the second phase, or the other machine is repaired. If there are two repairmen ($r = 2$), then the two events are equally likely. But if there is only one repairman only one of the two events can happen. Here, we assume that the machine that is already in its second phase will be the one to finish, since the repairman was already working on it when the other failed. Other assumptions are possible, and more complicated systems can introduce ever more complex scenarios. The important point to note is that the overall structure of the \mathbf{Q} matrix remains the same, and all the formal conclusions we have derived are valid.

It is not hard to find

$$\mathbf{V}_u = \mathbf{B}_u^{-1} = \frac{1}{b\beta(4\lambda\beta + \beta^2 + 4p\lambda^2)} \begin{bmatrix} (2\lambda + \beta)^2 & (1-p)b\beta(2\lambda + \beta) & 2(1-p)b\lambda\beta \\ (2\lambda)^2 & b\beta(2\lambda + \beta) & 2b\lambda\beta \\ 2\lambda(2\lambda + \beta) & 2(1-p)b\lambda\beta & b\beta(2\lambda + \beta) \end{bmatrix}.$$

From this, it follows that

$$\mathbf{Y}_u = \frac{1}{4\lambda\beta + \beta^2 + 4p\lambda^2} \begin{bmatrix} (2\lambda + \beta)(2p\lambda + \beta) & 2(1-p)\lambda\beta & 0 & 0 & 0 \\ \beta(2\lambda + \beta) + 4p\lambda^2 & 2\lambda\beta & 0 & 0 & 0 \\ 2\lambda(2p\lambda + \beta) & (2\lambda + \beta)\beta & 0 & 0 & 0 \end{bmatrix}.$$

(Yes, $\mathbf{Y}_u \boldsymbol{\varepsilon}'_d = \boldsymbol{\varepsilon}'_u$.) This is clearly *not* a rank 1 matrix because there are two independent rows. But, the \mathbf{Q}_{du} block has only one nonzero element, so it follows that

$$\mathbf{Y}_d = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} [1 \ 0 \ 0] = \boldsymbol{\varepsilon}'_d \mathbf{p}_1.$$

Therefore, \mathbf{Y}_d IS of rank 1, so the U_n 's, $n > 1$, are iid.

As an aside, observe that

$$\mathbf{Y}_u \mathbf{Y}_d = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} [1 \ 0 \ 0] = \boldsymbol{\varepsilon}'_u \mathbf{p}_1.$$

(b) The System Restarts as Soon as One Machine Is Repaired

This system only has a total of six states, because the two states, $\{\uparrow\downarrow_1, \uparrow\downarrow_2\}$ only occur in E_u , which is the same as before, but now

$$E_u = \{\uparrow\uparrow, \uparrow\downarrow_1, \uparrow\downarrow_2\}, \quad E_d = \{\downarrow_1\downarrow_1, \downarrow_2\downarrow_1, \downarrow_2\downarrow_2\}.$$

The \mathbf{Q} matrix follows.

$$-\mathbf{Q} = \left[\begin{array}{ccc|ccc} b\beta & -(1-p)b\beta & 0 & -pb\beta & 0 & 0 \\ 0 & 2\lambda + \beta & -2\lambda & -\beta & 0 & 0 \\ -2\lambda & 0 & 2\lambda + \beta & 0 & -\beta & 0 \\ \hline 0 & 0 & 0 & 2r\lambda & -2r\lambda & 0 \\ 0 & -2\lambda & 0 & 0 & 2r\lambda & -2(r-1)\lambda \\ 0 & 0 & -2r\lambda & 0 & 0 & 2r\lambda \end{array} \right]$$

At last, we have an example where neither \mathbf{Y}_u nor \mathbf{Y}_d is of rank 1, even though \mathbf{B}_u is the same as in the previous case. Observe that $(\mathbf{B}_u \mathbf{Y}_u)$ is also the same, so \mathbf{Y}_u must also be the same after deleting the last two columns of zeroes.

It is not hard to calculate \mathbf{Y}_u and \mathbf{Y}_d . They are

$$\mathbf{Y}_u = \frac{1}{4\lambda\beta + \beta^2 + 4p\lambda^2} \begin{bmatrix} (2\lambda + \beta)(2p\lambda + \beta) & 2(1-p)\lambda\beta & 0 \\ \beta(2\lambda + \beta) + 4p\lambda^2 & 2\lambda\beta & 0 \\ 2\lambda(2p\lambda + \beta) & (2\lambda + \beta)\beta & 0 \end{bmatrix}, \quad \mathbf{Y}_d = \frac{1}{r} \begin{bmatrix} 0 & 1 & r-1 \\ 0 & 1 & r-1 \\ 0 & 0 & r \end{bmatrix}.$$

It is obvious that $\mathbf{Y}_d \boldsymbol{\varepsilon}'_u = \boldsymbol{\varepsilon}'_d$. Furthermore, observe that

$$\boldsymbol{\varepsilon}'_u \mathbf{p}_u := \mathbf{Y}_u \mathbf{Y}_d = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & \frac{1}{r} & \frac{(r-1)}{r} \end{bmatrix}.$$

So, even though both \mathbf{Y}_u and \mathbf{Y}_d are of rank 2, their product is of rank 1. This tells us that all $U_n + R_n$ for $n > 1$ are iid, and independent of $U_1 + R_1$. First, observe that

$$\mathbf{p}_o \mathbf{Y}_u(\ell) \mathbf{Y}_d = \mathbf{p}_o \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u \mathbf{p}_u = g_o(\ell) \mathbf{p}_u,$$

where $g_o(\ell) := \mathbf{p}_o \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u = \Pr(U_1 < \ell)$. Then, from (11), $\mathbf{p}_{U_2} = \mathbf{p}_u$. Next define $g_1(\ell) := \mathbf{p}_u \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u$, then

$$[\mathbf{Y}_u(\ell) \mathbf{Y}_d]^n = (\mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u \mathbf{p}_u)^n = g_1(\ell)^{n-1} \mathbf{G}_u(\ell) \boldsymbol{\varepsilon}'_u \mathbf{p}_u,$$

and

$$\mathbf{p}_o[\mathbf{Y}_u(\ell) \mathbf{Y}_d]^n \mathbf{e}'_u = g_o(\ell) g_1(\ell)^{n-1}.$$

The interpretation of this is simple. It is the probability that the system will fail at least n times.

Next recall that $\mathbf{p}_{U_n} = \mathbf{p}_{U_{n-1}} \mathbf{Y}_u(\ell) \mathbf{Y}_d$. This leads to [again from (11)]

$$\mathbf{p}_{U_n} = \frac{\mathbf{p}_o g_1^{n-2} \mathbf{G}_u \mathbf{e}'_u \mathbf{p}_u}{\mathbf{p}_o g_1^{n-2} \mathbf{G}_u \mathbf{e}'_u \mathbf{p}_u \mathbf{e}'_u} = \frac{g_o g_1^{n-2} \mathbf{p}_u}{g_o g_1^{n-2}} = \mathbf{p}_u.$$

In a similar manner, using all the formulas we have already written down, it follows that

$$\mathbf{p}_{R_n} = \frac{1}{g_1(\ell)} \mathbf{p}_u \mathbf{G}_u(\ell) \mathbf{Y}_u.$$

Thus, both \mathbf{p}_{U_n} and \mathbf{p}_{R_n} are independent of $n > 1$ (although \mathbf{p}_{R_n} depends upon ℓ).

6.3.2 Hyperexponential-2 Repair Time Distributions

When we consider more complex systems, the \mathbf{Q} matrix grows ever bigger. But so far we have gained little insight as to the internal structure beyond what we already have described. Next, we will look at the hyperexponential distribution for repair. But it becomes ever so tedious to find analytic expressions for the various matrices (e.g., \mathbf{Y}_u , \mathbf{Y}_d , \mathbf{V}_u , \mathbf{V}_d and g_1 and g_o). Numerical solutions will have to suffice.

The hyperexponential distribution with two components is defined as

$$\bar{R}_2(t) := p_1 e^{-\lambda_1 t} + p_2 e^{-\lambda_2 t},$$

where $p_1 + p_2 = 1$. Again let $b = 1$ be cold backup and $b = 2$ be hot backup, p is the probability that the second server will fail when the first server fails, and β is the nominal failure rate of each server.

(a) Both Servers Must Be up Before RESTART

This scenario requires three UP states and six failed states. Then

$$E_u = \{\uparrow\uparrow, \uparrow\downarrow_1, \uparrow\downarrow_2\}$$

where the notation should be obvious, and

$$\mathbf{B}_u = \begin{bmatrix} b\beta & -(1-p)p_1 b\beta & -(1-p)p_2 b\beta \\ -\lambda_1 & \beta + \lambda_1 & 0 \\ -\lambda_2 & 0 & \beta + \lambda_2 \end{bmatrix}.$$

The failed states are listed below, where we must keep track of which server failed first. If there are two repairmen, then it does not make a difference, but if there is only one repairman, then he must continue repairing the first one that failed when the second one fails.

$$E_d = \{\downarrow_1\downarrow_2, \downarrow_1\downarrow_1, \downarrow_2\downarrow_1, \downarrow_2\downarrow_2, \downarrow_1\uparrow, \downarrow_2\uparrow\}$$

Then,

$$\mathbf{B}_d = \begin{bmatrix} \lambda_1 + (r - 1)\lambda_2 & 0 & 0 & 0 & -(r - 1)\lambda_2 & -\lambda_1 \\ 0 & r\lambda_1 & 0 & 0 & -r\lambda_1 & 0 \\ 0 & 0 & \lambda_2 + (r - 1)\lambda_1 & 0 & -\lambda_2 & -(r - 1)\lambda_1 \\ 0 & 0 & 0 & r\lambda_2 & 0 & -r\lambda_2 \\ 0 & 0 & 0 & 0 & \lambda_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda_2 \end{bmatrix}.$$

$$\mathbf{Q}_{ud} = \begin{bmatrix} pb p_1 p_2 \beta & pb p_1^2 \beta & pb p_1 p_2 \beta & pb p_2^2 \beta & 0 & 0 \\ p_2 \beta & p_1 \beta & 0 & 0 & 0 & 0 \\ 0 & 0 & p_1 \beta & p_2 \beta & 0 & 0 \end{bmatrix}$$

and

$$\mathbf{Q}_{du} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \lambda_1 & 0 & 0 \\ \lambda_2 & 0 & 0 \end{bmatrix}.$$

It is not hard to show that $\mathbf{Y}_u \mathbf{Y}_d = \mathbf{e}'_u [100]$, so once again we have a *simple* system. As a last try, we look at H_2 with one server up.

(b) RESTART Begins After One Server Is Repaired

As in the previous case, $E_u = \{\uparrow\uparrow, \uparrow\downarrow_1, \uparrow\downarrow_2\}$, and \mathbf{B}_u is the same. \mathbf{B}_d is different, with

$$E_d = \{\downarrow_1\downarrow_2, \downarrow_1\downarrow_1, \downarrow_2\downarrow_1, \downarrow_2\downarrow_2\}.$$

Note that $\downarrow_1\downarrow_2$ and $\downarrow_2\downarrow_1$ are different states, denoting the state of server that failed first.

$$\mathbf{B}_d = \begin{bmatrix} \lambda_1 + (r - 1)\lambda_2 & 0 & 0 & 0 \\ 0 & r\lambda_1 & 0 & 0 \\ 0 & 0 & \lambda_2 + (r - 1)\lambda_1 & 0 \\ 0 & 0 & 0 & r\lambda_2 \end{bmatrix}.$$

$$\mathbf{Q}_{ud} = \begin{bmatrix} pb p_1 p_2 \beta & pb p_1^2 \beta & pb p_1 p_2 \beta & pb p_2^2 \beta \\ p_2 \beta & p_1 \beta & 0 & 0 \\ 0 & 0 & p_1 \beta & p_2 \beta \end{bmatrix}$$

and

$$\mathbf{Q}_{du} = \begin{bmatrix} 0 & (r-1)\lambda_2 & \lambda_1 \\ 0 & r\lambda_1 & 0 \\ 0 & \lambda_2 & (r-1)\lambda_1 \\ 0 & 0 & r\lambda_2 \end{bmatrix}.$$

It is not worth the effort to find an analytic expression for \mathbf{Y}_u and \mathbf{Y}_d , since they can be calculated easily numerically for specific values of the parameter. But we see that yet again $\mathbf{Y}_u \mathbf{Y}_d$ is of rank 1, being $\mathbf{Y}_u \mathbf{Y}_d = \boldsymbol{\varepsilon}'_d [0 \ p_1 \ p_2]$.

6.4 Model of a System Where $\mathbf{Y}_u \mathbf{Y}_d$ Has Full Rank

Here, we present a simple model that plainly shows how and why correlation can persist for many cycles. Consider a system with two servers which can process at the same rate, but fail at different exponential rates (β_i). Only one can run at a time, and if it fails the system goes to the Down states. The Down states consist of two repairmen who repair at different rates (λ_i). If coming from server i repairman j is chosen with probability p_{ij} (so $p_{i1} + p_{i2} = 1$). When the server is repaired, processing restarts, with server m chosen with probability q_{km} when coming from repairman k . The starting vector, \mathbf{p}_0 determines which server begins the process.

The \mathbf{Q} matrix is

$$\mathbf{Q} = \left[\begin{array}{cc|cc} -\beta_1 & 0 & p_{11}\beta_1 & p_{12}\beta_1 \\ 0 & -\beta_2 & p_{21}\beta_2 & p_{22}\beta_2 \\ \hline q_{11}\lambda_1 & q_{12}\lambda_1 & -\lambda_1 & 0 \\ q_{21}\lambda_2 & q_{22}\lambda_2 & 0 & -\lambda_2 \end{array} \right]$$

It easily follows (from $\mathbf{Y}_u = \mathbf{V}_u \mathbf{Q}_{ud}$, etc.) that

$$[\mathbf{Y}_u]_{ij} = p_{ij} \quad \text{and} \quad [\mathbf{Y}_d]_{km} = q_{km}.$$

U_n and R_n look like they have hyperexponential distributions, but the starting probabilities can be different for each epoch, therefore $\mathbb{E}[U_n]$ could be different for each n . However, if $q_{1j} = q_{2j}$ then the U_n 's are iid with the same hyperexponential distribution, and $\text{rank}[\mathbf{Y}_d] = 1$. The corresponding situation is true for R_n as well.

We have calculated some expectations using the following parameters: $\beta_1 = 1$, $\beta_2 = 3$, $\lambda_1 = 2$, $\lambda_2 = 4$, $\mathbf{p}_0 = [0.4, 0.6]$, $p_{11} = 0.7$, $p_{21} = 0.3$, $q_{11} = 0.2$, and $q_{21} = 0.9$. These were used in evaluating $\mathbb{E}[X_r(\ell)]$ from (14). \mathbf{p}_u , the left eigenvector of $\mathbf{Y}_u \mathbf{Y}_d$ was found to be $\mathbf{p}_u = [0.5390625, 0.4609375]$, and was used in (16) to calculate $\mathbb{E}[X_r(\ell|\infty)]$. The two expectations are close, differing by less than $0.125 \forall \ell$. We then used (17) to calculate the asymptotic values of $\mathbb{E}[X_r(\ell|\infty)]$. Since $\beta_1 = 1$ is the minimum eigenvalue of \mathbf{B}_u , the formula is of the form

$$\mathbb{E}[X_r(\ell|\infty)] \sim f(\ell) = c_1 e^{\ell * 1} + c_2$$

where $c_1 = \mathbf{p}_u(\mathbf{V}_u + \mathbf{Y}_u \mathbf{V}_d \mathbf{Y}_d) \boldsymbol{\epsilon}'_u / [\mathbf{p}_u]_1$. In our case, $c_1 = 1.7716145/0.5390625$. c_2 was found by fitting to the curve and found to be -1.55951229 . The relative difference was found to be

$$\left| \frac{\mathbb{E}[X_r(\ell|\infty)] - f(\ell)}{\mathbb{E}[X_r(\ell|\infty)]} \right| < 0.0001 \quad \text{for } \ell > 4.0$$

going to $<10^{-6}$ by $\ell > 5.5$.

6.5 An Example of Numerical Evaluation of $\mathbb{E}[\mathbf{X}_r]$

In Sect. 5.3, we discussed how to deal with random job times. That involved evaluating the integral in (18). Unfortunately except for the simplest of systems, $\mathbb{E}[X_r(\ell)]$ from (14) can only be solved numerically. That means (18) must be integrated numerically. This can be a problem if $f(t)$ has an exponential tail, because the portion of the integral from large t to ∞ can contribute significantly to the total. One method would be to perform a numerical integration up to some large t_m , and then replace $\mathbb{E}[X_r(t)]$ with its asymptotic approximation, $\exp(t\beta_m)/q_u(t\beta_m)$, and then perform the rest of the integral analytically. We have chosen the brute force method for our example here, letting *Mathematica* grind away to its heart content. It was computationally time consuming, and if one were to make a parametric study of several systems, the former technique would have to be employed. None-the-less, the results presented in Fig. 3 were quite satisfactory for our purpose (Fig. 4).

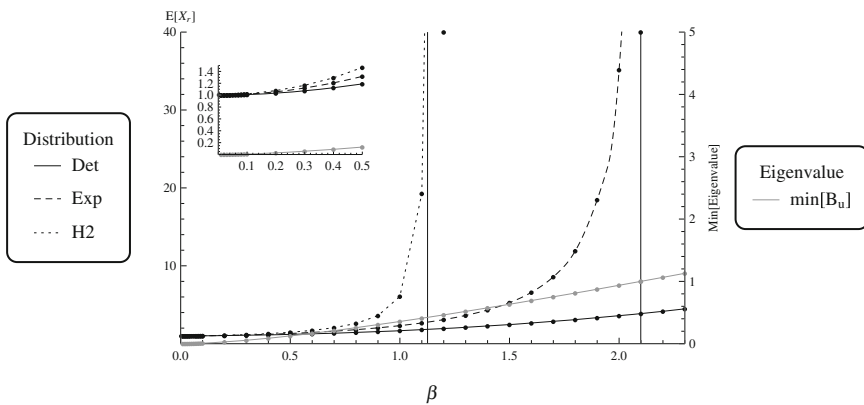


Fig. 3 Mean total time for system described in Sect. 6.3.2b. $\mathbb{E}[L] = 1$, with three different task time distributions. *Det* Deterministic; *Exp* Exponential; *H2* 2 phase Hyperexponential. Also included is β_m . All four curves vary with β , the failure rate for a single server. The figure is discussed fully in the text

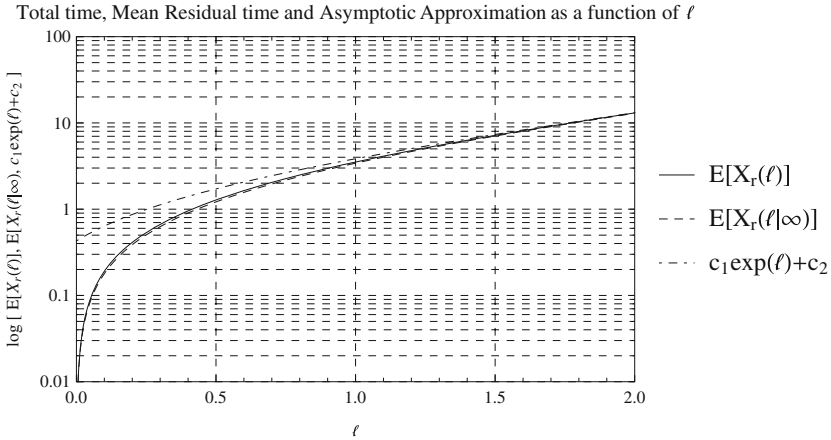


Fig. 4 $\mathbb{E}[X_r(\ell)]$, $\mathbb{E}[X_r(\ell|\infty)]$ and the asymptotic approximation of the model given in Sect. 6.4, as a function of ℓ . The formulas come from (14), (16), and (17)

The example chosen is taken from the previous section, part (2). That is, $\mathbb{E}[L] = 1$, $p = 0$, two exponential servers in cold backup ($b = 1$), one repair server ($r = 1$), and Hyperexponential repair time distribution, with mean time=1, $C_v^2 = 2$, $p_1 = 0.2113248$, $\lambda_1 = 0.4226497$ and $\lambda_2 = 1.57735$.

Our purpose is to see how $\mathbb{E}[X_r]$ varies as β increases. Three different trial functions were chosen for $F_{L(t)}$, all with $\mathbb{E}[L] = 1$. They are: (1) Deterministic distribution ($L = \ell = 1$); (2) the exponential $f(t) = \exp(-t)$; and (3) the same Hyperexponential distribution as the repair time distribution. Thus, (2) and (3) have a μ_m of 1.0 and 0.422649, respectively. The computations are presented in Fig. 3, together with $\beta_{m(\beta)}$. Observe that as $\beta_m \rightarrow 1$ (at $\beta = 2.09885$) curve (2) goes to ∞ . Similarly, curve (3) goes to ∞ as $\beta_{m(\beta)} \rightarrow 0.422649$ (at $\beta = 1.126203$).

Also note that when $\beta_m \geq \mu_m$ the variance of the total time distribution becomes infinite. In the cases here, this occurs at $\beta_m = 1.267592$. for (2) and $\beta_m = 0.702453$ for (3). We would expect that at these values performance would become unacceptable.

The inset in Fig. 3 shows the same curves for small β . Of course $\mathbb{E}[X_r(\beta = 0)] = 1$ in all cases, and grows very slowly, with an initial slope of 0. This is due to the redundant servers, since at least two failures must occur before the system fails. (If p had been set to some positive number, then the initial slopes would have been positive.) As β increases all the curves begin to increase rapidly as they go their individual merry ways to infinity.

So, we see that each system, and each job distribution has a value for β above which RESTART can no longer be neglected, and a larger β above which the system can no longer function properly.

7 Asymptotic Behavior of $H_r(x|\ell)$: Finding γ

So far, we have focussed on the mean value of the total time under *RESTART*. First, we considered $\mathbb{E}[X_r(\ell)]$ for a fixed $L = \ell$, and then used (18) to average over any jobtime distribution. Previously, Asmussen et al. [1] examined the asymptotic behavior of $H(x|\ell)$, ignoring the repair time. Since the number of failures must be geometrically distributed, it is clear that $H(x|\ell)$ goes to 0 exponentially as $x \rightarrow \infty$. They proved that the exponential parameter, $\gamma(\ell)$, is the solution of the equation

$$\int_0^\ell e^{s\gamma(\ell)} g(s) ds = 1,$$

where $g(s)$ is the failure distribution. Most recently, using an approach related to that used here, they proved a theorem which yields a value for γ which includes repair and the mutual dependence between UP and DOWN. We summarize that here, and discuss simplifications if the \mathbf{Y} 's are rank 1.

First, define the matrices

$$\mathcal{R}_{du}(\alpha) := \int_0^\infty e^{\alpha s} [\mathbf{g}(s)] ds = \int_0^\infty e^{\alpha s} [\exp(-s\mathbf{B}_d)\mathbf{B}_d \mathbf{Y}_d] ds$$

and

$$\mathcal{R}_{ud}(\alpha, \ell) := \int_0^\ell e^{\alpha s} [\mathbf{g}(s)] ds = \int_0^\ell e^{\alpha s} [\exp(-s\mathbf{B}_u)\mathbf{B}_u \mathbf{Y}_u] ds.$$

Then,

$$\mathcal{R}(\alpha, \ell) := \begin{bmatrix} \mathbf{0} & \mathcal{R}_{ud}(\alpha, \ell) \\ \mathcal{R}_{du}(\alpha) & \mathbf{0} \end{bmatrix}.$$

The matrix, $\mathcal{R}(\alpha, \ell)$ has the same block structure as \mathbf{Q} , where the diagonal blocks are identically $\mathbf{0}$, and the off-diagonal blocks are as given above. Then using Theorem 3.1 from [1] that states

Theorem 4 *There exists a $\gamma = \gamma(\ell)$ such that $\text{spr}[\mathcal{R}(\alpha = \gamma, \ell)] = 1$ and constant vectors $\mathbf{c}_u(\ell)$ and $\mathbf{c}_d(\ell)$ such that*

$$\Pr[X > x | i \in E_u] \approx [\mathbf{c}_u]_i e^{-\gamma x}, \quad \text{and} \quad \Pr[X > x | k \in E_d] \approx [\mathbf{c}_d]_k e^{-\gamma x}$$

□

We note that the formula $\mathbb{E}[X_r] = \int_0^\infty \Pr(X_r > x) dx$ and (17) suggest that $\gamma(\ell)$ is proportional to $e^{-\ell\beta_m}$ for large ℓ , and this can indeed be checked by comparison with the expressions in [16]. This is also demonstrated in the example below.

$\text{spr}[\mathcal{R}(\alpha, \ell)]$ can be brought into a more convenient form for computation by observing the following.

$$[\mathcal{R}(\alpha, \ell)]^2 = \begin{bmatrix} \mathcal{R}_{ud} \mathcal{R}_{du} & 0 \\ 0 & \mathcal{R}_{du} \mathcal{R}_{ud} \end{bmatrix}$$

so

$$\text{spr}[\mathcal{R}(\alpha, \ell)]^2 = \text{spr}[\mathcal{R}^2] = \max [\text{spr}[\mathcal{R}_{ud} \mathcal{R}_{du}], \text{spr}[\mathcal{R}_{du} \mathcal{R}_{ud}]].$$

It is straight-forward to show that

$$\mathcal{R}_{ud} = \{\mathbf{I}_u - \exp[-\ell(\mathbf{B}_u - \alpha \mathbf{I}_u)]\} (\mathbf{B}_u - \alpha \mathbf{I}_u)^{-1} \mathbf{B}_u \mathbf{Y}_u,$$

and

$$\mathcal{R}_{du} = (\mathbf{B}_d - \alpha \mathbf{I}_d)^{-1} \mathbf{B}_d \mathbf{Y}_d.$$

For a given ℓ these expressions can be evaluated for varying α , and by some root-finding method, find where

$$\text{spr}[\mathcal{R}(\alpha, \ell)]^2 = 1.$$

That value for α is $\gamma(\ell)$.

Note that \mathcal{R}_{ud} and \mathcal{R}_{du} do not commute, nor do \mathbf{Y}_u and \mathbf{Y}_d . But if either \mathbf{Y}_u or \mathbf{Y}_d are of rank 1, the eigenvalue problem simplifies greatly. Note that if either is of rank 1, then so are $\mathcal{R}_{ud} \mathcal{R}_{du}$ and $\mathcal{R}_{du} \mathcal{R}_{ud}$. Any such matrix can be written in the form $\mathbf{a}' \mathbf{b}$, where \mathbf{a}' and \mathbf{b} are a column-row vector pair of appropriate dimension. All such matrices have (at most) only one nonzero eigenvalue, and in fact,

$$\text{spr}[\mathbf{a}' \mathbf{b}] = \mathbf{b} \mathbf{a}'.$$

($\mathbf{b} \mathbf{a}'$ is a scalar.) This leads us to a corollary.

Corollary 4: *Suppose that $\mathbf{Y}_u = \boldsymbol{\varepsilon}'_u \mathbf{p}_d$, then*

$$\text{spr}[\mathcal{R}_{ud} \mathcal{R}_{du}] = \text{spr}[\mathcal{R}_{du} \mathcal{R}_{ud}] =$$

$$\mathbf{p}_d (\mathbf{B}_d - \alpha \mathbf{I}_d)^{-1} \mathbf{B}_d \mathbf{Y}_d \{\mathbf{I}_u - \exp[-\ell(\mathbf{B}_u - \alpha \mathbf{I}_u)]\} (\mathbf{B}_u - \alpha \mathbf{I}_u)^{-1} \mathbf{B}_u \boldsymbol{\varepsilon}'_u.$$

If $\mathbf{Y}_d = \boldsymbol{\varepsilon}'_d \mathbf{p}_u$, then

$$\text{spr}[\mathcal{R}_{ud} \mathcal{R}_{du}] = \text{spr}[\mathcal{R}_{du} \mathcal{R}_{ud}] =$$

$$\mathbf{p}_u \{\mathbf{I}_u - \exp[-\ell(\mathbf{B}_u - \alpha \mathbf{I}_u)]\} (\mathbf{B}_u - \alpha \mathbf{I}_u)^{-1} \mathbf{B}_u \mathbf{Y}_u (\mathbf{B}_d - \alpha \mathbf{I}_d)^{-1} \mathbf{B}_d \boldsymbol{\varepsilon}'_d.$$

Given any \mathcal{Q} we are then presented with the task of finding that value of α which makes these expressions equal 1, for each value of ℓ .

7.1 Simplest Example for $\gamma(\ell)$

We take as our example the simplest case from Sect. 6.1, an exponential server with failure rate, β , and a single exponential repair stage with rate λ . The various matrices and vectors trivialize to:

$$\mathbf{Y}_u = \mathbf{Y}_d = \mathbf{p}_u = \mathbf{p}_d = \boldsymbol{\varepsilon}'_u = \boldsymbol{\varepsilon}'_d = \mathbf{I}_u = \mathbf{I}_d = \mathbf{1}.$$

Also, $\mathbf{B}_u = \beta$, $\mathbf{B}_d = \lambda$ and Corollary 4 gives the following equation.

$$\text{spr}[\mathcal{R}(\alpha, \ell)] \text{spr}[\mathcal{R}_{ud} \mathcal{R}_{du}] = \text{spr}[\mathcal{R}_{du} \mathcal{R}_{ud}] = \text{spr}[\mathcal{R}]^2 = \frac{\lambda}{(\lambda - \alpha)} \frac{\beta [1 - e^{-\ell(\beta - \alpha)}]}{(\beta - \alpha)}$$

$\gamma(\ell)$ is that value of α which makes $\text{spr}[\mathcal{R}] = 1$. This is a standard root-finding problem with one difficulty. Even though $\alpha = \beta$ appears to be a root, it is not a solution because it is also a root of the denominator. It turns out that $\gamma(\ell = 0) = \lambda$. This makes sense for this simple system. When $\ell\beta \ll 1$ the probability that the job will fail is also very small, but since we are looking at the tail of $H(x|\ell)$, the job must have failed at least once for x to be large. For this example, the repair distribution is $e^{-x\lambda}$.

In Fig. 5, $\log(\gamma)$ approaches a straight line for every λ , and with the same slope. In fact, as was asserted in the comment after Theorem 4, This infers that in general,

$$\lim_{\ell \rightarrow \infty} \gamma(\ell|\lambda) e^{\ell\beta_m} = a(\lambda),$$

where β_m (the smallest eigenvalue of \mathbf{B}_u) is the slope of the line, and is independent of λ .

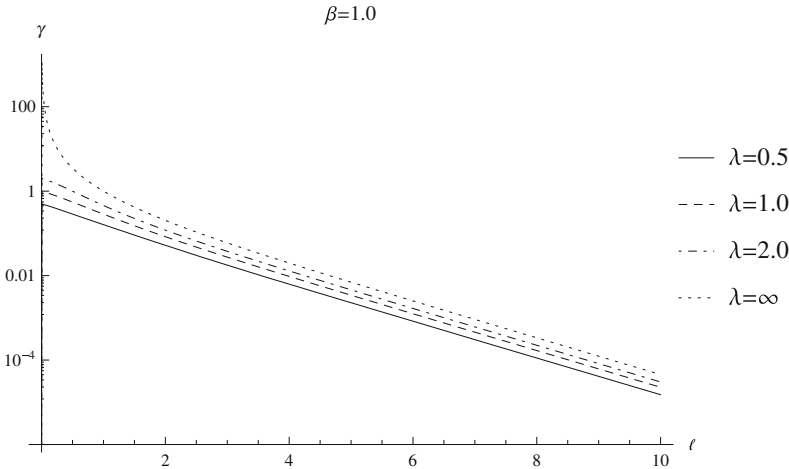


Fig. 5 $\log[\gamma(\ell|\lambda, \beta)]$ as a function of ℓ for $\lambda \in \{0.5, 1.0, 2.0, \infty\}$ and $\beta = 1$, for the simplest system of one UP and one DOWN state. For this system, $\gamma(0|\lambda, \beta) = \lambda$

8 Summary

We have examined arbitrary systems with repair that can be described by some Markov model, and have derived a general equation for the mean system time for a job that would take time ℓ if there were no failures, but recovers under the *RESTART* protocol. This includes dependence between failures and repairs. We then provided a few examples. Next, we described systems where the job time comes from some distribution, $F(t)$, and provide examples where $\mathbb{E}[X_r] \rightarrow \infty$. Finally, we discussed $\gamma(\ell)$, the exponential tail for $H_r(x|\ell)$, and how the formula for its computation simplifies if either \mathbf{Y} is of rank 1.

References

1. Asmussen S, Fiorini PM, Lipsky L, Rolski T, Sheahan R (2008) Asymptotic behavior of total times for jobs that must start over if a failure occurs. *Math Oper Res* 33(4):932–944
2. Asmussen S, Lipsky L, Thompson S (2014) Checkpointing in failure recovery in computing and data transmission. In: Sericola B, Telek M, Horváth G (eds) *Analytical and stochastic modelling techniques and applications—21st international conference, ASMTA 2014, Budapest, Hungary, 30 June–2 July 2014*. Proceedings, vol 8499 of *Lecture Notes in Computer Science*, pp 253–272. Springer
3. Bobbio A, Trivedi K (1990) Computation of the distribution of the completion time when the work requirement is a ph random variable. *Stoch Models* 6:133–150
4. Castillo X, Siewiorek DP (1980) A performance-reliability model for computing systems. In: *Proceedings of the FTCS-10, Silver Spring, MD, IEEE Computer Society*, pp 187–192
5. Chimento Jr PF, Trivedi KS (1993) The completion time of programs on processors subject to failure and repair. *IEEE Trans Comput* 42(1)
6. Chlebus BS, De Prisco R, Shvartsman AA (2001) Performing tasks on synchronous restartable message-passing processors. *Distrib Comput* 14:49–64
7. De Prisco R, Mayer A, Yung M (1994) Time-optimal message-efficient work performance in the presence of faults. In: *Proceedings of the 13th ACM PODC*, pp 161–172
8. Fiorini PM (1998) Modeling telecommunication systems with self-similar data traffic. Department of Computer Science, University of Connecticut, phd thesis edition
9. Gross D, Harris C (1998) *Fundamentals of queueing theory*, 3rd edn. Wiley
10. Jagerman D (1982) An inversion technique for the laplace transform. *Bell Syst Tech J* 61:1995–2002
11. Kulkarni V, Nicola V, Trivedi K (1986) On modeling the performance and reliability of multimode systems. *J Syst Softw* 6:175–183
12. Kulkarni V, Nicola V, Trivedi K (1987) The completion time of a job on a multimode system. *Adv Appl Probab* 19:932–954
13. Lipsky L (2008) *Queueing theory. A linear algebraic approach*, 2nd edn. Springer
14. Rubino G, Sericola B (2014) *Markov chains and dependability theory*, 1st edn. Cambridge University Press
15. Sheahan R, Lipsky L, Fiorini P, Asmussen S (2006) On the distribution of task completion times for tasks that must restart from the beginning if failure occurs. *SIGMETRICS Perform Eval Rev* 34:24–26
16. Søren A, Lester L, Asmussen S, Lipsky L, Thompson S (2015) Markov renewal methods in restart problems in complex systems. *Thiele Research Reports, Department of Mathematical Sciences, University of Aarhus*, vol 156
17. Trivedi K (2001) *Probability and statistics with reliability, queueing, and computer science applications*, 2nd edn. Wiley-Interscience

Vacation Queueing Models of Service Systems Subject to Failure and Repair

Oliver C. Ibe

Abstract We consider a queueing system that can randomly fail either when it is idle or while serving a customer. The system can be modeled by a vacation queueing system since it cannot serve customers when it is down; that is, the server is on a forced vacation when the system fails. We provide the availability and performance analysis of this system in this chapter.

Keywords Vacation queueing · Performance modeling · Availability analysis · Unreliable service

1 Introduction

System failure is a fact of life. No engineering device can be built such that it is failure-free. When a system fails, it can either be repaired because the cost of repairing it is far less than the cost of purchasing a new one, or it is abandoned because the cost of repairing it is prohibitively high. Fortunately, most computer systems belong to the first category of devices, and those are the subject of the discussion in this chapter.

A computer system can suffer one of two types of failures: hard failure and soft failure. A hard failure requires the physical repair of the failed system, which usually takes a long time because it requires the presence of the field services personnel. By contrast, after a system has suffered a soft failure, no physical repair is required. The system is restored to operation by means of a system reboot or some other repair function that does not require the presence of the field services personnel. As long as the system is down and is not being used for the intended service, it can be defined as being on vacation.

In this chapter, we consider different ways of modeling system failure by vacation models. It must be emphasized that the concept of server vacation as used by queueing

O.C. Ibe (✉)
Department of Electrical and Computer Engineering,
University of Massachusetts, Lowell, MA 01854, USA
e-mail: oliver_ibe@uml.edu

theorists is slightly different from the way it is used to model systems that are subject to failure and repair. In a classical vacation queueing system, the server takes a vacation when there is no customer left in the system. There are two types of vacation queueing systems: single vacation queueing systems and multiple vacation queueing systems.

In a single vacation queueing system, when the server returns from a vacation and finds at least one customer waiting, he exhaustively serves the customers, at the end of which he takes another vacation. If there is no waiting customer when the server returns from a vacation, the server waits until at least one customer is served before commencing another vacation. Thus, in a single vacation queueing system, the server takes a vacation after a busy period of nonzero duration.

In a multiple vacation queueing system, when the server returns from a vacation and finds at least one customer waiting, he serves the customers exhaustively, as in the single vacation queueing system, before commencing another vacation. However, if he finds the system empty upon his return from a vacation, he immediately commences another vacation. Thus, the difference between the single vacation queueing system and the multiple vacation queueing system lies in the server's behavior upon his return from a vacation and finding the system empty.

The concept of server vacation in queueing systems was first introduced by Cooper [1] and later analyzed more formally by Levy and Yechiali [2]. Vacation queueing systems are widely used to model different communication and manufacturing systems. A good survey is given in [3], and complete textbooks have been dedicated to the subject [4, 5].

In this chapter, we consider how to model different failure schemes by some form of vacation queueing systems. In the next section, we define the system model and introduce different vacation queueing systems to describe these models.

2 System Model

We consider a queueing system where customers arrive according to a Poisson process with rate λ . The time to serve a customer is assumed to be exponentially distributed with mean $1/\mu$, where $\mu > \lambda$. The system is subject to breakdown and repair. The breakdown can be scheduled or it can occur in a random manner. As discussed earlier, when a breakdown occurs we define the system to be on vacation since it can no longer serve customers when it breaks down. We consider two types of vacations:

- (a) A vacation scheme in which the server is forced to take a vacation at random instants when a high priority event occurs. The high priority event is usually a server breakdown. If the breakdown occurs while the server is busy serving a customer, the customer's service is abandoned and the repair of the system is started. At the end of the repair, the preempted customer's service is started from the beginning. All customers that arrive while the system is being repaired are

assumed to wait; that is, customer balking is not allowed. We refer to this case as the “random vacation model.”

- (b) A vacation scheme similar to case (a) above except that there are two types of priority events, which occur independently of each other. In this case we assume that there are two types of failures [6]:
 - (i) a hard failure that takes a long time to repair because it usually involves the presence of the field services personnel who may not be on site when the failure occurs
 - (ii) a soft failure that does not require the intervention of the field services personnel because it can be fixed by an action such as system reboot and thus takes a much shorter time to fix.

We refer to this case as the “differentiated random vacation model.”
 The analysis of these models is the subject of the remainder of this chapter.

3 Analysis of the Random Vacation Model

In this model, we assume that only hard failures can occur and that customers can continue to arrive and wait while the system is being repaired. The state of the system is denoted by (r, k) , where r is the number of customers and

$$k = \begin{cases} 0 & \text{if the system is up} \\ 1 & \text{if the system is down} \end{cases}$$

When the system is up, the time until it fails is assumed to be exponentially distributed with mean $1/\gamma$. Similarly, when the system is down, the time to repair it is assumed to be exponentially distributed with mean $1/\eta$. It is assumed that customers that arrive while the system is down will wait for it to be repaired; that is, balking is not allowed. Also, we assume that there is no defection from the queue when the server is down; a customer will only leave the system upon the completion of its service. Thus, the state transition-rate diagram of the model is shown in Fig. 1.

From local balance in Fig. 1 we have that

$$\begin{aligned} \eta \sum_{r=0}^{\infty} P_{r,1} &= \gamma \sum_{r=0}^{\infty} P_{r,0} \Rightarrow \sum_{r=0}^{\infty} P_{r,1} = \frac{\gamma}{\eta} \sum_{r=0}^{\infty} P_{r,0} \\ 1 &= \sum_{r=0}^{\infty} P_{r,0} + \sum_{r=0}^{\infty} P_{r,1} = \left(1 + \frac{\gamma}{\eta}\right) \sum_{r=0}^{\infty} P_{r,0} = \left(\frac{\eta + \gamma}{\eta}\right) \sum_{r=0}^{\infty} P_{r,0} \\ &\Rightarrow \sum_{r=0}^{\infty} P_{r,0} = \frac{\eta}{\eta + \gamma}, \quad \sum_{r=0}^{\infty} P_{r,1} = \frac{\gamma}{\eta + \gamma} \end{aligned}$$

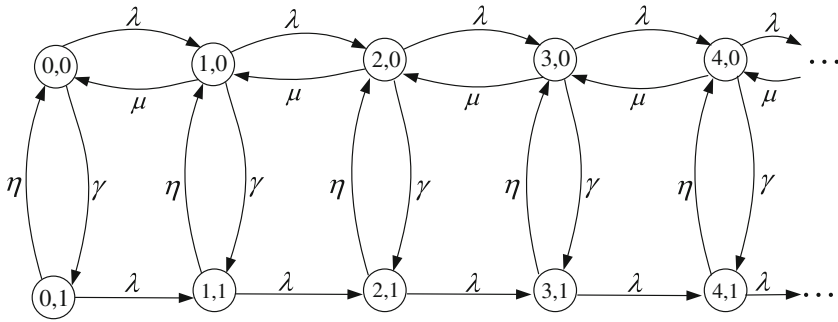


Fig. 1 State transition-rate diagram for the random vacation model

Thus, the availability of the system is

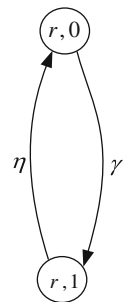
$$\alpha = \sum_{r=0}^{\infty} P_{r,0} = \frac{\eta}{\eta + \gamma} = \frac{(1/\gamma)}{(1/\gamma) + (1/\eta)} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} \tag{1}$$

where MTBF is the mean time between failures and MTTR is the mean time to repair; this is the traditional definition of availability.

Figure 1 is a special type of Markov chain called a quasi-birth-and-death process, which is usually analyzed via the matrix geometric method [7]. In this chapter we propose an alternative solution method that uses the concept of state aggregation. Specifically, let the states $(r, 0)$ and $(r, 1)$, $r = 0, 1, 2, \dots$, be lumped into a “superstate” (r, x) . That is, the superstate has the state transition-rate diagram shown in Fig. 2.

The concept of state aggregation has been successfully used by other authors such as [8, 9]. With respect to Fig. 2, the probability of being in the up phase $(r, 0)$, given that the process is in superstate (r, x) , is $\alpha = \eta / (\eta + \gamma)$. Let p_0 denote the probability that the process makes a transition from superstate $(0, x)$ to superstate $(1, x)$. To make this transition, the process will be in phase 0, with probability α , and

Fig. 2 Superstate (r, x)



a customer arrives before a failure occurs, or the process is in phase 1 and a customer arrives before repair is completed. Thus,

$$p_0 = \alpha \left(\frac{\lambda}{\lambda + \gamma} \right) + (1 - \alpha) \left(\frac{\lambda}{\lambda + \eta} \right) \tag{2}$$

Similarly, let p_1 denote the probability that the process makes a transition from superstate (r, x) to superstate $(r + 1, x)$, $r \geq 1$. To make this transition, the process will be in phase 0, with probability α , and a customer arrives before a failure occurs or a customer departs, or the process is in phase 1 and a customer arrives before repair is completed. Thus,

$$p_1 = \alpha \left(\frac{\lambda}{\lambda + \mu + \gamma} \right) + (1 - \alpha) \left(\frac{\lambda}{\lambda + \eta} \right) \tag{3}$$

Finally, let q_1 denote the probability that the process makes a transition from superstate (r, x) to superstate $(r - 1, x)$, $r \geq 1$. To make this transition, the process will be in phase 0, with probability α , and a customer departs before a failure occurs or a customer arrives. Thus,

$$q_1 = \frac{\alpha \mu}{\lambda + \mu + \gamma} \tag{4}$$

With these three probabilities defined we convert the continuous-time Markov chain into a discrete-time birth-and-death process (equivalently, a random walk with a reflecting barrier at 0 and with stay), as shown in Fig. 3. (A random walk with stay is used to model a two-player game that can end in a tie. Thus, when a game ends in a tie, neither of the players loses or gains; the value of each player’s net worth remains unchanged, which is denoted by a self-loop as shown in Fig. 3.)

From local balance we have that

$$\begin{aligned} \pi_0 p_0 &= \pi_1 q_1 \Rightarrow \pi_1 = \pi_0 \left(\frac{p_0}{q_1} \right) \\ \pi_1 p_1 &= \pi_2 q_1 \Rightarrow \pi_2 = \pi_1 \left(\frac{p_1}{q_1} \right) = \pi_0 \left(\frac{p_0}{q_1} \right) \left(\frac{p_1}{q_1} \right) \end{aligned}$$

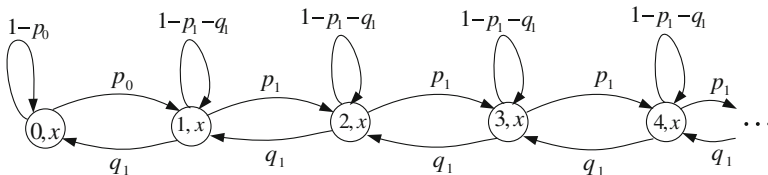


Fig. 3 State transition diagram of superstates

$$\begin{aligned} \pi_2 p_1 &= \pi_3 q_1 \Rightarrow \pi_3 = \pi_2 \left(\frac{p_1}{q_1} \right) = \pi_0 \left(\frac{p_0 p_1^2}{q_1^3} \right) = \pi_0 \left(\frac{p_0}{q_1} \right) \left(\frac{p_1}{q_1} \right)^2 \\ \pi_3 p_1 &= \pi_4 q_1 \Rightarrow \pi_4 = \pi_3 \left(\frac{p_1}{q_1} \right) = \pi_0 \left(\frac{p_0}{q_1} \right) \left(\frac{p_1}{q_1} \right)^3 \end{aligned}$$

In general,

$$\pi_r = \pi_0 \left(\frac{p_0}{q_1} \right) \left(\frac{p_1}{q_1} \right)^{r-1} \quad r = 1, 2, \dots \tag{5}$$

The normalization equation is

$$\begin{aligned} 1 &= \sum_{r=0}^{\infty} \pi_r = \pi_0 + \sum_{r=1}^{\infty} \pi_r = \pi_0 \left\{ 1 + \frac{p_0}{q_1} \sum_{r=1}^{\infty} \left(\frac{p_1}{q_1} \right)^{r-1} \right\} = \pi_0 \left\{ 1 + \frac{p_0}{q_1} \left(\frac{1}{1 - [p_1/q_1]} \right) \right\} \\ &= \pi_0 \left\{ 1 + \left(\frac{p_0}{q_1 - p_1} \right) \right\} \end{aligned}$$

where the sum holds only if $q_1 > p_1$; that is, we require that

$$\begin{aligned} q_1 - p_1 &= \alpha \left\{ \frac{\mu - \lambda}{\lambda + \mu + \gamma} \right\} - (1 - \alpha) \left\{ \frac{\lambda}{\lambda + \eta} \right\} = \alpha (1 - \rho) \left\{ \frac{\mu}{\lambda + \mu + \gamma} \right\} - (1 - \alpha) \left\{ \frac{\lambda}{\lambda + \eta} \right\} \\ &= \frac{\alpha \mu (1 - \rho) (\lambda + \eta) - \lambda (1 - \alpha) (\lambda + \mu + \gamma)}{(\lambda + \mu + \gamma) (\lambda + \eta)} > 0 \end{aligned}$$

that can be transformed to the inequality

$$\alpha \mu (1 - \rho) (\lambda + \eta) - \lambda (1 - \alpha) (\lambda + \mu + \gamma) > 0.$$

From this it follows that

$$\alpha \{ \mu (1 - \rho) (\lambda + \eta) + \lambda (\lambda + \mu + \gamma) \} > \lambda (\lambda + \mu + \gamma)$$

This means that

$$\alpha > \frac{\lambda (\lambda + \mu + \gamma)}{\mu (1 - \rho) (\lambda + \eta) + \lambda (\lambda + \mu + \gamma)} = \frac{\rho (\lambda + \mu + \gamma)}{(1 - \rho) (\lambda + \eta) + \rho (\lambda + \mu + \gamma)} \tag{6}$$

This implies that if the condition of Eq. (6) is satisfied, then

$$\pi_0 = \frac{1}{1 + \left(\frac{p_0}{q_1 - p_1} \right)} = \frac{q_1 - p_1}{q_1 - p_1 + p_0} \tag{7}$$

Under the condition of Eq. (6), the expected number of customers in the system is

$$\begin{aligned}
 E[N] &= \sum_{k=1}^{\infty} k\pi_k = \pi_0 \left(\frac{p_0}{q_1}\right) \sum_{k=1}^{\infty} k \left(\frac{p_1}{q_1}\right)^{k-1} = \pi_0 \left(\frac{p_0}{q_1}\right) \sum_{k=1}^{\infty} k\beta^{k-1} = \pi_0 \left(\frac{p_0}{q_1}\right) \left\{ \frac{1}{(1-\beta)^2} \right\} \\
 &= \frac{\pi_0 p_0 q_1}{(q_1 - p_1)^2} = \frac{p_0 q_1}{(q_1 - p_1)(q_1 - p_1 + p_0)} \tag{8}
 \end{aligned}$$

where $\beta = p_1/q_1$.

It is often assumed that a system that is subject to server breakdown can be modeled by a preemptive priority queue. However, as noted by Heathcote [10], one of the differences between a queue with server breakdown and a preemptive priority queue is that a queue of breakdowns cannot occur. There can be only one interruption at a time; multiple interruptions cannot be queued in the same way that priority customers can be queued. Thus, the preemptive priority queue model cannot represent a queue with server breakdown.

To obtain the mean delay $E[T]$ we proceed as follows. Consider a tagged customer who arrives at the system. Since Poisson arrivals see time averages [11], the mean delay of the customer is made up of two parts:

- (a) The mean time $E[R]$ to complete the current service, if a customer was receiving service upon the tagged customer’s arrival; or the mean time to complete the server repair, if the system was down and was being repaired when the tagged customer arrived.
- (b) The mean time $E[T_1]$ to serve the customers that were waiting when the tagged customer arrived.

Let M denote the number of times that a customer is preempted until service completion. That is, the customer is preempted $M - 1$ times and at the M th attempt the customer successfully completes service without interruption. Let v denote the probability of service completion without interruption. Thus, M is a geometrically distributed random variable with success probability v , mean $E[M] = 1/v$, and its probability mass function (PMF) is given by [12]

$$p_M(m) = v(1 - v)^{m-1} \quad m = 1, 2, \dots$$

As discussed earlier, let X denote the time to serve a customer without interruption and Y the time to repair the server when it fails. Since the service time and repair time are exponentially distributed, from random incidence, the mean residual service time of a customer is $1/\mu$, and the mean residual repair time is $1/\eta$. Thus,

$$E[R] = \frac{1 - \alpha}{\eta} + \alpha\rho \left\{ \frac{1}{\mu} + \left(\frac{1}{\mu} + \frac{1}{\eta}\right) \left(\frac{1}{v} - 1\right) \right\}$$

This result can be explained as follows. The first term on the right is the mean residual service time, which occurs when the server is being repaired. The second term is the mean residual service time when the server is up but busy. The second term within

the curly brackets accounts for the service time being interrupted on the average $(1/v) - 1$ times and the first term within the brackets is the mean uninterrupted service time. Similarly,

$$E[T_1] = \left\{ \frac{1}{\mu} + \left(\frac{1}{\mu} + \frac{1}{\eta} \right) \left(\frac{1}{v} - 1 \right) \right\} \{E[N] + 1\}$$

which follows from the same reason given above. Thus,

$$E[T] = E[R] + E[T_1] = \frac{1 - \alpha}{\eta} + \left\{ \frac{1}{\mu} + \left(\frac{1}{\mu} + \frac{1}{\eta} \right) \left(\frac{1}{v} - 1 \right) \right\} \{\alpha\rho + E[N] + 1\} \tag{9}$$

The probability that no failure occurs over an interval of a service time is

$$v = \int_0^\infty e^{-\gamma x} f_X(x) dx = \int_0^\infty e^{-\gamma x} \mu e^{-\mu x} dx = \frac{\mu}{\mu + \gamma}$$

where $f_X(x)$ is the probability density function (PDF) of X . Applying this result to Eq. (9) we obtain

$$E[T] = \frac{1 - \alpha}{\eta} + \left\{ \frac{1}{\mu} + \left(\frac{1}{\mu} + \frac{1}{\eta} \right) \left(\frac{\gamma}{\mu} \right) \right\} \{\alpha\rho + E[N] + 1\} \tag{10}$$

Observe that when $\alpha = 1 \Rightarrow \gamma = 0$, we obtain

$$E[T]_{\alpha=1} = \frac{1}{\mu} \{\rho + E[N] + 1\}$$

which is the sum of the mean residual service time, the mean service time for all waiting customers when the tagged customer arrived and the mean service time of the customer.

3.1 Numerical Results

The problem of the performance analysis of queueing systems with server breakdown has been studied by White and Christie [13], Miller [14], Gaver [15], Avi-Itzhak and Naor [16], and Mitrany and Avi-Itzak [17] as preemptive priority queueing systems. In this case, the times between occurrence of a breakdown is essentially the inter-arrival time of the high priority customer, and the repair time is the service time of the high priority customer. Since we assume that the service time of a high priority customer (i.e., repair time) is exponentially distributed, both preemptive resume and preemptive repeat disciplines have identical results because of the forgetfulness property of the exponential distribution.

For a two priority queuing system, it can be shown (see Ibe [18]) that the mean delay in the system for the lower priority customer is given by

$$\begin{aligned}
 E[T_{\text{PRIO}}] &= \frac{(1/\mu)}{1-u} + \frac{\gamma E[Y^2] + \lambda E[X^2]}{2(1-\rho-u)(1-u)} = \frac{1}{\mu(1-u)} + \frac{\gamma(2/\eta^2) + \lambda(2/\mu^2)}{2(1-\rho-u)(1-u)} \\
 &= \frac{1}{\mu(1-u)} + \frac{u/\eta + \rho/\mu}{(1-\rho-u)(1-u)} \tag{11}
 \end{aligned}$$

where $u = \gamma/\eta = (1-\alpha)/\alpha$.

We assume that $\alpha = 0.95, \mu = 1 \Rightarrow \lambda = \rho, 1/\gamma = 1000$. This implies that $\gamma = 0.001$ and $\eta = 0.019$. Table 1 compares the mean delay versus the server’s utilization factor for the current model, denoted by $E[T]$, with the results from the Mitrany–Avi-Itzak method, denoted by $E[T_{\text{MAvi}}]$; the Avi-Itzak–Naor method, denoted by $E[T_{\text{Avi-Naor}}]$; and the priority model, denoted by $E[T_{\text{PRIO}}]$. From the table we observe that the different models perform approximately identically for low to medium server utilization. However, at high values of server utilization there is noticeable difference in their behavior, which stems from the way the models are defined. As discussed earlier, the priority model assumes that a queue of breakdowns can occur, which means that high priority customers (i.e., failures) can accumulate. This tends to overestimate the mean delay of the low priority customers.

Table 1 Mean delay versus server utilization for different models

ρ	$E[T]$	$E[T_{\text{MAvi}}]$	$E[T_{\text{Avi-Naor}}]$	$E[T_{\text{PRIO}}]$
0.05	4.3612	3.8674	3.8889	4.3728
0.10	4.4935	4.0698	4.1176	4.6308
0.15	4.6301	4.2945	4.3750	4.9212
0.20	4.7767	4.5455	4.6667	5.2504
0.25	4.9372	4.8276	5.0000	5.6268
0.30	5.1155	5.1471	5.3846	6.0614
0.35	5.3166	5.5118	5.8333	6.5688
0.40	5.5476	5.9322	6.3636	7.1688
0.45	5.8184	6.4220	7.0000	7.8895
0.50	6.1440	7.0000	7.7778	8.7712
0.55	6.5477	7.6923	8.7500	9.8749
0.60	7.0685	8.5366	10.0000	11.2963
0.65	7.7760	9.5890	11.6667	13.1957
0.70	8.8090	10.9375	14.0000	15.8629
0.75	10.4891	12.7273	17.5000	19.8815
0.80	13.7683	15.2174	23.3333	26.6270
0.85	23.2677	18.9189	35.0000	40.3003

4 Analysis of the Differentiated Random Vacation Model

We now extend the random vacation model to permit hard and soft failures. The hard failure rate is γ_1 , the soft failure rate is γ_2 , the hard failure repair rate is η_1 and the soft failure repair rate is η_2 . The state of the system is represented by (r, k) where r is the number of customers in the system and

$$k = \begin{cases} 0 & \text{if the system is up} \\ 1 & \text{if the system is down due to hard failure} \\ 2 & \text{if the system is down due to soft failure} \end{cases}$$

Figure 4 is the state transition-rate diagram of the process. This is essentially a differentiated forced vacation queueing system.

Note that this model is different from the standard differentiated multiple vacation queueing model reported in [18]. In the standard differentiated multiple vacation queueing model the server takes a type 1 vacation after completing a busy cycle that includes serving at least one customer. If the server returns from a type 1 vacation and finds no customer waiting, he commences a type 2 vacation of a shorter duration. In the differentiated random vacation model, the vacation types are defined by the type of failure that occurs rather than serving all waiting customers.

From local balance in Fig. 4,

$$\eta_1 \sum_{r=0}^{\infty} P_{r,1} = \gamma_1 \sum_{r=0}^{\infty} P_{r,0} \Rightarrow \sum_{r=0}^{\infty} P_{r,1} = \frac{\gamma_1}{\eta_1} \sum_{r=0}^{\infty} P_{r,0}$$

$$\eta_2 \sum_{r=0}^{\infty} P_{r,2} = \gamma_2 \sum_{r=0}^{\infty} P_{r,0} \Rightarrow \sum_{r=0}^{\infty} P_{r,2} = \frac{\gamma_2}{\eta_2} \sum_{r=0}^{\infty} P_{r,0}$$

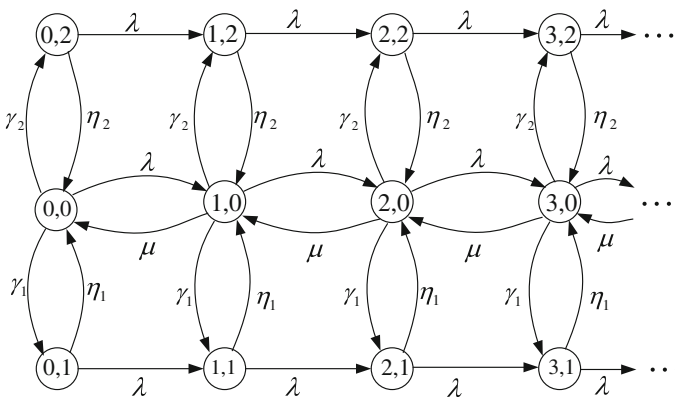
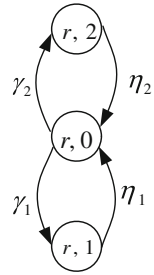


Fig. 4 State transition-rate diagram of the differentiated random vacation model

Fig. 5 Superstate (r, y)



$$\begin{aligned}
 1 &= \sum_{r=0}^{\infty} P_{r,0} + \sum_{r=0}^{\infty} P_{r,1} + \sum_{r=0}^{\infty} P_{r,2} = \left(1 + \frac{\gamma_1}{\eta_1} + \frac{\gamma_2}{\eta_2}\right) \sum_{r=0}^{\infty} P_{r,0} \\
 &= \left(\frac{\eta_1\eta_2 + \gamma_1\eta_2 + \eta_1\gamma_2}{\eta_1\eta_2}\right) \sum_{r=0}^{\infty} P_{r,0} \\
 \Rightarrow \sum_{r=0}^{\infty} P_{r,0} &= \frac{\eta_1\eta_2}{\eta_1\eta_2 + \gamma_1\eta_2 + \eta_1\gamma_2} \\
 \sum_{r=0}^{\infty} P_{r,1} &= \frac{\gamma_1}{\eta_1} \left\{ \frac{\eta_1\eta_2}{\eta_1\eta_2 + \gamma_1\eta_2 + \eta_1\gamma_2} \right\} = \frac{\gamma_1\eta_2}{\eta_1\eta_2 + \gamma_1\eta_2 + \eta_1\gamma_2} \\
 \sum_{r=0}^{\infty} P_{r,2} &= \frac{\gamma_2}{\eta_2} \left\{ \frac{\eta_1\eta_2}{\eta_1\eta_2 + \gamma_1\eta_2 + \eta_1\gamma_2} \right\} = \frac{\eta_1\gamma_2}{\eta_1\eta_2 + \gamma_1\eta_2 + \eta_1\gamma_2}
 \end{aligned}$$

The availability of the system is given by

$$\alpha = \sum_{r=0}^{\infty} P_{r,0} = \frac{\eta_1\eta_2}{\eta_1\eta_2 + \gamma_1\eta_2 + \eta_1\gamma_2} \tag{12}$$

We use the same clustering scheme used for the hard-failure-only model by noting that in the present case there are three phases in each level. Thus, the superstate structure is shown in Fig. 5.

The only difference between this model and the previous one is the new value of the availability, α . Thus, the analysis is exactly the same as that of the two-phase superstate.

5 Extension of the Random Vacation Model: System with Customer Balking

An extension of the random failure model is to introduce customer balking when the server is down. This means that when the server is down some customers choose to join the queue while others choose not to join the queue. Let ϕ denote the probability

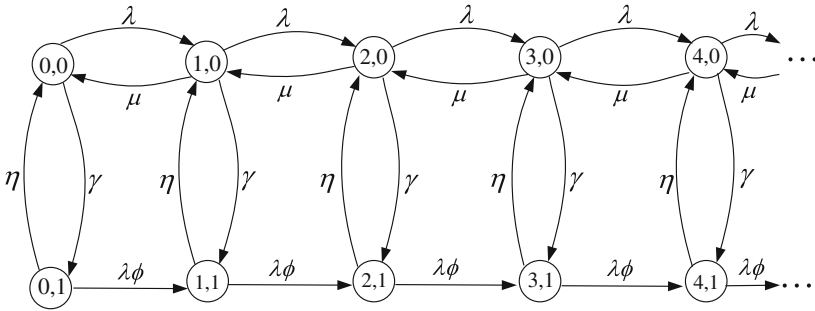


Fig. 6 Markov chain for system with customer balking

that a customer arriving when the server is down actually joins the queue; thus, with probability $1 - \phi$ an arriving customer does not join the queue. Thus, the effective arrival rate when the server is down is $\lambda\phi$. The Markov chain for this case is shown in Fig. 6.

The only modification to the original model is a new value for $p_i, i = 0, 1$. These new values are:

Table 2 Values of $E[T]$ for different values of ϕ

ρ	$\phi = 1$	$\phi = 0.5$	$\phi = 0.4$	$\phi = 0.3$	$\phi = 0.2$
0.05	4.3612	4.3503	4.3464	4.3416	4.3350
0.10	4.4935	4.4839	4.4800	4.4746	4.4665
0.15	4.6301	4.6211	4.6173	4.6117	4.6029
0.20	4.7767	4.7680	4.7641	4.7584	4.7489
0.25	4.9372	4.9283	4.9243	4.9182	4.9079
0.30	5.1155	5.1061	5.1018	5.0952	5.0838
0.35	5.3166	5.3064	5.3017	5.2944	5.2815
0.40	5.5476	5.5362	5.5309	5.5226	5.5078
0.45	5.8184	5.8052	5.7991	5.7894	5.7720
0.50	6.1440	6.1283	6.1209	6.1093	6.0882
0.55	6.5477	6.5283	6.5192	6.5047	6.4783
0.60	7.0685	7.0434	7.0315	7.0127	6.9781
0.65	7.7760	7.7415	7.7251	7.6991	7.6514
0.70	8.8090	8.7574	8.7329	8.6940	8.6225
0.75	10.4891	10.4012	10.3596	10.2937	10.1728
0.80	13.7683	13.5803	13.4920	13.3525	13.0992
0.85	23.2677	22.5758	22.2584	21.7664	20.9014
0.90	526.7141	267.4225	216.3477	165.5295	114.9660

$$p_0 = \alpha \left(\frac{\lambda}{\lambda + \gamma} \right) + (1 - \alpha) \left(\frac{\lambda\phi}{\lambda\phi + \eta} \right)$$

$$p_1 = \alpha \left(\frac{\lambda}{\lambda + \mu + \gamma} \right) + (1 - \alpha) \left(\frac{\lambda\phi}{\lambda\phi + \eta} \right)$$

Substituting these modified values of p_0 and p_1 into the mean delay obtained for the original model we obtain the solution to the new problem.

Table 2 shows the mean delay for different values of ϕ , where $\phi = 1$ corresponds to the original model where there is no customer balking. It can be observed that as the probability of balking, $1 - \phi$, increases, the mean delay slightly decreases.

6 Extension of the Random Vacation Model: System with Full and Partial Service

In some applications the service rate is not completely zero when the system is down. Instead the system operates at a reduced service rate when the system is down. Thus, another extension of the random failure model is the case where the server can render partial service when it is down and full service when it is up. This is similar to the *working vacation queuing model* [19] where the server provides service at a reduced rate rather than completely stopping service when he is on vacation. Thus, in this extension we assume that there are two service rates: μ_1 , which is the service rate when the system is up, and $\mu_2 < \mu_1$ is the service rate when the system is down. Figure 7 is the state transition-rate diagram of the model.

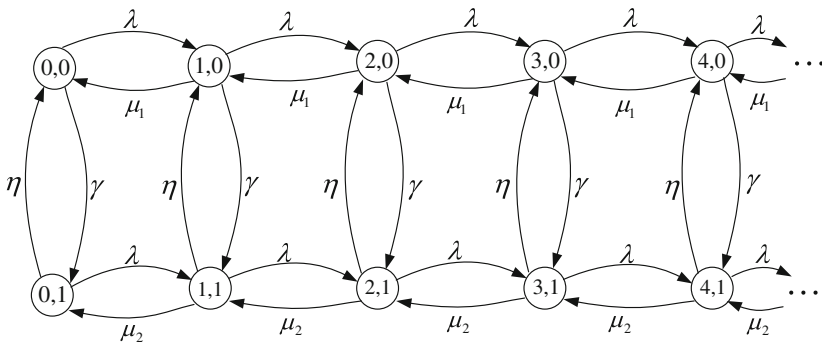


Fig. 7 Markov chain for system with full and partial service

In this case we obtain the following parameters:

$$\begin{aligned}
 p_0 &= \alpha \left(\frac{\lambda}{\lambda + \gamma} \right) + (1 - \alpha) \left(\frac{\lambda}{\lambda + \eta} \right) \\
 p_1 &= \alpha \left(\frac{\lambda}{\lambda + \mu_1 + \gamma} \right) + (1 - \alpha) \left(\frac{\lambda}{\lambda + \mu_2 + \eta} \right) \\
 q_1 &= \alpha \left(\frac{\mu_1}{\lambda + \mu_1 + \gamma} \right) + (1 - \alpha) \left(\frac{\mu_2}{\lambda + \mu_2 + \eta} \right)
 \end{aligned}$$

Substituting these modified values of p_0 , p_1 and q_1 into the mean delay obtained for the original model we obtain the solution to the new problem. Assume that $\mu_2 = c\mu_1$. Table 3 shows the mean delay for different values of c , where $c = 0$ corresponds to the original model. From the table we observe that as c increases the mean delay decreases. The impact is more drastic at high values of the offered load.

Table 3 Values of $E[T]$ for different values of c

ρ	$c = 0$	$c = 0.2$	$c = 0.3$	$c = 0.4$	$c = 0.5$
0.05	4.3612	4.3189	4.3152	4.3132	4.3118
0.10	4.4935	4.4437	4.4372	4.4333	4.4306
0.15	4.6301	4.5739	4.5646	4.5588	4.5547
0.20	4.7767	4.7134	4.7012	4.6931	4.6874
0.25	4.9372	4.8650	4.8495	4.8389	4.8312
0.30	5.1155	5.0320	5.0125	4.9988	4.9887
0.35	5.3166	5.2185	5.1940	5.1766	5.1635
0.40	5.5476	5.4301	5.3993	5.3770	5.3601
0.45	5.8184	5.6747	5.6356	5.6069	5.5849
0.50	6.1440	5.9637	5.9133	5.8759	5.8470
0.55	6.5477	6.3146	6.2482	6.1984	6.1597
0.60	7.0685	6.7550	6.6647	6.5969	6.5439
0.65	7.7760	7.3321	7.2043	7.1082	7.0334
0.70	8.8090	8.1331	7.9415	7.7982	7.6870
0.75	10.4891	9.3397	9.0263	8.7953	8.6181
0.80	13.7683	11.4021	10.8121	10.3910	10.0755
0.85	23.2677	15.8289	14.3767	13.4150	12.7312
0.90	526.7141	32.7260	25.3298	21.5480	19.2514

7 Conclusion

This chapter analyzes the failures of service systems by vacation queueing models. Specifically, when a system can encounter only one type of failure at any random time, including when the system is idle, it can be modeled by a random vacation queueing system. This differs from the traditional single vacation and multiple vacation queueing models because in the traditional vacation schemes, the server goes on vacation when there is no customer in the system. However, in the random vacation model, the server can go on vacation at any time when the system fails.

When a system can suffer either a hard failure or a soft failure at any time, we model it by a differentiated random vacation queueing system. This is also different from the traditional differentiated vacation queueing system where vacations are taken when there is no customer in the system.

The random vacation models assume that there is neither customer renegeing nor customer balking when the system is down. They also assume that when the system is down the service rate is zero. However, there are systems that provide partial customer service in the downstate. The versatility of the state aggregation method is demonstrated in its ease of use in the analysis of the models.

References

1. Cooper RB (1970) Queues served in cyclic order: waiting times. *Bell Syst Techn J* 49:399–413
2. Levy Y, Yechiali U (1975) Utilization of idle time in an M/G/1 queueing system. *Manage Sci* 22:202–211
3. Doshi BT (1986) Queueing systems with vacations, a survey. *Queueing Syst* 1:29–66
4. Takagi H (1991) Queueing analysis: a foundation of performance analysis, volume 1: vacation and priority systems, part 1. Elsevier Science Publishers B.V., Amsterdam
5. Tian N, Zhang G (2006) Vacation queueing models: theory and applications. Springer, New York
6. Ibe OC, Howe RC, Trivedi KS (1989) Approximate availability analysis of VAXcluster systems. *IEEE Trans Reliab* 38:146–152
7. Neuts NF (1981) Matrix geometric solutions in stochastic model. Johns Hopkins University Press, Baltimore
8. Ibe OC, Maruyama K (1985) An approximation method for a class of queueing systems. *Perform Eval* 5:15–27
9. Feinberg BN, Chiu SS (1987) A method to calculate steady-state distributions of large Markov chains by aggregating states. *Oper Res* 35:282–290
10. Heathcote CR (1961) Preemptive priority queueing. *Biometrika* 48:57–63
11. Wolff RW (1982) Poisson arrivals see time averages. *Oper Res* 30:223–230
12. Ibe OC (2014) Fundamentals of applied probability and random processes, 2nd edn. Elsevier Academic Press, Waltham, MA
13. White H, Christie LS (1958) Queueing with preemptive priorities or with breakdown. *Oper Res* 6:79–95
14. Miller RG (1960) Priority queues. *Ann Math Stat* 31:86–103
15. Gaver DP Jr (1962) A waiting line with interrupted service, including priorities. *J R Stat Soc Ser B* 24:73–90

16. Avi-Itzhak B, Naor P (1963) Some queueing problems with the service station subject to breakdown. *Oper Res* 11:303–320
17. Mitrany IL, Avi-Itzhak B (1968) A many-server queue with service interruptions. *Oper Res* 16:628–638
18. Ibe OC, Isijola OA (2014) M/M/1 multiple vacation queueing systems with differentiated vacations. *Model Simul Eng* 2014(158247)
19. Servi LD, Finn SG (2002) M/M/1 queue with working vacations (M/M/1/WV). *Perform Eval* 50:41–52

Part IV
Software Simulation, Testing, Workloads,
Aging, Reliability, and Resilience

Combined Simulation and Testing Based on Standard UML Models

Vitali Schneider, Anna Deitsch, Winfried Dulz and Reinhard German

Abstract The development of complex software and embedded systems is usually composed of a series of design, implementation, and testing phases. Challenged by their continuously increasing complexity and high-performance requirements, model-driven development approaches are gaining in popularity. Modeling languages like UML (Unified Modeling Language) cope with the system complexity and also allow for advanced analysis and validation methods. The approach of Test-driven Agile Simulation (TAS) combines novel model-based simulation and testing techniques in order to achieve an improved overall quality during the development process. Thus, the TAS approach enables the simulation of a modeled system and the simulated execution of test cases, such that both system and test models can mutually be validated at early design stages prior to expensive implementation and testing on real hardware. By executing system specifications in a simulation environment, the TAS approach also supports a cheap and agile technique for quantitative assessments and performance estimates to identify system bottlenecks and for system improvements at different abstraction levels. In this chapter we will present the current status of the TAS approach, a software tool realization based on the Eclipse RCP, and a detailed example from the image processing domain illustrating the methodology.

V. Schneider (✉) · A. Deitsch · W. Dulz · R. German
Computer Science 7, Friedrich-Alexander-University of Erlangen-Nürnberg,
Martensstr. 3, D-91058 Erlangen, Germany
e-mail: vitali.schneider@fau.de

A. Deitsch
e-mail: anna.deitsch@fau.de

W. Dulz
e-mail: dulz@cs.fau.de

R. German
e-mail: german@cs.fau.de

1 Introduction

Rapid and efficient development of complex hardware and software systems for telecommunication, automotive, or medical applications needs support from dedicated tool chains and customized modeling environments. Model-driven engineering (MDE) [1] is a promising approach to address the complexity that is inherent in each technical system. MDE is combining two technologies that may help to overcome the complexity hurdle:

- Domain-specific modeling languages (DSML) focus on particular application domains like automotive or telecommunications.
- Transformation and generation mechanisms support the analysis of specific model artifacts in order to generate simulation or source code and alternative model representations. An automated transformation process also ensures the consistency between application development and the assurance of functional and extra-functional requirements like timing aspects, reliability, or performance issues captured by model artifacts.

By providing different abstraction levels and distinct model types, which are standardized by the Object Management Group (OMG) in the Unified Modeling Language (UML) [2], the OMG Model-Driven Architecture (MDA) [3] offers basic concepts and techniques to specify a system independently of the platform that supports it:

1. Starting from a Computation Independent Model (CIM) that specifies the system functionality without showing constructional details,
2. a Platform-Independent Model (PIM) is derived by adding architectural knowledge, at the same time hiding details of the platform used.
3. Finally, a Platform-Specific Model (PSM) arises when all elements and services are added that complete the target platform.

The main advantage of having different views of the same system (see Fig. 1) is to include domain and business experts, as well as software architects and IT

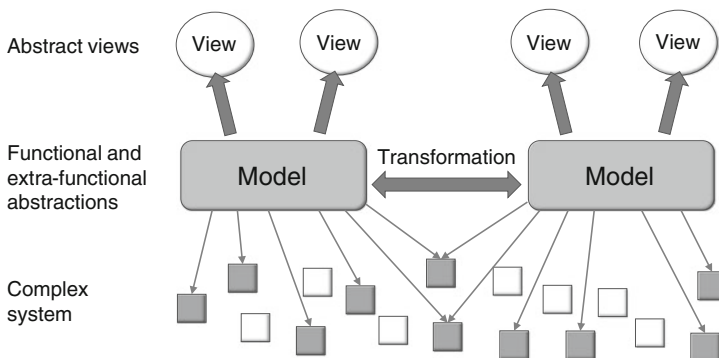


Fig. 1 Relationship between views, models, and system (Reproduced from [1])

specialists in the overall development process, whereby dedicated models focus on different tasks. In an iterative modeling process model transformations support the refinement of the models up to a formal, precise, and detailed specification that will be the basis for implementing and testing the system.

In order to handle specific issues, OMG provides standardized UML profiles that can be combined with normal UML models. For example, the profile for System Modeling Language (SysML) [4] enables modeling of requirements for embedded systems in greater detail. Extra-functional properties, time, and analysis details are expressed by the profile for Modeling and Analysis of Real-time and Embedded systems (MARTE) [5] profile, while UML Testing Profile (UTP) [6] is used to consider testing aspects.

Test-driven Agile Simulation (TAS) [7] intends to improve the overall quality of the development process by combining UML-based engineering, simulation, and testing techniques [8]. The TAS approach assists the transformation of specification models to executable simulation code and uses standardized model-to-text transformation methods. By simulating a given system and running tests on it, TAS provides an agile technique to validate specification models at an early stage of the development process. To express extra-functional requirements and testing details for embedded systems, TAS offers the possibility to integrate model extensions that conform to the SysML, MARTE, and UTP profiles.

This practice is also applied in recent industrial design and development processes, and was evaluated in the European research project COMPLEX [9] for building complex systems. In contrast to the methodology used in COMPLEX, the TAS approach concentrates the verification and validation (V&V) activities on simulation and testing and also involves the UTP profile.

Because UML profiles may handle the same modeling aspect in different ways, model interferences and inconsistencies may appear. For instance, SysML and MARTE provide quite different approaches for the specification of quantitative values, e.g., for time or duration. We therefore showed in a recent paper how to avoid model interferences for test-driven agile simulations based on standardized UML profiles [10].

Avoiding model interferences was also a topic that has been tackled within the EU funded MADES [11] project. Using the MADES language it is possible to restrict SysML and MARTE profiles to a consistent model subset. V&V activities during the development cycle focus on analyzing temporal properties of the components instead of testing. Hence, validation in the context of MADES is mostly time-related.

Since the benefits of standardized graphical languages and simulation-based testing have been recognized, several efforts have been undertaken in that area. In this context, mention can be made of the tool environments Matlab/Simulink¹ or SCADE² that are often used in practice for the development of embedded systems. Both also offer solutions for the integration of UML-/SysML-based modeling techniques within a combined simulation and test environment. However, the tools used

¹<http://mathworks.com/products>.

²<http://www.esterele-technologies.com/products>.

are still very much tied to their own proprietary modeling languages. The notations used in this case primarily support synchronous data flow-oriented paradigms. Thus, both tool environments are rather more suitable for the development of control-oriented systems. In contrast, TAS is seamlessly based on the standardized UML as a more expressive and flexible common modeling language, which can also consider nondeterministic, asynchronous systems.

Furthermore, numerous efforts have been considered so far to develop techniques for deriving quality of service (QoS) properties from UML specifications using diverse analytical techniques. For instance, in [12] the authors propose a framework to transform UML models that are annotated, among others, with stereotypes from the MARTE profile to Stochastic Reward Nets (SRNs). These SRNs are evaluated with the software package SHARPE³ [13] to obtain performability results. In situations where the complexity of the system specification allows the application of analytical techniques, these results will certainly provide a valuable insight in the system under development. In contrast, however, the TAS approach aims on the simulation-based, test-driven development and evaluation of complex systems.

In the following sections, we provide a modeling methodology for the TAS approach by combining standardized UML profiles. We focus on strategies that will avoid model interferences caused by overlapping specification parts that are described by different UML profiles. We also show how tracing of functional and extra-functional requirements can be achieved in the SimTAny tool environment.

2 Test-driven Agile Simulation

In this section, we introduce the concept of TAS. We start with the motivation for the suggested approach, describing its main idea and the basic concept. Then, we outline the most important features covered by TAS.

2.1 *Idea and Concept of TAS*

The development process of complex software and embedded systems usually consists of a series of design, implementation, and testing phases, aligned to some formal process model. Despite a large number of different process models, the development typically starts with requirement definition followed by several specification, programming, and testing steps. Due to a continuously increasing complexity of systems, the approaches based on formal modeling languages like UML are gaining in popularity. On one side, modeling with graphical diagrams helps to deal with the complexity. On the other side, formal specifications enable automated derivation of the implementation code as well as of advanced analysis and validation capabilities.

³<http://sharpe.pratt.duke.edu>.

With our TAS approach, initially introduced in [7], we propagate the combination of model-driven simulation and testing techniques to achieve an improved overall quality during the development process. Thus, our approach enables to derive executable simulations from UML-based specifications of both the system and test models in order to analyze a modeled system and to perform simulated tests on it at early stages of the development process. This approach supports a cheap and agile technique for design error detection as well as for first quantitative assessments and performance estimates. Even prior to expensive implementation and testing on a real system, potential drawbacks and bottlenecks in the system can be identified by the use of simulation. By means of simulation it is also easily possible to investigate and compare alternative designs and solutions at the level of models. Furthermore, the early validation of the specification models helps to reduce development risks. In order to achieve mutual validation of the system and test specifications, we suggest starting with the specification of requirements and then to derive system and test specifications independently and in parallel to each other from these common requirements (see Fig. 2).

Solely based on UML and using its standardized extension profiles, the TAS approach provides for the application of one common modeling language for different design stages like requirements, system, simulation, and test design. Amongst several obvious advantages of having a common modeling language, like simplifying the communication between team members of different disciplines, ability to use only one modeling tool, and cost savings, it also contributes to the quality improvement. Thus, the traceability between relevant model parts can be easily created and examined. Furthermore, in distributed development teams and processes it is particularly important to ensure a joint understanding of definitions and relations across different

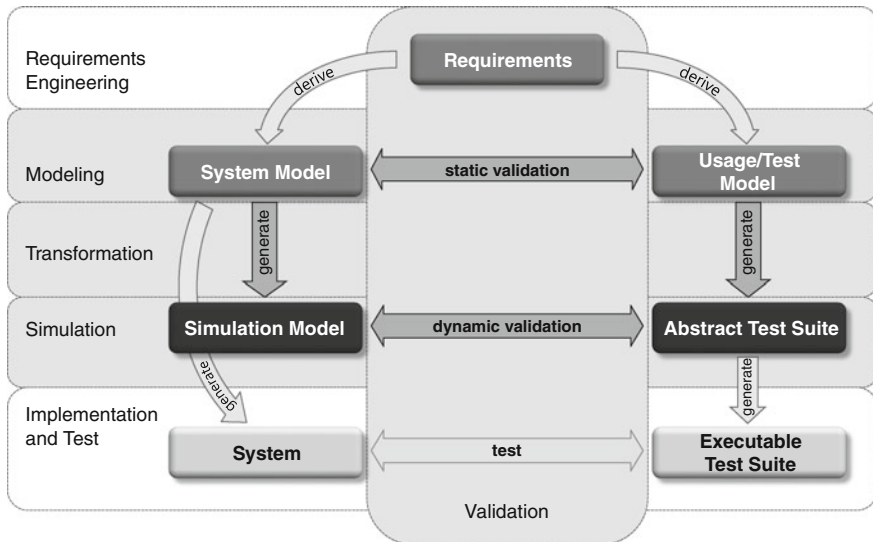


Fig. 2 Concept of test-driven agile simulation (Reproduced from [14])

disciplines. Although specialized tools and languages have been established for each discipline, it is nevertheless possible to use common UML-based specifications as central documents shared during the development process, for instance, by applying integration and transformation solutions as provided by ModelBus.⁴

2.2 Main Features

The TAS approach makes provisions for various aspects of the model-driven development process. Among others, it covers modeling, model-based validation, simulation code generation, and test as well as integrability and extendability for different domains and development environments. In the following, we will shortly introduce some of the main features of TAS (see also Fig. 2).

Modeling

As already mentioned, our approach is widely based on UML as a common modeling language. Due to its general nature, UML is principally suitable for modeling in several development stages, like requirements, system, simulation, and test modeling, which are addressed by the TAS approach. However, for specifications of domain-specific aspects we propose a UML-based modeling methodology, which utilizes several standardized UML extension profiles. As will be shown in detail in Sect. 3, we apply a combined subset of OMG's SysML, MARTE, and UTP profiles. In particular, we use basic elements and diagrams of UML and apply SysML to specify requirements, system blocks, port directions, and traces between requirements and other model elements. A number of stereotypes of several sub-profiles of MARTE are used, for example, to characterize analysis aspects and extra-functional properties, to introduce nondeterminism, and to describe HW/SW allocation. UTP profile is utilized to represent parts of the test model, like test contexts, test components, and test cases. Additionally, our modeling methodology allows for use of the textual action specification language ALF [15] and its library, which provides useful collection types and operations.

However, the combination of different specialized profiles involves special challenges caused primarily by a number of semantic and syntactic overlappings between different profiles. To overcome these challenges, we suggest a strategy of selective combination of proper subsets of profiles presented in our previous work [10].

Verification and Validation

In order to improve the quality of a developed system and to increase the efficiency of the development process itself, it is reasonable to perform verification and validation activities as soon as possible. Our TAS approach enables validation of model-based specifications at very early stages of the design phase. In addition to the known verification and model checking methods, independent formal system and test models derived from the same requirements can serve for their mutual validation.

⁴<http://www.modelbus.org>.

At this point, our approach distinguishes between static and dynamic validation. Static validation can be applied directly on the specification models of both the system and the test models. It consists of examination of constraints, which relates to the static modeling aspects like structure, naming, constraints definitions and traceability to requirements.

On the other side, dynamic validation aims to inspect the dynamic behavior of the modeled system and its corresponding test model. On the level of models the behavior can be validated by executing test cases from the test model on the simulated system model.

Transformation to Simulation Code

Since simulation plays an important role for validation and system's behavior analysis in TAS, our approach assists an automated transformation of the specification models to the executable simulation code. It consists of generation of the simulation code from system models as well as from test models. Thereby, in the latter case we speak about abstract test suites to differentiate the simulated test suites from those derived for real tests on the implemented system.

Referring to the OMG standard for model-to-text transformations MOFM2T [16], a standardized transformation method can be applied to transform structural, behavioral, and analytical elements from UML models to appropriate representations in the desired simulation environment. The definition of such compatible mappings poses the biggest challenge on this stage.

Above all, of course, discrete-event simulation tools are eligible for the purpose of simulation due to the original time-discrete nature of UML and of most computing systems. Thus, the focus of our approach is primarily on supporting the transformation of time-discrete models for discrete-event simulators (see Sect. 4.3). Nevertheless, with some limitations time-continuous aspects and systems may also be represented with SysML and transformed to appropriate simulation tools, as shown for instance in [17].

Simulation

On one side, the simulation code derived from a system model can be utilized for design and performance analysis of the whole system. The generated simulation represents in this case all active components of the system with their reproduced behaviors. Running a simulation of the system one can first investigate its dynamic behavior. Depending on the simulation tool, even interactive or stepwise execution could be possible, which is particularly helpful for debugging. At the end of or even during the simulation, predefined analytical values could be assessed. By modeling different design solutions as specific parameter configurations, simulation tools can provide support for parameter variation and searching for optimal solutions with respect to the specified requirements.

Test

On the other side, abstract test suites generated from a test model serve for the simulated execution of tests. A test suite corresponds at the model level to the UTP's test context, which describes the configuration of a test consisting of a system under test (SuT) with surrounding test components and contains a number of test cases.

Depending on the type of a test, i.e., unit, integration, or system test, the SuT may either be one system component, a set of components, or the whole system, respectively. A test case is thereby usually represented as a sequence of interactions between a SuT and test components.

After the transformation to a simulation code an abstract test suite largely consists of simulated test components, which can create stimuli for the SuT and evaluate its responses by comparing them with the expected results. While the behavior of test components is simulated according to the currently executed test case, the behavior of the SuT is coming from the embedded simulation code generated from the system model. It is the task of each test component to determine its local verdict according to the expected responses from the SuT. The verdict of a test case is then composed of local verdicts of all containing test components. A failed or inconclusive test case first indicates some inconsistencies in the system or test model, or event in the original requirements model and requires a closer inspection of these models.

Analysis

The output provided by the simulation runs helps to assess different design solutions or to predict the performance of a developed system. However, a comprehensive statistical analysis of simulation results is often also needed. In order to facilitate this task, our approach provides support for convenient calculation and visualization of some basic measures.

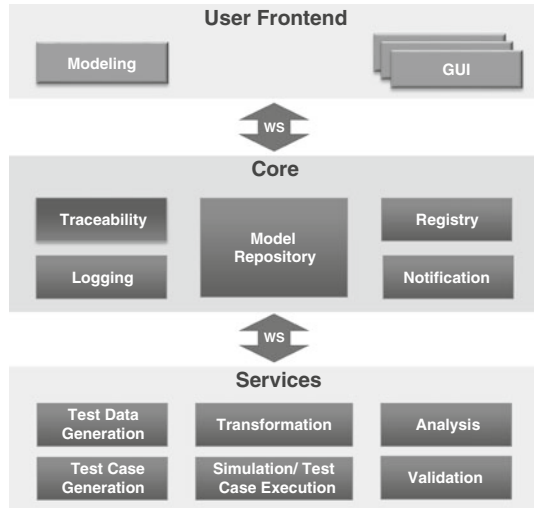
Traceability

Traceability information about relationships between the requirements, the system and test design, and their implementations is quite crucial for the development of complex systems, since the traceability analysis can help to improve the development process enormously. In general, traceability information helps to determine the impact of an element on the other specification parts. Furthermore, traceability analysis can provide coverage and traceability metrics to assist in localizing gaps or inconsistencies in complex specifications.

Using SysML, requirements of the system to be developed can be represented in UML models. They can be either directly defined in the model or imported from specialized requirement management tools. Furthermore, SysML provides special association types to define traceability links between requirements among themselves or between requirements and elements that are determined to realize or to verify requirements.

In order to maintain an overview of the many links and to advance traceability analysis, the TAS approach summarizes traceability information in a dedicated model. Such a traceability model largely includes references (traces) to the relevant elements from the requirements, system, and test model. In addition, this model is enriched with traceability links to the artifacts, like implementation or simulation code, derived from the specification models. Our traceability model allows for clear visualization of the traceability information and for easier navigation across different modeling domains. Furthermore, traceability metrics can be easily calculated with this model and potential gaps like unsatisfied or untested requirements can be easily identified.

Fig. 3 Service-oriented architecture for TAS (Reproduced from [14])



Service-Oriented Architecture

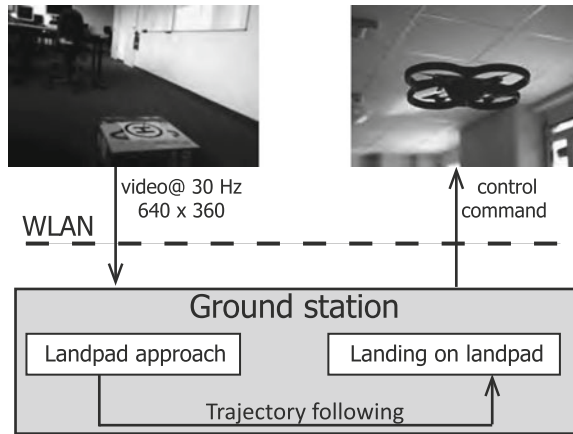
The previously outlined features of the TAS approach require a broad tool support for modeling, model transformations, simulation, and analysis. Of course, it is desirable to have one integrated tool environment on a single workstation. For large distributed processes or due to performance reasons it is, however, often required to source out some designated functionality or rather to make it accessible as services over the network. For instance, the simulation of large, complex models could be very time and resource consuming or could require special system requirements.

Furthermore, to enable integration of the TAS approach into existing development environments, the interoperability and loose coupling of heterogeneous tools via ubiquitous standards is needed. Therefore, in [14] we suggested a service-oriented architecture design [18] for the TAS approach (see also Fig. 3). The core of this design consists of a central repository with several common services, for example, services for registry, notification, or logging. The additional tools or components, provided for TAS, can be either realized as external services or are even part of the user front end. The communication between individual components is realized using open standards and well-defined Web service interfaces.

3 The TAS Modeling Methodology for Image Processing Systems

In this section, we illustrate how our modeling approach can be used for the design of an image processing system. It should be mentioned that we have reported this possible utilization of the TAS approach for the image processing domain in our

Fig. 4 Monocular vision system for the autonomous approach and landing using a low-cost micro aerial vehicle (MAV) system



previous works [19], but we now look at a concrete example. The system, which is shown in Fig. 4, enables an off-the-shelf Parrot AR.Drone 2.0 low-budget quadrotor micro aerial vehicle (MAV) to autonomously detect a typical helicopter landpad, approach it, and land on it. To fly toward the landpad while accurately following a trajectory, monocular simultaneous localization and mapping (SLAM) have been used [20].

The workflow in this application is based on the monocular, stereo, and RGB-D cameras as the main sensors and consists of the following steps: (1) exploitation of the geometric properties of the circular landpad marker and detection of the landpad; (2) determination of the exact flight distance between the quadrotor and the landpad spot; (3) moving toward the landpad by means of monocular simultaneous localization and mapping (SLAM); and (4) landing on the landpad. Development and hardware details related to this application have been illustrated in [20].

The Parrot AR.Drone 2.0 is a low-cost quadrotor with a simple IMU and two monocular cameras. The quadrotor features are 1 GHz, ARM Cortex-A8 processor, 32-bit 800 MHz DSP, and 1 GB of DDR2 RAM 200 MHz. Due to the complexity of the computational tasks, the above-mentioned workflow cannot be performed directly on-board. Therefore, the quadrotor communicates with ground station through wireless LAN. The ground station receives video data, performs the computations, and sends the generated steering commands back. In the example system, there are significant delays in the communication between the quadrotor and the ground station, as all the computations are performed externally. By means of simulation at the level of models, we aim to investigate and to compare alternative designs and solutions for the described system.

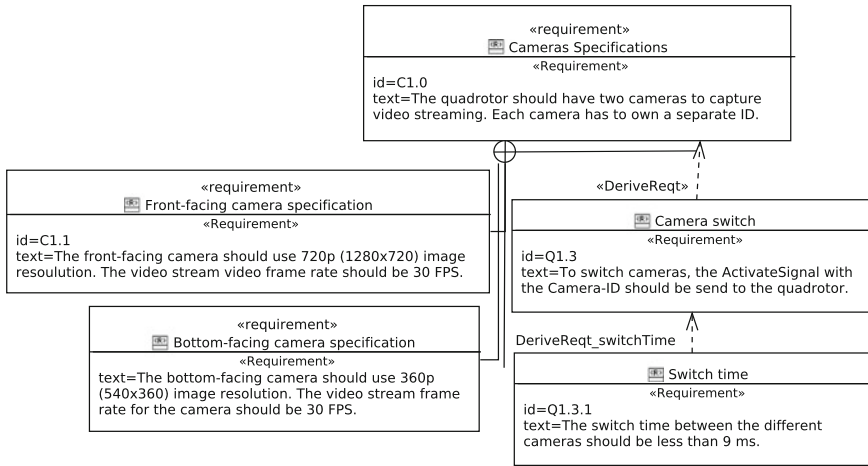


Fig. 5 Simple SysML requirement model for the video stream control systems

3.1 Requirements Modeling

In the scope of TAS, the design specification starts with SysML-based modeling, which involves the initial requirement specification. We have analyzed fundamental requirements on the hardware and software architectures for the example system. The requirements captured in text are represented and clustered directly in the model by means of the SysML requirements and package diagrams. Main requirements, which need to be considered to achieve a safe flying system with the minimum functionality including the video stream control systems, are illustrated in Fig. 5.

The *Camera* requirement includes the description of the quadcopt’s cameras: a HD (1280*720) 30fps front-facing camera and a QVGA (320*240) 60fps bottom-facing camera. During the video streaming, it shall be possible to switch between two cameras in a short time (see Fig. 5).

3.2 Structure Modeling

Based on the specified requirements, the system model as well as the test model can be created independent from each other in order to ensure their utilization for mutual validation (see in Sect. 2.1). The aim of this specification phase is to design the system architecture in terms of functional blocks. The functional and behavior aspects of the block can be expressed using SysML block definition, internal block, and state machine diagrams. Figure 6 shows the main block definition diagram from our example system. In addition to SysML concepts, the diagram will include a set of MARTE concepts in order to specify a context in which the system should be analyzed. We

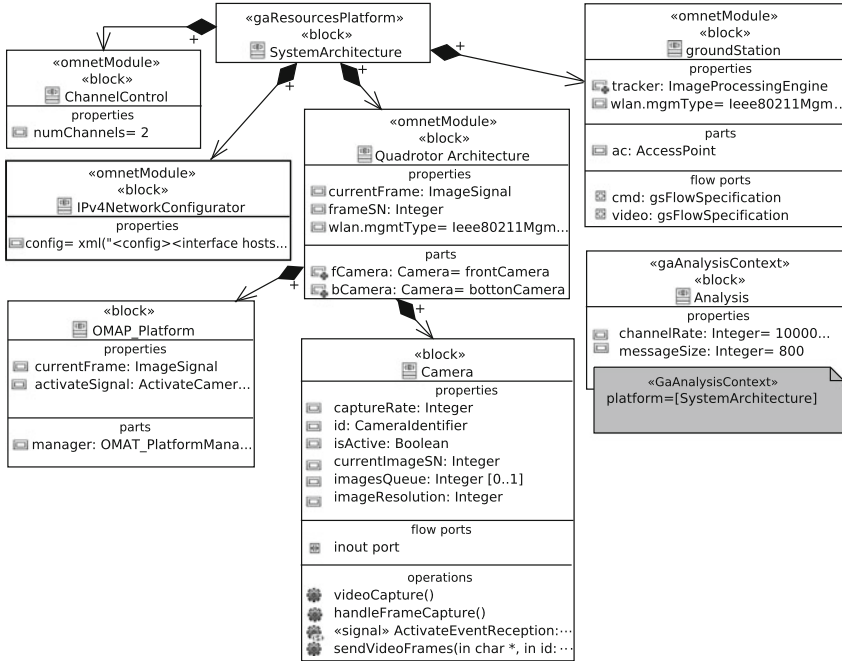


Fig. 6 Example of a description of system components and their relationships using SysML block definition diagram

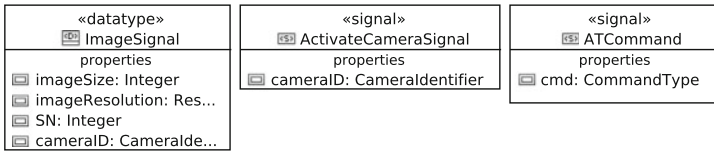


Fig. 7 Example of data type and signal definitions

use the stereotype *GaAnalysisContext* applied to a separate block to specify which of the blocks in the model should be analyzed. The platform attribute points to a *SystemArchitecture* block that has a *GaResourcesPlatform* stereotype applied. It represents a logical container for the resources used in the analysis context. All system components nested in this block will be simulated. The *GaAnalysisContext* block can also include input and output parameters for the simulation, which are specified with MARTE stereotype *Var*.

Figure 7 shows the signals in our model. The signal *ATcommand* is used to manage the quadrotor during the flight. *ImageSignal* encapsulates application-level data like images of the video stream for the transmission over the channel. *Activate CameraSignal* is used to switch the camera during the video stream.

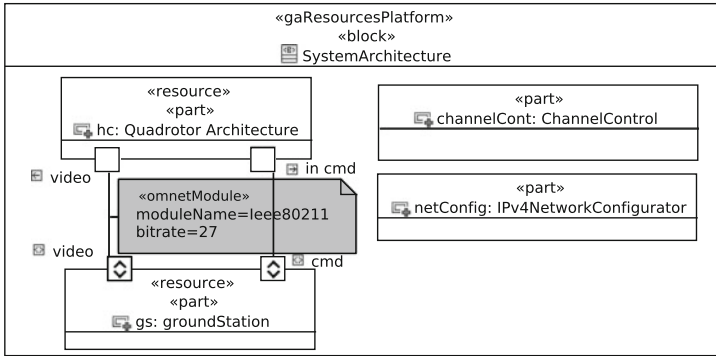


Fig. 8 Composite structure for the system

The SysML internal block diagrams describe the internal structure of a block. We use the SysML internal block diagram to model the system architecture. Figure 8 shows the internal structure of the *SystemArchitecture* block from our model. The system consists of a *Quadrotor Architecture* and a *groundStation*. *ChannelControl* is required for wireless simulation. It represents a system component that keeps track of which nodes are within interference distance of other nodes. *IPv4NetworkConfigurator* component assigns IP addresses and sets up static routing for an IPv4 network. Using the SysML concepts of ports and the connectors, it is possible to express communication links between various components.

3.3 Behavior Modeling

To model system behavior at a high level of abstraction, we primary use UML state diagrams. The state machines representing the image processing algorithm for the detection of the landpad in our model are shown in Figs. 9 and 10.

Figure 9 shows the top-level behavior of the *groundStation* block. As the state name *wait for video stream* indicates, the ground station waits in this state until it receives an *ImageSignal* message. This is the trigger of the only transition leaving from the *wait for video stream* state to the *landpad detection* state. The transition has applied a MARTE stereotype *GaStep* that indicates subsequent occurrences of the *ImageSignal* event which are of interest. The *do* behavior of the *landpad detection* state invokes the *detectionStateMachine* behavior.

Figure 10 shows the ground station behavior for the detection of the landpad. The main transition is triggered by the reception of the *Image* signal. The algorithm consists of the following main steps: (1) edge detection and (2) landpad matching. In the first step, we detect the edges in the image. After that, we group detected edges to a curve and check whether it contains the letter “H” [20], which indicates the landpad. Once the letter is detected, the *NotificationMsg* must be sent out to the quadrotor and the state machine moves into the final state.

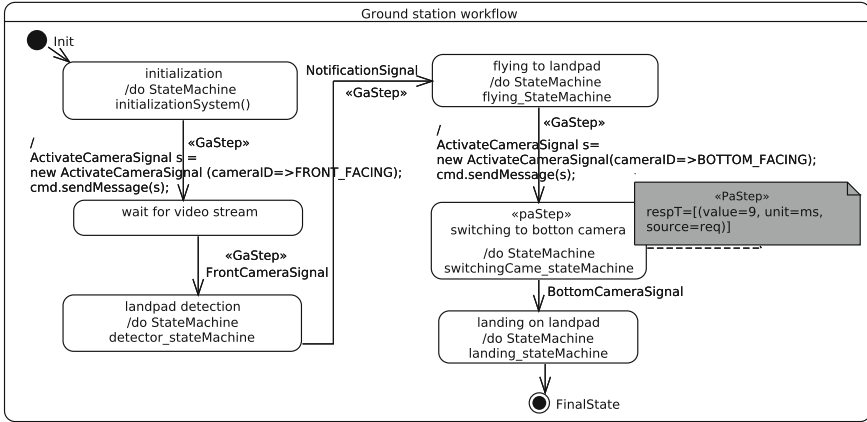


Fig. 9 State machine of the ground station

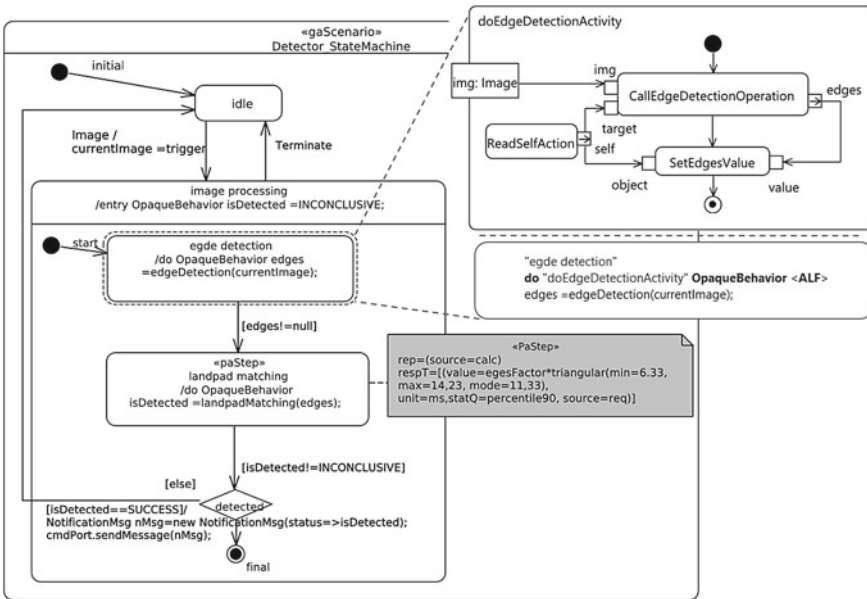


Fig. 10 System behavior building using SysML and ALF syntaxes

As exemplarily shown in Fig. 10 for the *do* activity of the state *edge detection*, we can define detailed behaviors either using standard UML behavior diagrams, for instance, like activity diagrams, or using a more compact high-level action language ALF [15].

In order to express performance attributes of behavior steps, we can apply the MARTE’s *PaStep* stereotype. For example, we specify that the response time of

calling the landpad matching activity is required to match the given distribution. To collect the statistics of interest during the simulation, one just have to apply an additional MARTE expression *source=calc* on the corresponding property value. In our example, we do so for *rep* and *respT* properties of *PaStep* stereotype on *landpad matching* state to determine the number of repetitions and response time. The collected data can then be analyzed for comparison with the expected result by utilizing the analysis capabilities of our framework as it will be shown later in Sect. 4.4.

3.4 Test Modeling

As a counterpart to the system modeling presented in the previous sections, the modeling of test specifications can also be performed in UML. Thereby, quite similar modeling paradigms can be applied as used for system modeling. Based on common requirements a test designer has to specify proper tests for subsequent validation of the system specification and later of its implementation. The purpose of these tests is to determine whether the system satisfies the requirements.

To provide a complete test specification, one has first to define the context of a test identifying the SuT and required test components. Figure 11 shows an example of a test context provided to test the behavior of the quadrotor component of our

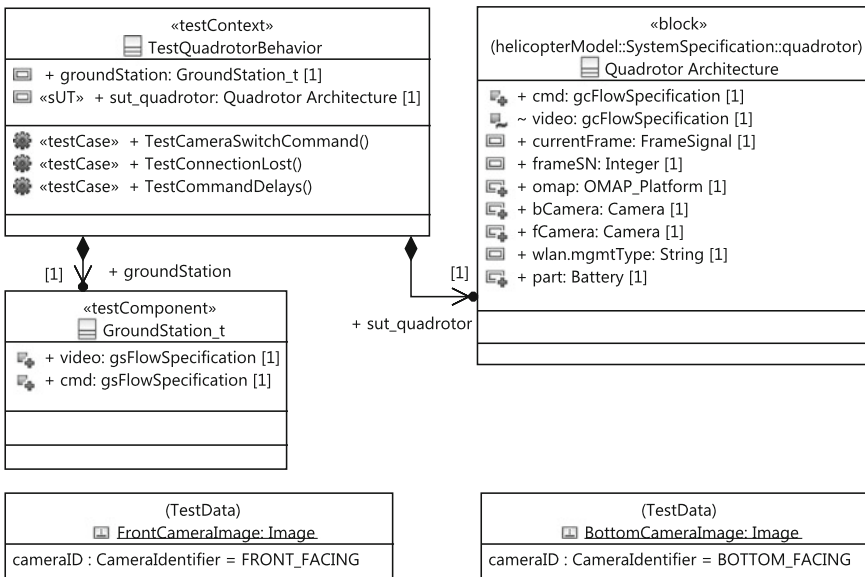


Fig. 11 Example of a test context definition

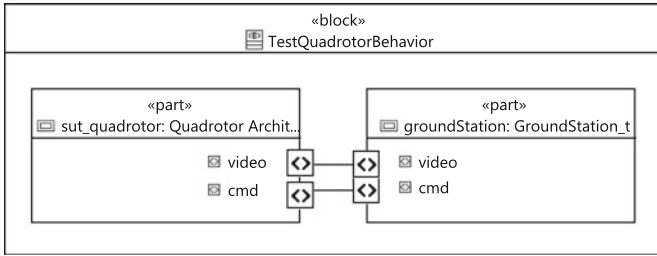


Fig. 12 Internal block diagram of a test context

example system. As already mentioned, we apply stereotypes of the UTP profile in order to declare test relevant aspects.

Thus, using a SysML block definition diagram or UML class diagram, a test context can be modeled as a structured block with the applied *TestContext* stereotype. A test context consists of test components, SuT, and serves at the same time as container for test cases. Whereas test components are simple blocks or classes containing ports for communication and declared in the test model using the stereotype *TestComponent*, SuT represents a block of the system specification model with its own behavior. To identify the SuT in a test context, the property referencing the SuT is marked with the *SuT* stereotype of UTP. The internal structure of the test context, which defines connections between test components and SuT, can be represented by the SysML internal block diagram, as shown in Fig. 12.

The behavior of test components is individually specified for each test case of the owning context. Therefore, using UML sequence diagrams a test case can be

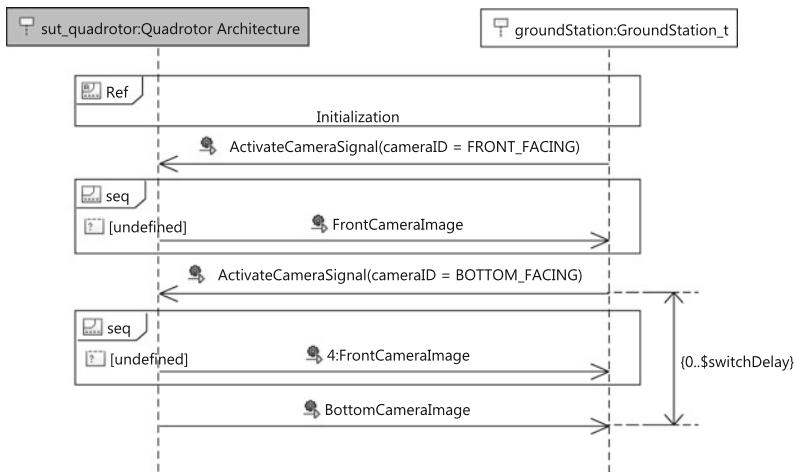


Fig. 13 Test case specification using UML sequence diagram

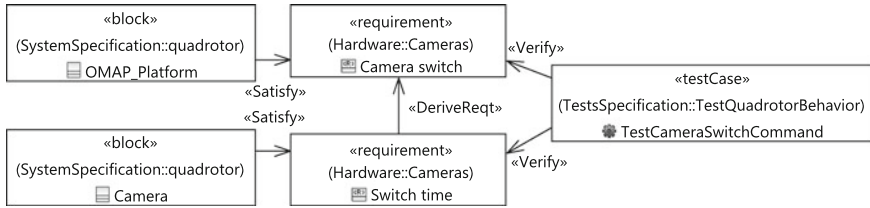


Fig. 14 Modeling of traceability links to requirements from satisfying or verifying elements

defined in every detail. As shown in Fig. 13, a test case is represented as a sequence of messages between the test component *groundStation* and the SuT *sut_quadrotor*. This test case, for instance, checks whether the quadrotor is able to activate its front camera and switch to the bottom camera appropriately after receiving appropriate camera activation command messages from the ground station.

3.5 Traceability Modeling

Although it is not the case in the modeling example presented, in a strict requirement-driven development process, nearly all system components and test cases shall relate to corresponding requirements. To express these relations, SysML provides special association links that can be assigned between requirements and other modeling elements. In Fig. 14 we show an example of traceability links defined for requirements regarding camera switching of the quadrotor. As shown in the figure, elements exist in the system specification, which satisfy the requirements and at least one test case in the test model, which verifies them. Currently, an engineer has to specify and manage most traceability links manually. However, additional tool support for automatic generation of traceability links while deriving model elements from requirements, for instance, is certainly possible and is part of the ongoing work.

4 SimTAny Framework

The main features of the suggested TAS approach are widely supported by the framework SimTAny (formerly introduced in [7] as ‘VeriTAS’) that will be further extended. SimTAny integrates relevant tools with newly developed components in a common environment based on the popular Eclipse RCP⁵ platform. Among others, we utilize a UML modeling tool, a transformation framework, a simulation engine, and an analysis tool (see Fig. 15). In the following, we will depict the main features of our framework and describe their realization in some more details.

⁵http://wiki.eclipse.org/Rich_Client_Platform.

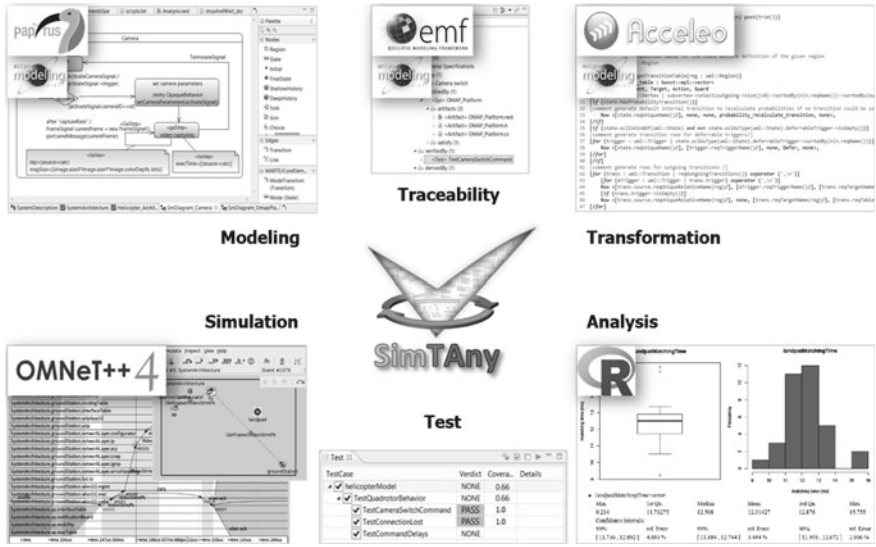


Fig. 15 Overview of tools integrated in the SimTAny framework

4.1 Modeling

In order to support extensive modeling capabilities required for TAS, we apply the open source modeling tool Papyrus,⁶ which is closely integrated with Eclipse. We preferably use Papyrus, since it has been designed to exactly implement the UML specification. All in all, it provides a very advanced modeling support for OMG standards including UML-related SysML and MARTE profiles. Nevertheless, other modeling tools, which can consistently export models into the OMG’ interchange format XMI, can also be used instead of Papyrus.

In order to improve the modeling efficiency, our framework adds some extensions to Papyrus. This primarily concerns the modeling of detailed behaviors and expressions with ALF textual editors. The main reason for supporting ALF’s textual notation is because specifying of a detailed behavior with a higher programming language, in most cases, is much more compact, faster, and intuitive than with standard UML behavior diagrams like state, activity, or sequence diagrams.

As it has been previously illustrated in Fig. 10 in Sect. 3.3, in principle, we can always express detailed activities of a state with activity diagrams, for instance. Because of the excessive complexity and inefficiency of this method, UML also provides the possibility to describe behaviors and expressions with a natural or programming language encapsulated in a so-called *OpaqueBehavior* or *OpaqueExpression* correspondingly. Thus, we apply the standardized action language ALF to directly specify such expressions and behaviors at appropriate places in our models in a more

⁶<http://eclipse.org/papyrus>.

compact and intuitive way. Among others, entry, exit, and do behaviors of a state as well as guard conditions and effects of a transition in state diagrams are predestined for ALF and thus are supported by appropriate text editors in SimTAny.

4.2 *Static Validation and Verification*

The special feature for static validation of system and test models, suggested for the TAS approach, has been realized in our framework by means of the Eclipse EMF validation framework.⁷ In order to achieve static validation and verification of the models, we provide constraints to check, on the one hand, for inconsistencies in each model separately and for compatible relations within the models to each other and to their common requirements, on the other hand. The defects detected in this way by the framework are listed in the Eclipse problems' view. Moreover, affected elements are marked as erroneous in the model editor afterwards. It is further possible to navigate from the problems listed in the view to corresponding elements in the model editor. Although only few simple constraints are currently implemented in SimTAny, the framework can be easily extended by new constraints.

4.3 *Transformation to Simulation Code*

Since the generation of the executable simulation code from UML models is one of the most challenging issues in our approach, a solid methodology with extensive tool support is required to perform this task. That is why we decided to build upon the OMG's standard MOFM2T [16] and its reference implementation, i.e., the Eclipse Acceleo⁸ code generation framework. MOFM2T provides a template-based model-to-text transformation language, where a template is a text containing specially marked areas that have to be generated accessing the elements of the input model. Generally, any kind of text or code for any textual language (C++, Java, Python) can be generated with this method. The framework Acceleo provides a code generation engine along with tools to support the efficient development of code generators. Besides a comprehensive editor with syntax highlighting, error detection, and auto-completion, it assists with a debugger, a profiler, and a traceability API.

With Acceleo we have implemented model-to-text transformation templates for generation of the simulation code (see Fig. 16). In order to demonstrate the feasibility of our approach, we currently generate code that is executable with the simulation engine OMNeT++.⁹ Nevertheless, our transformation module is designed to be

⁷<http://projects.eclipse.org/projects/modeling.emf.validation>.

⁸<http://eclipse.org/acceleo>.

⁹<http://omnetpp.org>.

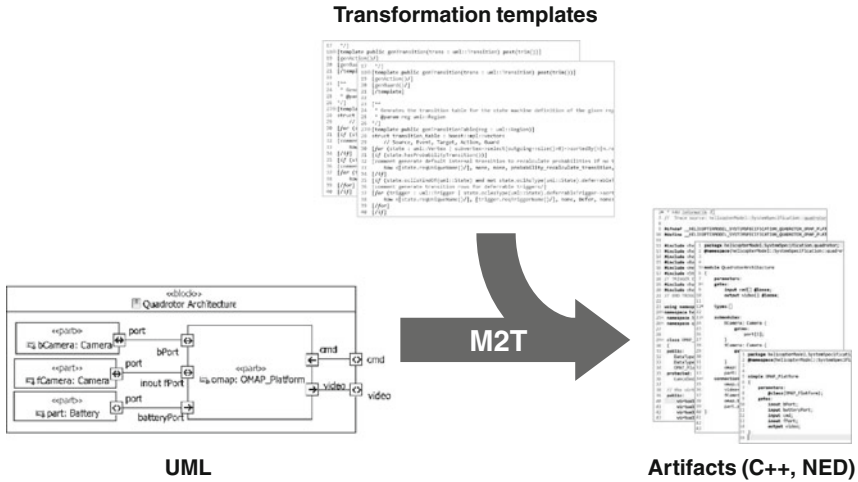


Fig. 16 Template-based model-to-text transformation

extendable for other simulation engines, too. OMNeT++ is an open source discrete-event simulator that is quite popular for simulation of communication networks.

An OMNeT++ simulation project (also called *simulation model*) typically consists of active components (*simple modules*) programmed in C++ that are composed of *compound modules* and *networks* using the OMNeT++'s own NED language. Furthermore, initialization files (ini) are used in OMNeT++ for additional configuration of simulation experiments. Thus, as a result of the model-to-text transformation our framework automatically generates C++, NED, and ini files of the complete simulation model. As shown in Fig. 17 for our use case, the simulation model generated

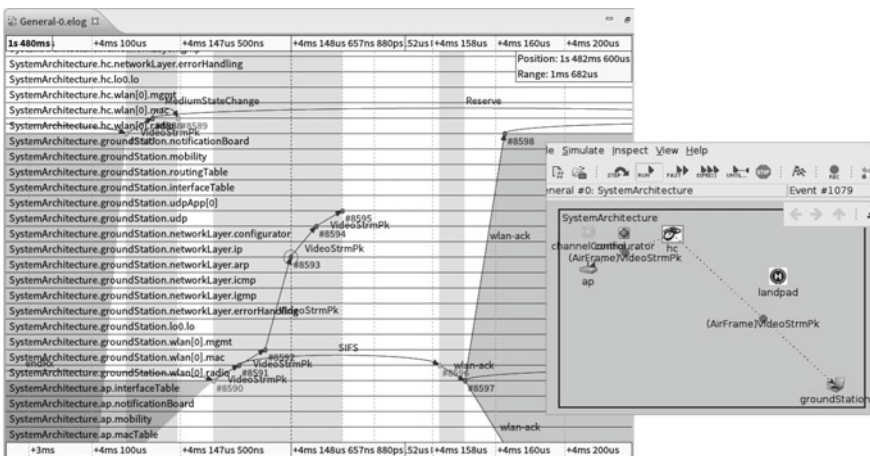


Fig. 17 Simulation with OMNeT++

can then be executed with OMNeT++ to analyze the behavior of the system (during the simulation or afterwards using the event log).

4.4 Analysis

The output data collected during the simulation can be directly analyzed in our framework, in the first instance, without prior external analysis tools being required. Therefore, SimTAny provides a dedicated perspective for analysis where the simulation results can be imported and visualized.

In the background, a very popular environment for statistical computing and graphics, i.e., the R-Project,¹⁰ is applied to generate plots and to calculate statistics of the data. Thus, for instance, the user can obtain an immediate overview about the key statistical measures like the mean, median, deviation, or confidence intervals of a data sample as well as to take a look at its time series, histogram, or box plots. To illustrate this, Fig. 18 shows example plots and statistics generated for landpad matching times observed during the simulation of our quadrotor model.

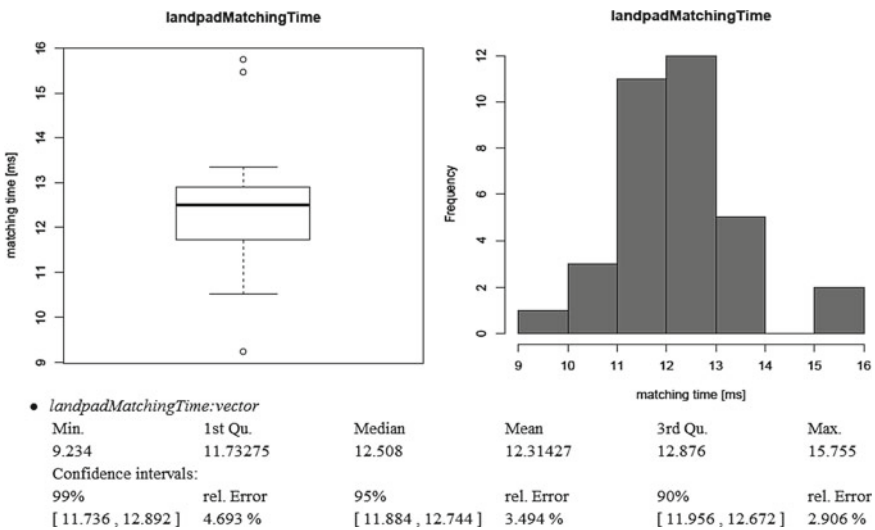
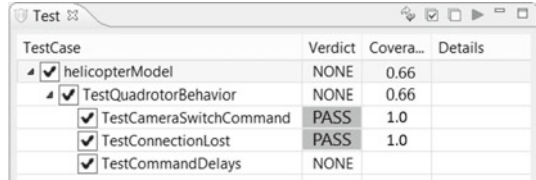


Fig. 18 Analysis of simulation results

¹⁰<http://www.r-project.org>.

Fig. 19 TestView provides an overview of tests contained in a model, control of test case execution, and test verdict



TestCase	Verdict	Covera...	Details
✓ helicopterModel	NONE	0.66	
✓ TestQuadrotorBehavior	NONE	0.66	
✓ TestCameraSwitchCommand	PASS	1.0	
✓ TestConnectionLost	PASS	1.0	
✓ TestCommandDelays	NONE		

4.5 Test

In order to support tests at the level of simulation models as described in Sect. 2, along with the generation of simulation code from the system model our framework provides the generation of executable OMNeT++ simulations for each test case contained in the test model. In Sect. 3.4 we have demonstrated an example for modeling of a test context and a test case. The user can obtain an overview of all test contexts and test cases defined in the model in the dedicated test view (see Fig. 19). Once the model has been transformed to the simulation code, the user can perform the execution of tests from this view. The verdict and eventual error reports of each completed test run are then also accessible in the view.

4.6 Traceability

As already mentioned, our approach provides for a special model aimed to store traceability information. Initially coming from specification models, this traceability information is enriched with the traceability links to artifacts generated during model-to-text transformations. Based on the Eclipse modeling framework EMF,¹¹ the traceability meta model has been developed and integrated in our framework. Instances of this model are created in SimTAny automatically from specification models by performing a model-to-model transformation according to the OMG standard specification (MOF) 2.0 Query/View/Transformation [21] supported by the Eclipse component *QVT Operational*.¹²

Furthermore, using the traceability listeners mechanism provided by Acceleo, SimTAny is able to collect traceability information during the code generation and to add it to the related traceability model instance.

In order to provide a better overview about all available traceability information, SimTAny provides a special traceability view (see Fig. 20). In this view the user can inspect the relationships between elements in both directions starting either from the requirements, from the system or test model elements, or even from the artifacts. The view also provides special filters to show possible deficiencies such as unsatisfied or

¹¹<http://eclipse.org/modeling/emf>.

¹²<http://projects.eclipse.org/projects/modeling.mmt.qvt-oml>.

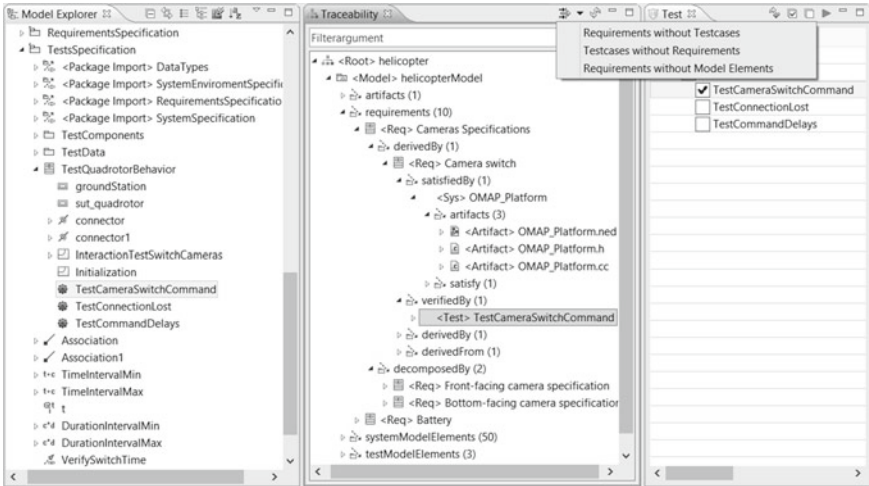


Fig. 20 Traceability view to inspect traceability relationships between different model elements

untested requirements. Thus, for instance, the requirements that do not relate to any test case can be easily detected for additional investigations.

Moreover, as depicted in Fig. 20, SimTAny allows easy switching from the traceability view to other views or editors relevant for further inspection of the selected element. In this manner, a model element occurring in the traceability view can be opened in the model editor, or a test model element can be shown in the corresponding test view.

5 Conclusions

Due to the ever increasing complexity of hardware and software systems, model-driven development methods and tools are gaining popularity as methods to fulfill functional and extra-functional requirements like timing aspects, reliability, or performance issues.

Model-driven engineering technique based on OMG’s UML and the MDA is a promising practice to address the complexity that is inherent in each technical system. The suggested TAS approach improves the overall quality of the development process by combining UML-based engineering, simulation, and testing techniques. TAS also assists the transformation of specification models to executable simulation code and uses MOFM2T, a standardized model-to-text transformation method.

By simulating a given system and running tests on it, TAS provides an agile technique to validate specification models at an early stage of the development process. To express extra-functional requirements and testing details for embedded systems

TAS offers the possibility to integrate model extensions that conform to OMG's SysML, MARTE, and UTP profiles.

In order to integrate the different tasks of the TAS approach we built the domain-independent framework SimTAny, which is based on the Eclipse RCP platform. Amongst other tools, we utilize the UML modeling tool Papyrus, the code generation framework Aceleo, the simulation engine OMNeT++, and the analysis tool R, which provides statistical measures and expressive plots. In addition, the Eclipse modeling framework EMF is used to develop and integrate a traceability meta model in order to store and exchange traceability information between different components.

Last but not least, we have shown how to apply TAS techniques for the development of a concrete image processing system. Here, the main task was the design of a system consisting of a quadrotor and a ground station that exchange image data over a wireless IPv4 channel. Diverse models from different UML profiles are used to specify functional and extra-functional properties of the overall system.

6 Future Work

Our ongoing work consists of proving more detailed hardware and software specifications for image processing systems by further elaboration of the modeling approach presented and by extending the transformation rules for simulation code generation.

Since extensive improvements regarding the UML modeling tools are still required to increase the efficiency of the model-based engineering, one part of our future work will be to make the modeling in the context of the suggested approach more user friendly. Thus, amongst others, to reduce the effort in creating and updating models, dealing with different modeling views, profiles, and a large number of stereotypes, we are developing appropriate extensions within the scope of our tool environment. For the special domain of the modeling of image processing systems, we would like to provide a modeling library, which will contain several predefined elements like typical image operators and data types.

Furthermore, an extensive experiment design framework is being developed to support model-based design and management of simulation experiments. Moreover, in order to allow precise simulation of hardware, we intend to integrate SystemC¹³ in our simulation environment.

References

1. Schmidt DC (2006) Model-driven engineering. IEEE Comput 39(2). <http://www.truststc.org/pubs/30.html>. Accessed Feb 2006
2. Object Management Group (OMG) (2011) UML unified modeling language. <http://omg.org/spec/UML>

¹³<http://www.systemc.org/downloads/standards/systemc>.

3. Object Management Group (OMG) (2003) MDA model driven architecture guide. <http://omg.org/cgi-bin/doc?omg/03-06-01>
4. Object Management Group (OMG) (2012) SysML systems modeling language. <http://omg.org/spec/SysML>
5. Object Management Group (OMG) (2011) UML profile for MARTE modeling and analysis of real-time and embedded systems. <http://omg.org/spec/MARTE>
6. Object Management Group (OMG) UTP: UML testing profile 1.2. <http://omg.org/spec/UTP>
7. Djanatliev A, Dulz W, German R, Schneider V (2011) VeriTAS—a versatile modeling environment for test-driven agile simulation. In: Proceedings of the 2011 winter simulation conference, Phoenix, AZ, USA, Dec 2011, pp 3657–3666
8. Dietrich I, Dressler F, Dulz W, German R (2010) Validating UML simulation models with model-level unit tests. In: Proceedings of the 3rd international ICST conference on simulation tools and techniques (SIMUTools' 10), Malaga, Spain, March 2010
9. Herrera F, Posadas H, Peil P, Villar E, Ferrero F, Valencia R, Palermo G (2014) The COMPLEX methodology for UML/MARTE modeling and design space exploration of embedded systems. *J Syst Archit* 60(1):55–78
10. Schneider V, Yumatova A, Dulz W, German R (2014) How to avoid model interferences for test-driven agile simulation based on standardized UML profiles. In: Proceedings of the symposium on theory of modeling and simulation, Tampa, FL, USA, April 2014, pp 540–545
11. Quadri IR, Indrusiak L, Sadovykh A (2012) MADES: a SysML/MARTE high level methodology for real-time and embedded systems. In: Proceedings of the international conference on embedded realtime software and systems (ERTS2)
12. Khan RH, Machida F, Heegaard PE, Trivedi KS (2012) From UML to SRN: a performability modeling framework considering service components deployment. In: *International journal on advances in networks and services*, vol 5, no c, pp 346–366
13. Trivedi KS, Sahner R (2009) Sharpe at the age of twenty two. *SIGMETRICS Perform Eval Rev* 36(4):52–57
14. Schneider V, German R (2013) Integration of test-driven agile simulation approach in service-oriented tool environment. In: Proceedings of the 46th annual simulation symposium (ANSS'13), San Diego, USA, April 2013
15. Object Management Group (OMG) (2013) Action language for foundational UML (ALF). <http://omg.org/spec/ALF/>
16. Object Management Group (OMG) (2008) MOFM2T MOF model to text transformation language. <http://omg.org/spec/MOFM2T>
17. Johnson TA (2008) Integrating models and simulations of continuous dynamic system behavior into SysML. Master Thesis, Georgia Institute of Technology, Aug 2008
18. Josuttis N (2007) SOA in practice: the art of distributed system design. O'Reilly Media Inc
19. Yumatova A, Schneider V, Dulz W, German R (2014) Test-driven agile simulation for design of image processing system. In: Proceedings of 16th international conference on advances in system testing and validation life cycle (VALID 14), IARIA, Oct 2014
20. Dotenco S, Gallwitz F, Angelopoulou E (2014) Autonomous approach and landing for a low-cost quadrotor using monocular cameras. In: *Computer vision—ECCV, (2014) workshops—Zurich, Switzerland, September 6–7 and 12, 2014, Proceedings, Part I*. Springer International Publishing Switzerland, pp 209–222
21. Object Management Group (OMG) (2011) MOF Query/View/Transformation (QVT). <http://omg.org/spec/QVT>

Workloads in the Clouds

**Maria Carla Calzarossa, Marco L. Della Vedova, Luisa Massari,
Dana Petcu, Momin I.M. Tabash and Daniele Tessera**

Abstract Despite the fast evolution of cloud computing, up to now the characterization of cloud workloads has received little attention. Nevertheless, a deep understanding of their properties and behavior is essential for an effective deployment of cloud technologies and for achieving the desired service levels. While the general principles applied to parallel and distributed systems are still valid, several peculiarities require the attention of both researchers and practitioners. The aim of this chapter is to highlight the most relevant characteristics of cloud workloads as well as identify and discuss the main issues related to their deployment and the gaps that need to be filled.

Keywords Cloud computing · Workload characterization · Monitoring · Resource management · Scheduling · Reliability · Failure

M.C. Calzarossa (✉) · L. Massari · M.I.M. Tabash
Dipartimento di Ingegneria Industriale e dell'Informazione,
Università degli Studi di Pavia, Via Ferrata 5, 27100 Pavia, Italy
e-mail: mcc@unipv.it

L. Massari
e-mail: luisa.massari@unipv.it

M.I.M. Tabash
e-mail: momin.tabash01@ateneopv.it

M.L. Della Vedova · D. Tessera
Dipartimento di Matematica e Fisica, Università Cattolica del Sacro Cuore,
Via Musei 41, 25121 Brescia, Italy
e-mail: marco.dellavedova@unicatt.it

D. Tessera
e-mail: daniele.tessera@unicatt.it

D. Petcu
Departament Informatica, Universitatea de Vest din Timișoara,
Bvd. Vasile Pârvan 4, 300223 Timișoara, Romania
e-mail: petcu@info.uvt.ro

1 Introduction

Cloud technologies are being successfully deployed nowadays in many business and scientific domains, such as e-commerce, e-government, engineering design and analysis, finance, healthcare, web hosting, and online social networks. In particular, these technologies provide cost-effective scalable solutions, thanks to the flexibility and elasticity in resource provisioning and the use of advanced virtualization and scheduling mechanisms [5, 13].

Cloud workloads consist of a collection of many diverse applications and services, each characterized by its own performance and resource requirements and by constraints specified in the form of service-level agreements (SLAs). A large number of factors affect cloud performance, including, among others, the variability in the resource and network conditions and the highly dynamic nature of the workloads, whose intensity can suddenly grow or shrink as a consequence of the user interactions. More specifically, the use of virtualized time-shared resources could lead to performance degradation. This degradation is mainly due to the interference and resource contention arising from the colocation of heterogeneous workloads on the same physical infrastructure and to the overheads caused by the resource management policies being adopted. Similarly, the mix of workloads concurrently executed on a given virtual machine (VM) can be responsible for some unpredictable effects on the performance because of incompatible temporal patterns of the resource usage [74]. These performance issues could become even more critical in multicloud environments where the workload is distributed across different cloud infrastructures.

In these complex scenarios, mapping cloud resources to workload characteristics is very challenging [43]. Nevertheless, it is of primary importance for an effective deployment of cloud technologies and to achieve the desired service levels. Hence, to address resource management, provisioning and online capacity planning, and, more generally, to manage and predict performance and Quality of Service (QoS), it is essential to gain a deep understanding of the properties and the evolution of cloud workloads. Therefore, systematic and structured approaches toward workload characterization have to be considered as an integral component of all these strategies.

Despite their importance, the characterization and forecasting of cloud workloads have been addressed in the literature to a rather limited extent and mostly at the level of the VMs without taking into consideration the features of the individual workload components running on the VMs themselves. The aim of this chapter is to provide an overview of the main issues related to the entire life cycle of workload deployment in cloud environments. More specifically, starting from the identification of the most relevant behavioral characteristics of cloud workloads, we define some broad workload categories described in terms of qualitative and quantitative attributes. The chapter then focuses on the various workload categories and discusses the challenges related to their monitoring, profiling, and characterization. This thorough investigation of the state of the art is complemented by a literature review of the exploitation of scheduling strategies and failure analysis and prediction mechanisms of the framework of cloud workloads.

The chapter is organized as follows. Section 2 presents the categories identified for cloud workloads, while Sect. 3 discusses the main issues related to their monitoring and profiling. The workload structures and resource requirements are addressed in Sects. 4 and 5, whereas the challenges related to workload scheduling and failure analysis and prediction are briefly illustrated in Sects. 6 and 7, respectively. Finally, Sect. 8 presents some concluding remarks.

2 Workload Categories

The term workload refers to all inputs (e.g., applications, services, transactions, data transfers) submitted to and processed by an e-infrastructure. In the framework of cloud computing, these inputs usually correspond to online interactions of the users with web-based services hosted in the cloud or to jobs processed in batch mode. On the contrary, cloud workloads almost never refer to hard real-time applications.

In this section, we analyze the behavioral characteristics of cloud workloads (i.e., their qualitative and quantitative attributes) to identify some broad categories specified in terms of various dimensions, namely:

- Processing model.
- Architectural structure.
- Resource requirements.
- Nonfunctional requirements.

The choice of these dimensions is mainly driven by their role in the formulation of the cloud management strategies and in the assessment of the service levels foreseen by the workloads.

The *processing model* adopted by the workload, that is, online (i.e., interactive) and offline (i.e., batch or background), is an important high-level dimension that identifies two workload categories. These categories are characterized by very diverse behaviors and performance requirements as well as by a different impact on management policies (e.g., resource scheduling, VM placement, VM migration). An *interactive workload* typically consists of short-lived processing tasks submitted by a variable number of concurrent users, whereas a *batch workload* consists of resource intensive long-lived tasks. Hence, as we will discuss later on, these workload categories exercise cloud resources to a rather different extent.

Another dimension chosen to classify cloud workloads focuses on their *architectural structure* expressed in the form of processing and data flows characterizing each individual application. More precisely, these flows are described by the number and types of services or tasks being instantiated by a cloud application and their mutual dependencies, and, as such, have a strong impact on the scheduling policies. In particular, multiple task applications can be organized according to different models, namely:

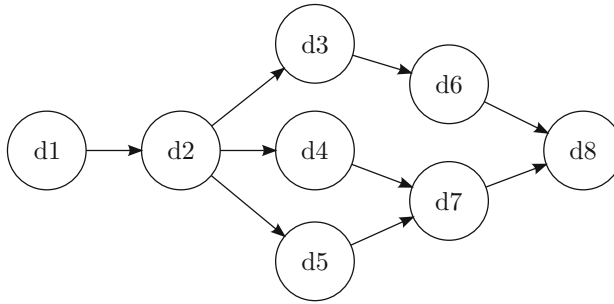


Fig. 1 Directed acyclic graph representing the data flow of a cloud application organized according to a hybrid model

- Pipeline model.
- Parallel model.
- Hybrid model.

In the *pipeline model*, tasks need to be processed sequentially one after the other with tight precedence constraints. On the contrary, in the *parallel model*, the tasks are characterized by precedence constraints that allow for concurrent execution of multiple tasks. In addition, these models are often combined in a sort of *hybrid architectural model* where the relationships among tasks are usually more complex. Figure 1 shows an example of a directed acyclic graph that represents the data flow of a simple cloud application organized according to a hybrid model. The nodes and edges denote the datasets and relationships between them, respectively.

In the framework of scientific workloads, their description often relies on the *many-task computing* (MTC) paradigm [67], an architectural structure consisting of loosely coupled tasks and involving large volumes of data. Conversely, interactive cloud applications are typically organized according to *multitier* architectures. As we will discuss in Sect. 4, the interdependency among tiers and the patterns followed by the applications strongly affect the deployment of scaling strategies in cloud environments. Moreover, it is not always possible to derive a detailed view of the workload structure because of the lack of specific design information.

The definitions of workload architectural structures do not include any details about the behavioral characteristics of the workload at runtime (e.g., resource requirements, scheduling events). Nevertheless, *qualitative attributes* (e.g., priority, termination status) and *quantitative attributes* (e.g., workload intensity, demands and usage patterns of cloud resources) are very relevant to devise accurate resource allocation strategies. In particular, quantitative attributes provide a detailed characterization of the *computing, communication, and storage requirements* of the workload and have to be assessed very carefully to avoid overprovisioning or underprovisioning of the resources (e.g., CPU, memory, I/O, network).

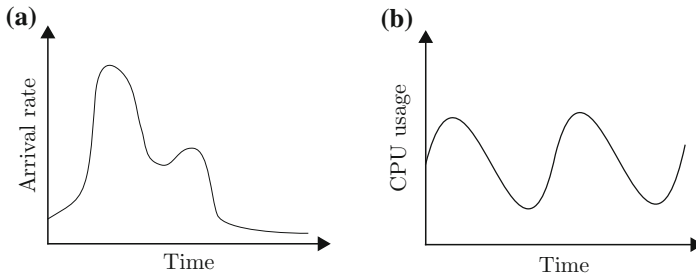


Fig. 2 Examples of a diurnal pattern characterizing the workload arrivals **(a)** and a periodic pattern describing the CPU usage **(b)**

Depending on the amount of resources used, workloads are classified as:

- Compute intensive or I/O intensive.
- Elastic or bandwidth sensitive.

Generally speaking, we can say that network bandwidth is more critical for online interactive workloads, whereas storage and computing resources often characterize batch workloads. Moreover, the *resource requirements* of some workloads are *stable* (i.e., uniformly distributed over their execution), whereas other workloads (e.g., workloads associated with online services) exhibit some specific *temporal patterns*, such as *periodic*, *bursting*, *growing*, and *on/off*. These patterns typically depend on the intrinsic characteristics of the applications, as well as on the workload intensity. In detail, patterns can refer to a single resource or multiple resources. A communication intensive phase can be followed by a compute intensive phase. Similarly, during the execution of an application, the bandwidth usage can change and follow some specific patterns.

As already pointed out, cloud workloads consist of streams of jobs and requests submitted at unpredictable times. Hence, their *arrival process* is seldom deterministic. It is often characterized by various effects (e.g., *diurnal patterns*, *seasonal effects*, *flash crowd* phenomena). In general, the *burstiness* in the workload intensity and heavy load conditions cause sudden and unexpected peaks in the resource demands that have a critical impact on resource provisioning strategies. Figure 2 shows two examples of qualitative patterns, namely, a diurnal pattern typically associated with the intensity of interactive workloads and a periodic pattern corresponding to CPU usage.

An additional dimension describing the workload refers to *nonfunctional requirements* related to SLA constraints (e.g., performance, dependability, security). Among these attributes *reliability* is particularly important in cloud environments especially when deploying business-critical or safety-critical applications. Reliability denotes the probability that workloads can successfully complete in a given time frame. The presence of *failures* decreases the reliability. Failures are due to various types of events (e.g., software bugs, exceptions, overflows, timeouts). For example, for data intensive workloads, a sudden increase in the rate at which data are submitted for

processing can lead to failures, thus making the service unavailable. Moreover, failures are often correlated, that is, they often occur between dependent or colocated services or applications.

The remainder of the chapter focuses on the approaches typically adopted for monitoring and characterizing the workload categories presented in this section. The issues related to workload scheduling and failure analysis will also be discussed.

3 Workload Monitoring and Profiling

Monitoring and profiling are the basis for measuring the qualitative and quantitative attributes of the workloads. Generally speaking, monitoring keeps track of the activities performed by the workloads being processed and of the status of the allocated and the available resources. Profiling focuses on describing how workload exploits the cloud resources. Monitoring and profiling in the clouds are particularly difficult because of the heterogeneity and dynamicity of these environments [82]. Nevertheless, these activities play a critical role when addressing scenarios, such as

- Capacity planning and resource management.
- Performance tuning.
- Billing.
- Security and troubleshooting.
- SLA verification.

Various approaches have been devised to tackle specific monitoring issues (e.g., measurement sources and accuracy, sampling granularity, intrusiveness, and scalability). In what follows, we focus on the workload attributes that can be monitored at runtime to describe the resource usage. The level of details of the measurements collected in the clouds depends on the monitoring perspective adopted, namely, *cloud providers* and *cloud users*. Three basic types of cloud monitoring targets can be considered:

- Client.
- Virtual machine.
- Physical machine.

More specifically, cloud providers can measure resource usage of physical machines and of individual VMs from the vantage point of the hypervisor. On the other hand, cloud users are restricted to measure their own workloads using client logging and profiling facilities. Indeed, the VM isolation typical of virtualization technologies hides the characteristics and performance of the underlying physical machines and the VM management policies. In detail, to collect measurements on resource usage and cross-correlate them with application-specific data and scheduling details, cloud users have often to resort to profiling facilities made available by providers (see, e.g., [29, 70]). To derive a more detailed description of the workloads being processed, VM measurements can be complemented with additional information

about the workload structure, as well as with guest operating system statistics. Moreover, application logs are exploited to correlate the resource usage with workload intensity and characteristics.

Monitoring tools usually collect measurements by deploying distributed software agents that periodically gather information about the usage of resources, such as CPU, memory, and I/O devices. In general, monitoring approaches rely on system tools and interfaces (e.g., `vmstat`, `iostat`, `netstat`) or on proprietary solutions [1, 46]. Moreover, depending on the monitoring capabilities of the virtualization technologies, ad hoc scripts can be used for sampling low-level quantitative attributes, such as CPU waiting times, number of virtual memory swaps, TLB flushes, and interrupts [7]. The monitoring agents can also collect VM scheduling and provisioning events, (e.g., number and types of allocated VMs) [11]. The granularity and level of details of the measurements have to be chosen with the aim of limiting the monitoring intrusiveness. Measurements are usually stored into *tracelogs*, that is, collections of time-stamped recordings of various types of information (e.g., resource demands, scheduling events, application specific data). Note that, despite the importance of workload measurements for both researchers and practitioners, cloud providers are seldom willing to publish detailed measurements about their own workloads often to prevent leakage of confidential competitive information.

Profiling is another approach applied to measure the resource usage of individual workload activities for driving performance tuning actions. In particular, profiling can be exploited by cloud users for optimal dynamic resource provisioning and by cloud providers for tuning VMs placement and scheduling policies [28]. In cloud environments, profiling has to cope with new challenges due to interference among colocated VMs. Indeed, the sharing of hardware resources could result in unpredictable behaviors of hardware components, such as cache memories, CPU pipelines, and physical I/O devices [85]. Typical solutions for collecting profiling measurements are based on *dynamic instrumentation* and *sampling hardware performance counters*. An alternative approach is based on measuring at the *hypervisor level* the overall behavior of the VMs hosting the target applications. In detail, the dynamic instrumentation takes advantage of software probes that selectively record runtime events about the application behavior (e.g., time stamps related to the execution of a given portion of an application). On the other hand, hardware-based profiling exploits CPU performance monitoring unit for sampling counters related to low-level events, such as cache misses, clock cycles per instruction, pipeline stalls, and branch mispredictions.

In general, profiling can cause significant intrusiveness. Indeed, fine-grained instrumentation and high sampling frequency result in large volume of measurements and perturbations of the workload behavior. On the contrary, coarse grain sampling and instrumentation could lead to ignore some rare though important events that might have a significant impact on the overall resource usage. To reduce the intrusiveness and the resource requirements of profiling activities various solutions, such as adaptive bursty tracing technique, based on a sampling rate inversely proportional to code execution frequency, have been devised [49].

Although monitoring and profiling are essential aspects of cloud computing, up to now no portable, general purpose, and interoperable monitoring and profiling tools exist. This lack results in a plethora of open source and commercial tools addressing specific targets and platforms [15, 32]. Examples of *open source monitoring tools* are: Nagios,¹ that is part of the OpenStack suite, Ganglia, Collectl,² and MonALISA.³ Cloud providers offer several *commercial tools* (e.g., Amazon Cloudwatch, Microsoft Azure Watch, IBM Tivoli Monitoring, Rackspace, Rightscale, Cloudify, Aneka). While these monitoring facilities are designed to be deployed to cloud environments, external monitoring services like CloudHarmony,⁴ CloudSleuth,⁵ CloudClimate,⁶ and Up.time,⁷ focus on monitoring applications and infrastructures from multiple locations on the Internet.

The development of a common framework for workload monitoring and profiling in the clouds is an open issue, that might also prevent users to deploy their businesses in these environments [39]. To improve the scalability and effectiveness of monitoring service consolidation and isolation, recent studies introduced the concept of *monitoring-as-a-service* (MaaS) [60, 63]. The possibility for cloud users to monitor the global state of their applications is a challenging research question that deserves some further explorations.

4 Workload Structures

In this section, we present a literature review of the most common structures of cloud workload introduced in Sect. 2, and the models used for their representation. Workload architectural structure is the description of the tasks an application consists of and of their relationships. This structure is usually known at design time, whereas it can be difficult to derive it at runtime. Nevertheless, it is an important characteristic to be taken into account for the dynamic provisioning and optimal allocation of cloud resources and for identifying cost-effective solutions able to exploit the available parallelism (see, e.g., [14, 58, 84]). From the cloud provider perspective the aim is to maximize both resource utilization and energy savings, whereas from the cloud user perspective the aim is to minimize the operational costs while achieving optimal performance. The workload structures covered in this section refer to the following frameworks:

¹<http://nagios.sourceforge.net>.

²<http://collectl.sourceforge.net>.

³<http://monalisa.caltech.edu>.

⁴<http://cloudharmony.com>.

⁵<http://cloudsleuth.net>.

⁶<http://www.cloudclimate.com>.

⁷<http://www.suptimesoftware.com>.

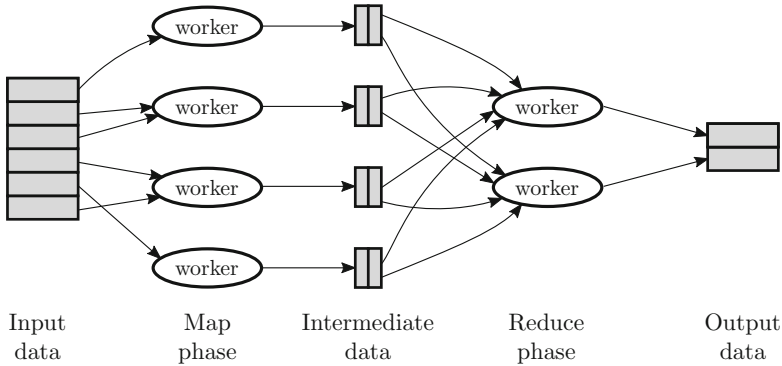


Fig. 3 Overview of MapReduce programming model

- MapReduce programming model.
- Workflow technologies.
- Many-task computing paradigm.
- Multitier architecture.

These structures are not restricted to a single processing model. For example, a multitier architecture can be exploited for both batch and interactive applications. In detail, *batch applications* often consist of tasks with parent–child relationships. These applications are modeled as workflows describing the tasks in terms of data dependencies and data and control flows. As stated in Sect. 2, typical workflow schemes are pipeline, parallel, and hybrid, that is, sequential, concurrent, and combinations of sequential and concurrent tasks, respectively.

Several approaches have been proposed to take advantage of the concurrency of application workflows. For example, *MapReduce* [25] is a programming model introduced to ease the exploitation of the parallelism in big data analytic workflows. Applications based on this paradigm are executed according to a hybrid structure consisting of multiple concurrent tasks (i.e., map and reduce workers), as illustrated in Fig. 3. The intermediate data shuffle addresses the data dependencies of the workflow. To describe and predict interarrival times and resource demands of MapReduce workloads, statistical techniques, such as kernel canonical correlation analysis and fitting, have been proposed [6, 34]. Due to the heterogeneity of the application domains in which MapReduce is exploited, the workloads are often characterized in terms of different attributes (e.g., workload intensity, task durations, and constraints) [21, 71]. Very popular cloud technologies based on MapReduce (i.e., Apache Hadoop,⁸ Spark⁹) perform automatic optimizations and data distributions. However, the deployment of Hadoop applications requires the tuning of many configuration parameters that might heavily affect the overall performance [87].

⁸<http://hadoop.apache.org>.

⁹<http://spark.apache.org>.

On the other hand, Spark applications take advantage of in-memory computations to reduce the overhead of Hadoop distributed file system [45].

In the framework of *scientific computing*, workflow technologies are an approach for easy and efficient development of applications with hybrid structures. In the literature, the workflow of these applications has been analyzed in terms of resource demands (e.g., number of tasks and their CPU, memory and I/O demands) [48, 92]. Similarly to workflow technologies, the *many-task computing* paradigm is widely used to develop distributed loosely coupled scientific applications. MTC applications typically require over short-time periods a large amount of computational resources to process the so-called bag-of-tasks. Hence, MTC is well suited to take advantage of dynamic provisioning of cloud resources. The studies related to the deployment of these applications on the clouds mainly focus on performance analysis of various types of infrastructures, such as commercial cloud computing services and federated clouds [62, 72]. In particular, performance and resource demands of scientific MTC applications have been investigated by analyzing workload tracelogs collected in environments other than clouds (e.g., parallel production infrastructures, grids). The behavior of scientific workflows is characterized in [41] in terms of number of jobs and of bag-of-tasks to identify the bottleneck in the resources. Workload tracelogs have also been analyzed for developing strategies aimed at reducing the impact of transient failures on the overall behavior of MTC applications [17]. These strategies, based on checkpoint and speculative execution policies, reduce the large overheads due to the entire bag-of-tasks resubmission, although they might affect resource usage with unnecessary duplicated task executions. It is worth noting that clouds can be a cost-effective and scalable alternative to the traditional high performance computing environments for a large variety of scientific applications. However, performance can be an issue. Indeed, bandwidth and jitters on network delays are among the most critical factors that limit the performance of scientific applications [59].

Regarding *interactive workloads*, that usually need to cope with the dynamic behavior of users, cloud computing is mainly adopted for deploying large-scale applications in domains, such as e-commerce, financial services, healthcare, gaming, and media servers. A common solution to address these highly variable load conditions is based on *multitier architectures*, where each tier, deployed on one or multiple VMs, addresses a specific functionality (e.g., web, database, application logic, load balancing). As an example, Fig. 4 depicts an architecture of a five-tier web application. The advantage of this solution is the possibility of dynamically scaling each tier independently, both horizontally and vertically. Horizontal scaling deals with varying the number of VM instances (see Fig. 4b). On the contrary, vertical scaling is about varying the amount of resources allocated to individual VMs. Figure 4c shows that the VM deploying the web server scales up and doubles its number of cores.

Resource provisioning for multitier architectures is challenging because of the functional interdependence among the tiers and the network overhead. Therefore, the sizing of each tier plays a critical role for this kind of applications [42]. Moreover, it is difficult to model multitier applications due to the dynamic and unpredictable behavior of their users. In this framework, resource provisioning and

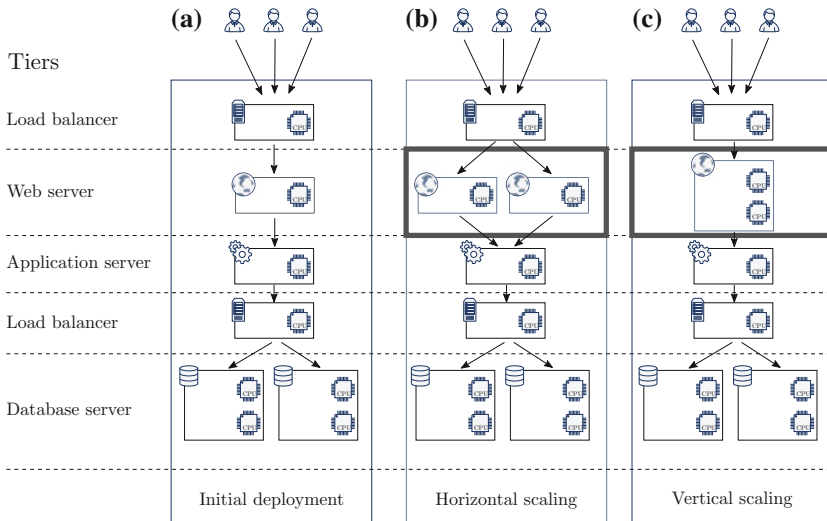


Fig. 4 Example of a five-tier architecture typical of large-scale web applications with its initial deployment (a), and the *horizontal* (b) and *vertical* (c) scaling of the web server tier, respectively

scaling have been investigated using stochastic models based on queuing networks and control theory [40]. For vertical scaling, linear regression methods, Markov chains and queuing network models are often used to represent the relationships between workload being processed and resource demands. Many studies focus on probability distributions and statistical bounds to derive performance metrics of cloud environments, such as response time, throughput, and resource utilizations (see, e.g., [10, 75]). These metrics are used to characterize the workload, predict its behavior, and scale resources accordingly. An alternative approach is based on multi-input multi-output control systems, where inputs are the resources allocated to each tier and outputs are the measured performance metrics [93]. For horizontal scaling, reactive heuristics leveraging a threshold-based set of rules are commonly used. Thresholds on resource utilization trigger the start or the shutdown of VMs in order to ensure given QoS levels [54]. Moreover, proactive approaches for resource provisioning take into account the resource demands as a function of the workload intensity. For example, queuing networks can be used for modeling the relationships between number and characteristics of allocated VMs and metrics, such as blocking and immediate service probabilities [52]. Additionally, optimal resource provisioning has been addressed by means of queuing network and simulation models [37, 38].

5 Workload Attributes

In this section, we present a literature review of the approaches typically applied to characterize cloud workloads in terms of both *qualitative attributes* related to jobs and tasks events and *quantitative attributes* describing workload intensity and the demands of cloud resources (i.e., computing, communication, and storage). These attributes are usually obtained from historical data (e.g., tracelogs) and runtime measurements. It is important to point out that tracelogs published by Google [68] are among the few publicly available cloud measurements. These logs store both qualitative and quantitative anonymized attributes about the jobs executed on a large cluster (i.e., demands and actual usage of CPU, memory and disk space, scheduling events of jobs and tasks).

The workload models obtained as a result of characterization studies are very useful when addressing the optimization of resource usage, the definition of scheduling policies and energy aware solutions, the prediction of failures and many other cloud management issues. In the literature, cloud workloads are characterized by focusing on jobs and tasks and analyzing their attributes, referring to

- Resource usage.
- Workload intensity.

Commonly adopted approaches are based on various types of techniques, often used in combination, such as

- Statistical and numerical techniques.
- Stochastic processes.

Some papers [55, 69] analyze the *resource usage* and its dynamics at *job* and *task* levels, by applying a statistical approach based on a high-level *exploratory analysis* (i.e., descriptive statistics, empirical distributions of resource usage, visual inspection of their temporal behavior). These studies rely on the Google tracelogs. In particular, patterns of task submissions, interarrival times, relationships between resource usage and task status (i.e., killed, normally terminated, failed) are considered. For example, the identification of jobs resubmitted because of failures or evictions provides some interesting insights for predicting the resources actually required by the workload.

In order to derive realistic models that capture the heterogeneity of jobs and tasks, more advanced *statistical* and *numerical techniques* (e.g., clustering, fitting) are adopted. *Clustering* techniques are usually applied to identify groups of workload components characterized by similar behaviors. Early papers [20, 61] classify jobs and tasks based on their CPU and memory usage. In particular, a medium grain classification of tasks highlights the presence of few tasks that consume a large amount of resources. More recently, the statistical properties of the workload are analyzed to classify cloud applications in terms of both quantitative (i.e., resource requirements) and qualitative (i.e., task events) attributes [26]. In general, job and task classification has been applied for devising scheduling and allocation policies. For example, the approach proposed in [66] estimates the resource demands of tasks

and predicts the cluster to which a new arriving job belongs to according to its initial resource demands. As a consequence, resource utilization and energy saving can be improved. In [9] clustering is applied to identify tasks characterized by similar memory and CPU usage, as well as tasks whose memory usage is independent from their CPU usage. Moreover, this study analyzes the dynamics of the CPU usage to discover weekly and daily patterns and in particular synchronized peaks whose presence is important for devising more efficient allocation strategies.

As pointed out in Sect. 3, it is difficult to obtain detailed measures on resource usage of the *specific workload components*. Most studies rely on measurements collected at the *VM level*. Although these measurements refer to the overall resource usage of individual VMs, they provide an accurate description of the application behavior in virtualized environments. Understanding and modeling this behavior are important in many domains, such as workload scheduling, VM failure monitoring, and intrusion detection. To highlight the variability in resource usage and the presence of temporal patterns, some studies combine statistical metrics (e.g., correlation between attributes) with autocorrelation functions and time series analysis [7, 24]. The evolution of CPU, memory, and disk utilizations is analyzed in [11] by representing their dynamics and fluctuations as a time series at different time scales. *Numerical fitting* techniques are applied to build models that capture the temporal variability in resource usage. Moreover, by looking at the correlations among resource usage, dependencies to be exploited in the design of effective consolidation strategies are identified. A time series approach is also adopted in [51] to represent CPU usage patterns. Additionally, a co-clustering technique identifies groups of VMs with correlated workload patterns, whereas a Hidden Markov Model predicts the changes of these patterns.

Workload intensity is another important aspect extensively analyzed in the literature because of its strong impact on cloud performance. In [78] workload intensity is quantified in terms of task submission rate and clustering is applied to highlight variability in the submission rate across groups of tasks. Other papers model the workload intensity by means of *stochastic processes*. It has been shown that simple Poisson processes generating independent identically distributed interarrival times are not suited to represent real cloud workloads [47]. *Burstiness*, a well-known characteristic of network traffic, has also been observed in cloud environments. Bursty and fractal behaviors of the arrival processes affect in particular load balancing strategies [81]. In addition, detecting, measuring, and predicting these phenomena are important for devising efficient resource-provisioning and energy-saving strategies. To describe the time-varying behavior and self-similar effects, metrics, such as index of dispersion and coefficient of variation, are complemented with models based on 2-state Markovian arrival processes, parameterized with different levels of burstiness [88]. The two states represent the bursty and nonbursty request arrival processes, respectively (see Fig. 5). Markovian arrival processes are integrated in [64] with analytical queueing models to predict system performance. A different approach based on fractal techniques is proposed in [16, 36] for representing workload dynamics in terms of job arrivals. The arrival process is modeled using fractional-order differen-

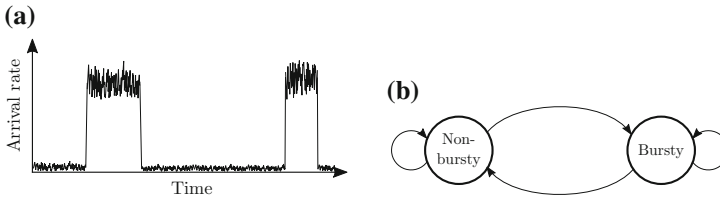


Fig. 5 Example of nonbursty and bursty arrivals **(a)** and the corresponding 2-state Markovian arrival process **(b)**

tial equations with time-dependent parameters, whereas fitting is applied to identify statistical distributions for CPU and memory usage.

The literature review presented in this section highlights the importance of taking into account workload characteristics to effectively deploy cloud technologies. Even though different approaches to workload characterization in cloud environments have been proposed, few studies focus on the attributes of the individual workload components. In addition, there is the need to devise more systematic approaches toward workload characterization. The lack of publicly available workload measurements makes quite difficult to investigate real-life cloud computing scenarios.

6 Workload Scheduling

Workload scheduling, i.e., the mapping between jobs/tasks and VMs, is a challenging issue in cloud environments because of the heterogeneity of workload characteristics (e.g., intensity and resource demands). The problem of finding an *optimal mapping* is NP-complete and therefore intractable with exact methods when the number of VMs and tasks is large, as it is typically the case of cloud environments. For this reason, (meta)heuristics are currently used to find suboptimal solutions. Metaheuristics based on methods, such as neural networks, evolutionary algorithms, or set of rules, are proved to be efficient in solving optimization problems related to scheduling. In the remainder of this section, we review the literature (see Table 1 for an overview) by briefly discussing the following aspects of workload scheduling:

- Scheduling objectives.
- Optimization approaches.
- Resource scaling.
- Load balancing.
- Scheduling of real-time applications.

Objectives of the scheduling problem are multiple (e.g., to minimize makespan, data transfer, energy consumption, and economic cost, to satisfy SLAs). A simple approach takes into account one objective at a time. Alternative approaches are aimed at combining multiple objectives into a single aggregate objective

Table 1 Summary of the state of the art on workload scheduling in cloud environments

Reference	Workload structure	Method	Optimization objectives	Deployment model
Somasundaram and Govindarajan [79]	Tasks with deadlines	Particle swarm	Execution time and economic cost	Private
Duan et al. [30]	MTC	Game theory (multi-objective)	Makespan and economic cost	Hybrid
Zhang et al. [90]	MTC	Vectorized ordinal optimization (multi-objective)	Makespan and economic cost	Private
Zhan et al. [89]	Survey on evolutionary approaches to scheduling			
Van den Bossche et al. [83]	Tasks with deadlines	Binary integer programming	Resource utilization	Hybrid
Pandey et al. [65]	Workflow	Particle swarm	Makespan	Public
Kessaci et al. [50]	Tasks with deadlines	Pareto-based genetic algorithm (multi-objective)	Energy consumption, carbon emission and profit	Federated
de Oliveira et al. [24]	Independent tasks	Ant colony	Makespan and load balancing	Federated
Wu et al. [86]	Survey on workflow scheduling			
Jiang et al. [44]	Workflow	Path clustering heuristics and list based scheduling	Makespan and resource utilization	Private (High Performance Computing)
Zhang et al. [91]	Independent tasks with priorities	Model predictive control with heuristics	Energy consumption, scheduling delay and economic cost	Public and Private
Mao and Humphrey [57]	Workflow	Heuristics	Economic cost	Public and private
Dutta et al. [31]	Interactive	Decision tree	Resource usage	Public and private
Ardagna et al. [4]	Interactive	Reactive set of rules	Resource usage	Public and private
Cheng et al. [22]	Interactive and batch	Nonlinear optimization	QoS and load balancing	Hybrid

(continued)

Table 1 (continued)

Reference	Workload structure	Method	Optimization objectives	Deployment model
Singh et al. [77]	Interactive multitier	Clustering and queuing	QoS	Public
Spicuglia et al. [80]	Interactive	Reactive heuristic with thresholds	Response time, resource utilization and load balancing	Public and private
Li et al. [53]	Real-time tasks	Heuristic with penalties on deadline	Economic cost	Public and private
Liu et al. [56]	Real-time tasks	Heuristic with eviction	Economic cost	Public and private

References are ordered as they appear in the text

function (see, e.g., [79]) or considering multi-objective algorithms (see, e.g., [30, 90]). A recent survey summarizes the evolutionary approaches for scheduling in cloud environments [89]. The different viewpoints for scheduling and the corresponding objectives are identified as follows:

- *Scheduling for user QoS*, where objectives include the makespan and user costs minimization, application performance, and reliability.
- *Scheduling for provider efficiency*, where objectives are load balancing, utilization maximization, and energy savings.
- *Scheduling for negotiation*, where the goal is to satisfy both user and provider objectives at the same time.

Exact methods for solving the optimization problem (e.g., constrained binary integer programming) can be used in simple scenarios only, such as trivial parallel workloads where tasks are fully decoupled without any precedence constraint [83]. For more general workload structures, the problem complexity increases and it is necessary to devise *heuristic* optimization methods, such as particle swarm optimization [65], genetic algorithms [50], ant colony optimization [24], and game theoretic algorithms [30]. Another recent survey [86], focusing on the main issues related to *workflow scheduling*, subdivides the scheduling methods into three main categories, namely

- *Static scheduling*, where workload structure is known a priori and resources have instantaneous availability.
- *Dynamic scheduling*, where workload structure can be obtained at runtime.
- *Static planning with dynamic scheduling*, where the structure and communication time can be estimated. Tasks are statically planned, although dynamically scheduled to resources at runtime.

Examples of offline methods for static scheduling multitenant workflows in cloud environments are presented in [44]. These methods take advantage of gaps in the schedule due to communication overheads and task dependencies. In particular, the gap search is performed on the entire task group or in a distributed fashion by working on its partitions. The scheduling problem can be even more complex when *task priorities* are considered [91]. Low-priority tasks are often evicted because of the overcommitment of physical resources. Moreover, changes in the cloud environment properties can affect the priorities of jobs and tasks.

Job scheduling and *resource scaling* are often considered in conjunction [57]. Several frameworks have been recently introduced to address resource scalability. For example, SmartScale [31] is an automated scaling framework that uses a combination of vertical and horizontal approaches to optimize both resource usage and reconfiguration overheads. Scaling mechanisms are also encountered in [4] where different scalability patterns are considered and an approach to performance monitoring that allows automatic scalability management is proposed. Autoscaling is often used in conjunction with *load balancing* strategies. Even though physical machines are often the main target of these strategies, effective load balancing and resource

allocation policies take into account the concurrent execution of different application types, i.e., interactive, batch, and the mix of applications with different resource requirements and workload structures (see Sect. 4) [22, 77, 80].

Hard real-time applications (i.e., applications characterized by hard deadlines that are a priori guaranteed to be met) are not well suited to the current cloud infrastructures. In fact, the virtualization technologies and network protocols used in the clouds are not designed to provide the timing guarantees required by these applications. However, the so-called *soft deadlines* are often taken into account by the schedulers because of the penalties associated with the negotiated SLAs [53, 56]. Despite hard real-time applications, for online services hosted in cloud environments the main goal of the scheduling is to maximize the profit by providing timely services.

The analysis of the state of the art presented in this section has shown that workload scheduling in the clouds is a very important research field. Although there are numerous studies on workload scheduling on parallel and distributed systems, few papers address real cloud environments, and even fewer cloud workload management systems. Nevertheless, all these topics need further investigation.

7 Workload Failures

As described in the previous sections, workloads typically consist of diverse applications with different priorities and deadlines that reflect the user requirements. Unforeseen workload behaviors or incompatibility between workload requirements and the resources offered in the clouds result in *failures*. Increasing functionality and complexity of cloud environments are leading to inevitable failures that can be caused by different types of events, such as outage, vulnerability, and automatic updates [33]. Other examples of failures are software crashes due to hidden bugs, out of memory exceptions due to the lack of resources, denial of service due to malicious activities, deadline violations due to unexpected processing delays. There are also failures caused by unknown events.

A decrease in the reliability associated with the workload does not necessarily mean that the applications are not successfully completed because of bugs. The failure rate often depends on the workload intensity and mixes. In particular, heavy load conditions are often responsible of the increase of the overall failure rate. All failures and in particular deadline violations are crucial in cloud environments because of their negative impact on QoS and SLA. Hence, whenever an SLA has been established between a cloud provider and a cloud user, various strategies, such as replication and checkpointing, have to be deployed in order to cope with failures.

In the literature (see Table 2 for an overview) cloud failures have been addressed under two different perspectives, namely

- Failure analysis.
- Failure prediction.

Table 2 Summary of the state of the art in the field of cloud failure analysis and prediction

Reference	Target	Failure type	Parameters	Modeling Approach
Garraghan et al. [35]	Google tracelog	Task and server	Failure and repair times and task termination status	Probabilistic
Chen et al. [18]	Google tracelog	Job and task	Job and task attributes	Statistical and probabilistic
Chen et al. [17]	Scientific workflows	Transient	Task runtime and failure interarrival time	Probabilistic
Di Martino et al. [27]	Cloud data	Operational	Failure rate and MTBF	Probabilistic
Chen et al. [19]	Google tracelog	Job and task	Resource usage, task priority and resubmission	Machine learning
Samak et al. [73]	Scientific workflows	Job	VM attributes	Machine learning
Bala and Chana [8]	Scientific workflows	Task	Resource utilizations	Machine learning

References are ordered as they appear in the text

In particular, to prevent wasting resources, avoid performance degradation, and reduce costs and energy consumption, in the last years, extensive research has focused on *failure analysis*. Failures are characterized using different statistical and analytical techniques focused on resource usage (e.g., CPU, memory, disk I/O, bandwidth) and on other workload qualitative attributes (e.g., priority, termination status). The basis of these analyses is often represented by the large variety of workload information collected in cloud production environments (see Sect. 3). For example, the analysis of the Google tracelog presented in [35] focuses on the characteristics of failures of cloud workloads. This empirical study considers the failure and repair times, and, in particular, two important metrics, namely

- Mean Time Between Failure (MTBF).
- Mean Time To Repair (MTTR).

More specifically, the statistical properties of these metrics together with the theoretical distributions that best fit the empirical data (e.g., Weibull, lognormal) are the basis for characterizing the behavior of the failures. The study shows that, in general, the workload failure rates vary significantly and depend on the priority associated with the individual tasks, thus reflecting the diversity in the workload characteristics. The Google tracelog is also analyzed in [18] to evaluate the effects exercised on failures by workload attributes, such as job and task resource usage, task resubmission for single and multiple task jobs and termination statuses. In addition, the study investigates the relationships between user behavior and failures. Clustering

techniques have been applied to identify groups of users submitting jobs with similar characteristics and termination status, thus exhibiting similar reliability properties. Transient failures associated with scientific workflows are investigated in [17] by modeling failure interarrival times and system overheads.

Failed jobs typically consume a significant amount of resources. Hence, it is crucial to mitigate their negative impact by predicting failures in a timely manner. In [27] the operational failures of a business data processing platform are characterized to estimate common failure types, their rates and relationships with the workload intensity and data volume. In addition, a trend analysis is performed to assess whether failure arrivals significantly change over time.

Failure prediction usually relies on machine learning techniques, such as naive Bayes, random forest, and artificial neural networks. In particular, recurrent neural networks are applied in [19] to predict the failures of jobs and tasks by analyzing their resource usage. In the framework of large-scale scientific applications represented as *workflows*, naive Bayes classifiers are used to study the behavior of jobs and predict their failure probability [73]. Similarly, failure prediction models for tasks in workflow applications are proposed in [8]. These models rely on various machine learning approaches and are the basis of proactive fault tolerant strategies for failure prediction to be used for the identification of tasks that could fail due to the overutilization of resources (e.g., CPU, storage).

A special category of failures is related to *software aging*. The presence of these failures is manifested as either an increase in their rate or in performance and QoS degradations. Typical causes of software-aging failures are elusive bugs, such as memory leaks, unterminated threads, and unreleased locks. The effects of these bugs usually become evident whenever peaks and bursts appear in the workload. A common solution to cope with these problems is represented by *software rejuvenation*, that is, a cost-effective software maintenance technique based on preventive rollbacks of continuously running applications. A recent survey [2] presents an interesting classification of the most common approaches used in this framework (see Fig. 6).

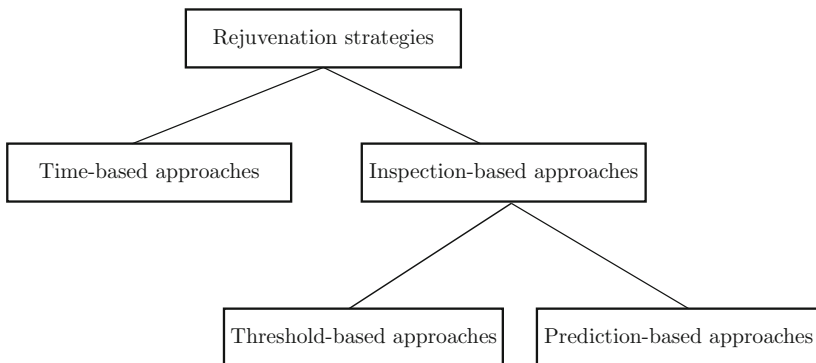


Fig. 6 Classification of the main rejuvenation strategies

A detailed overview of the analysis techniques proposed in the literature for software aging and rejuvenation (e.g., stochastic processes, time series analysis, machine learning) is offered in [23]. In particular, in cloud environments, software rejuvenation can be applied to either individual VMs and to the hypervisor. Techniques based on live VM migrations and checkpointing are often exploited to reduce downtimes due to failures [12]. Similarly, to reduce the downtime during the rejuvenation, time series approaches are used to predict the proper time to trigger the process [3]. In detail, to guarantee a safe scheduling of rejuvenation actions, the resource-aware rejuvenation policy introduced in the paper considers multiple thresholds referring to the resource usage (e.g., virtual memory).

Despite the effort already put in the domain of cloud failure analysis and prediction, some open challenges remain to be investigated. In particular, to improve workload reliability in the clouds, failure awareness resource provisioning and integration of failure prediction mechanisms in the schedulers should be devised.

8 Conclusions

A deep understanding of workload properties and behavior is essential for an effective deployment of cloud technologies and for achieving the desired service levels. In this chapter, we discussed the main issues related to the entire life cycle of the workloads in the clouds, starting with their characterization at design time (i.e., workload categories, structures, and patterns), their matching at the deployment phase (i.e., resource requirements and scheduling), and the issues in the execution phase (i.e., failure analysis and prediction).

The list of topics and issues related to cloud workloads presented in this chapter does not pretend to be exhaustive. However, the snapshot of the state of the art gathers in one place the pointers to many different approaches and can be therefore seen as a starting point in the design of a comprehensive framework dealing with all stages of the workload life cycle. In particular, the analysis of the literature suggests some interesting research challenges dealing with the design and the development of:

- Portable frameworks for workload monitoring and profiling.
- Systematic approaches toward workload characterization to be exploited in resource management strategies.
- Management systems for workload scheduling in real cloud environments that address the heterogeneity and variability in the resource requirements.
- Failure-aware resource provisioning and scheduling mechanisms that improve workload reliability.

Finally, a major issue faced by the research in cloud environments is the lack of publicly available large-scale workload measurements. In general, providers and users are very reluctant to disclose data about their workloads to avoid leakage of competitive and confidential information. Nevertheless, the availability of this data would be very beneficial for accelerating cloud deployments.

References

1. Alhamazani K, Ranjan R, Mitra K, Rabhi F, Jayaraman P, Khan SU, Guabtini A, Bhatnagar V (2015) An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. *Computing* 97(4):357–377
2. Alonso J, Trivedi K (2015) Software rejuvenation and its application in distributed systems. In: Bruneo D, Distefano S (eds) *Quantitative assessments of distributed systems: methodologies and techniques*, Wiley, pp 301–325
3. Araujo J, Matos R, Alves V, Maciel P, de Souza FV, Matias R, Trivedi KS (2014) Software aging in the Eucalyptus cloud computing infrastructure: characterization and rejuvenation. *ACM J Emerg Technol Comput Syst* 10(1):11:1–11:22
4. Ardagna C, Damiani E, Frati F, Rebecani D, Ughetti M (2012) Scalability patterns for Platform-as-a-Service. In: *Proceedings of the 5th international conference on cloud computing—CLOUD’12*, IEEE, pp 718–725
5. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. *Commun ACM* 53(4):50–58
6. Atikoglu B, Xu Y, Frachtenberg E, Jiang S, Paleczny M (2012) Workload analysis of a large-scale key-value store. In: *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on measurement and modeling of computer systems*, ACM, pp 53–64
7. Azmandian F, Moffie M, Dy JG, Aslam JA, Kaeli DR (2011) Workload characterization at the virtualization layer. In: *Proceedings of the 19th international symposium on modeling, analysis simulation of computer and telecommunication systems—MASCOTS’11*, IEEE, pp 63–72
8. Bala A, Chana I (2015) Intelligent failure prediction models for scientific workflows. *Expert Syst Appl* 42(3):980–989
9. Beaumont O, Eyraud-Dubois L, Lorenzo del Castillo JA (2014) Analyzing real cluster data for formulating allocation algorithms in cloud platforms. In: *Proceedings of the 26th international symposium on computer architecture and high performance computing—SBAC-PAD*, IEEE, pp 302–309
10. Bi J, Zhu Z, Tian R, Wang Q (2010) Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In: *Proceedings of the 3rd international conference on cloud computing—CLOUD’10*, IEEE, pp 370–377
11. Birke R, Chen LY, Smirni E (2014) Multi-resource characterization and their (in)dependencies in production datacenters. In: *Proceedings of the network operations and management symposium—NOMS’14*, IEEE
12. Bruneo D, Distefano S, Longo F, Puliafito A, Scarpa M (2013) Workload-based software rejuvenation in cloud systems. *IEEE Trans Comput* 62(6):1072–1085
13. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Gener Comput Syst* 25(6):599–616
14. Byun EK, Kee YS, Kim JS, Maeng S (2011) Cost optimized provisioning of elastic resources for application workflows. *Future Gener Comput Syst* 27(8):1011–1026
15. Calero JMA, Aguado JG (2015) Comparative analysis of architectures for monitoring cloud computing infrastructures. *Future Gener Comput Syst* 47:16–30
16. Chen S, Ghorbani M, Wang Y, Bogdan P, Pedram M (2014) Trace-based analysis and prediction of cloud computing user behavior using the fractal modeling technique. In: *Proceedings of the 7th international congress on big data*, IEEE, pp 733–739
17. Chen W, Ferreira da Silva R, Deelman E, Fahringer T (2015) Dynamic and fault-tolerant clustering for scientific workflows. *IEEE Trans Cloud Comput* 4(1):46–62
18. Chen X, Lu CD, Pattabiraman K (2014) Failure analysis of jobs in compute clouds: a Google cluster case study. In: *Proceedings of the 25th international symposium on software reliability engineering—ISSRE’14*, IEEE, pp 167–177
19. Chen X, Lu CD, Pattabiraman K (2014) Failure prediction of jobs in compute clouds: a Google cluster case study. In: *Proceedings of the IEEE international symposium on software reliability engineering workshops—ISSREW’14*, pp 341–346

20. Chen Y, Ganapathi AS, Griffith R, Katz RH (2010) Analysis and lessons from a publicly available Google cluster trace. Technical report UCB/EECS-2010-95, EECS Department, University of California, Berkeley
21. Chen Y, Alspaugh S, Katz R (2012) Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads. *Proc VLDB Endow* 5(12):1802–1813
22. Cheng D, Jiang C, Zhou X (2014) Heterogeneity-aware workload placement and migration in distributed sustainable datacenters. In: *Proceedings of the 28th international symposium on parallel and distributed processing—IPDP’14*, IEEE, pp 307–316
23. Cotroneo D, Natella R, Pietrantonio R, Russo S (2014) A survey of software aging and rejuvenation studies. *ACM J Emerg Technol Comput Syst* 10(1):8:1–8:34
24. de Oliveira GSS, Ribeiro E, Ferreira DA, Araujo A, Holanda MT, Walter MPF (2013) ACOsched: a scheduling algorithm in a federated cloud infrastructure for bioinformatics applications. In: *Proceedings of the international conference on bioinformatics and biomedicine—BIBM’13*, IEEE, pp 8–14
25. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
26. Di S, Kondo D, Cappello F (2014) Characterizing and modeling cloud applications/jobs on a Google data center. *J Supercomput* 69(1):139–160
27. Di Martino C, Kalbarczyk Z, Iyer RK, Goel G, Sarkar S, Ganesan R (2014) Characterization of operational failures from a business data processing SaaS platform. In: *Proceedings of the 36th international conference on software engineering companion—ICSE’14*, ACM, pp 195–204
28. Do AV, Chen J, Wang C, Lee YC, Zomaya AY, Zhou BB (2011) Profiling applications for virtual machine placement in clouds. In: *Proceedings of the 4th international conference on cloud computing—CLOUD’11*, IEEE, pp 660–667
29. Du J, Sehrawat N, Zwaenepoel W (2011) Performance profiling of virtual machines. *ACM SIGPLAN Not* 46(7):3–14
30. Duan R, Prodan R, Li X (2014) Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds. *IEEE Trans Cloud Comput* 2(1):29–42
31. Dutta S, Gera S, Verma A, Viswanathan B (2012) Smartscale: automatic application scaling in enterprise clouds. In: *Proceedings of the 5th international conference on cloud computing—CLOUD’12*, IEEE, pp 221–228
32. Fatema K, Emeakaroha VC, Healy PD, Morrison JP, Lynn T (2014) A survey of Cloud monitoring tools: taxonomy, capabilities and objectives. *J Parallel Distrib Comput* 74(10):2918–2933
33. Fiondella L, Gokhale SS, Mendiratta VB (2013) Cloud incident data: an empirical analysis. In: *Proceedings of the international conference on cloud engineering—IC2E’13*, IEEE, pp 241–249
34. Ganapathi A, Yanpei C, Fox A, Katz R, Patterson D (2010) Statistics-driven workload modeling for the Cloud. In: *Proceedings of the 26th international conference on data engineering workshops—ICDEW’10*, IEEE, pp 87–92
35. Garraghan P, Townend P, Xu J (2014) An empirical failure-analysis of a large-scale cloud computing environment. In: *Proceedings of the 15th international symposium on high-assurance systems engineering—HASE’14*, IEEE, pp 113–120
36. Ghorbani M, Wang Y, Xue Y, Pedram M, Bogdan P (2014) Prediction and control of bursty cloud workloads: a fractal framework. In: *Proceedings of the international conference on hardware/software codesign and system synthesis—CODES’14*, ACM, pp 12:1–12:9
37. Grozev N, Buyya R (2015) Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments. *Comput J* 58(1):1–22
38. Han R, Ghanem MM, Guo L, Guo Y, Osmond M (2014) Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Gener Comput Syst* 32:82–98
39. Huang J, Nicol DM (2013) Trust mechanisms for cloud computing. *J Cloud Comput* 2(1):1–14
40. Huang D, He B, Miao C (2014) A survey of management in multi-tier web applications. *IEEE Commun Surv Tutor* 16(3):1574–1590
41. Iosup A, Ostermann S, Yigitbasi MHJ, Prodan R, Fahringer T, Epema D (2011) Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans Parallel Distrib Syst* 22(6):931–945

42. Jayasinghe D, Malkowski S, Li J, Wang Q, Wang Z, Pu C (2014) Variations in performance and scalability: an experimental study in IaaS clouds using multi-tier workloads. *IEEE Trans Serv Comput* 7(2):293–306
43. Jennings B, Stadler R (2015) Resource management in clouds: survey and research challenges. *J Netw Syst Manage* 23(3):567–619
44. Jiang HJ, Huang KC, Chang HY, Gu DS, Shih PJ (2011) Scheduling concurrent workflows in HPC cloud through exploiting schedule gaps. In: Xiang Y, Cuzzocrea A, Hobbs M, Zhou W (eds) *Algorithms and architectures for parallel processing, lecture notes in computer science*, vol 7016, Springer, pp 282–293
45. Jiang T, Zhang Q, Hou R, Chai L, Mckee SA, Jia Z, Sun N (2014) Understanding the behavior of in-memory computing workloads. In: *Proceedings of the international symposium on workload characterization—IISWC’14*, IEEE, pp 22–30
46. Johnson SK, Huizenga G, Pulavarty B (2005) *Performance tuning for linux servers*. IBM RedBooks
47. Juan DC, Li L, Peng HK, Marculescu D, Faloutsos C (2014) Beyond Poisson: modeling inter-arrival time of requests in a datacenter. In: Tseng V, Ho T, Zhou ZH, Chen AP, Kao HY (eds) *Advances in knowledge discovery and data mining, lecture notes in computer science*, vol 8444, Springer, pp 198–209
48. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K (2013) Characterizing and profiling scientific workflows. *Future Gener Comput Syst* 29(3):682–692
49. Kaviani N, Wohlstadter E, Lea R (2011) Profiling-as-a-Service: adaptive scalable resource profiling for the cloud in the cloud. In: Kappel G, Maamar Z, Motahari-Nezhad H (eds) *Service-oriented computing, lecture notes in computer science*, vol 7084, Springer, pp 157–171
50. Kessaci Y, Melab N, Talbi EG (2012) A Pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation. *Cluster Comput* 16(3):451–468
51. Khan A, Yan X, Tao S, Anerousis N (2012) Workload characterization and prediction in the cloud: a multiple time series approach. In: *Proceedings of the 13th network operations and management symposium—NOMS’12*, IEEE, pp 1287–1294
52. Khazaei H, Misic J, Misic VB (2012) Performance analysis of cloud computing centers using $M/G/m/m+r$ queuing systems. *IEEE Trans Parallel Distrib Syst* 23(5):936–943
53. Li S, Ren S, Yu Y, Wang X, Wang L, Quan G (2012) Profit and penalty aware scheduling for real-time online services. *IEEE Trans Ind Inform* 8(1):78–89
54. Lim HC, Babu S, Chase JS, Parekh SS (2009) Automated control in cloud computing: challenges and opportunities. In: *Proceedings of the 1st workshop on automated control for datacenters and clouds—ACDC’09*, ACM, pp 13–18
55. Liu Z, Cho S (2012) Characterizing machines and workloads on a Google cluster. In: *Proceedings of the 41st international conference on parallel processing workshops—ICPPW’12*, IEEE, pp 397–403
56. Liu S, Quan G, Ren S (2010) On-line scheduling of real-time services for cloud computing. In: *Proceedings of the 6th world congress on services*, IEEE, pp 459–464
57. Mao M, Humphrey M (2013) Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In: *Proceedings of the 27th international symposium on parallel and distributed processing—IPDPS’13*, IEEE, pp 67–78
58. Marshall P, Keahey K, Freeman T (2010) Elastic site: using clouds to elastically extend site resources. In: *Proceedings of the 10th IEEE/ACM international symposium on cluster, cloud and grid computing—CCGrid’10*, IEEE, pp 43–52
59. Mauch V, Kunze M, Hillenbrand M (2013) High performance cloud computing. *Future Gener Comput Syst* 29(6):1408–1416
60. Meng S, Liu L (2013) Enhanced Monitoring-as-a-Service for effective cloud management. *IEEE Trans Comput* 62(9):1705–1720
61. Mishra AK, Hellerstein JL, Cirne W, Das CR (2010) Towards characterizing cloud backend workloads: insights from Google compute clusters. *ACM SIGMETRICS Perform Eval Rev* 37(4):34–41

62. Moreno-Vozmediano RS, Montero RS, Llorente IM (2011) Multicloud deployment of computing clusters for loosely coupled MTC applications. *IEEE Trans Parallel Distrib Syst* 22(6):924–930
63. Mueller J, Palma D, Landi G, Soares J, Parreira B, Metsch T, Gray P, Georgiev A, Al-Hazmi Y, Magedanz T, Simoes P (2014) Monitoring as a service for cloud environments. In: *Proceedings of the 5th international conference on communications and electronics–ICCE’14*, IEEE, pp 174–179
64. Pacheco-Sanchez S, Casale G, Scotney B, McClean S, Parr G, Dawson S (2011) Markovian workload characterization for QoS prediction in the cloud. In: *Proceedings of the 4th international conference on cloud computing–CLOUD’11*, IEEE, pp 147–154
65. Pandey S, Wu L, Guru SM, Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: *Proceedings of the 24th international conference on advanced information networking and applications–AINA’10*, IEEE, pp 400–407
66. Patel J, Jindal V, Yen IL, Bastani F, Xu J, Garraghan P (2015) Workload estimation for improving resource management decisions in the cloud. In: *Proceedings of the 12th international symposium on autonomous decentralized systems–ISADS’15*, IEEE, pp 25–32
67. Raicu I (2009) Many-task computing: bridging the gap between high-throughput computing and high-performance computing. Ph.D. thesis, University of Chicago
68. Reiss C, Wilkes J, Hellerstein JL (2011) Google cluster-usage traces: format+schema. Google Inc
69. Reiss C, Tumanov A, Ganger GR, Katz RH, Kozuch MA (2012) Heterogeneity and dynamics of clouds at scale: Google trace analysis. In: *Proceedings of the 3rd symposium on cloud computing–SoCC’12*, ACM, pp 7:1–7:13
70. Ren G, Tune E, Moseley T, Shi Y, Rus S, Hundt R (2010) Google-wide profiling: a continuous profiling infrastructure for data centers. *IEEE Micro* 30(4):65–79
71. Ren Z, Xu X, Wan J, Shi W, Zhou M (2012) Workload characterization on a production Hadoop cluster: a case study on Taobao. In: *Proceedings of the international symposium on workload characterization–IISWC’12*, IEEE, pp 3–13
72. Sadooghi I, Palur S, Anthony A, Kapur I, Belagodu K, Purandare P, Ramamurty K, Wang K, Raicu I (2014) Achieving efficient distributed scheduling with message queues in the cloud for many-task computing and high-performance computing. In: *Proceedings of the 14th IEEE/ACM international symposium on cluster, cloud and grid computing–CCGrid’14*, IEEE, pp 404–413
73. Samak T, Gunter D, Goode M, Deelman E, Juve G, Silva F, Vahi K (2012) Failure analysis of distributed scientific workflows executing in the cloud. In: *Proceedings of the 8th international conference on network and system management–CNSM’12*, IEEE, pp 46–54
74. Schad J, Dittrich J, Quiané-Ruiz JA (2010) Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc VLDB Endow* 3(1–2):460–471
75. Sharma U, Shenoy P, Towsley DF (2012) Provisioning multi-tier cloud applications using statistical bounds on sojourn time. In: *Proceedings of the 9th international conference on autonomic computing–ICAC’12*, ACM, pp 43–52
76. Shen S, van Beek V, Iosup A (2015) Statistical characterization of business-critical workloads hosted in cloud datacenters. In: *Proceedings of the 15th IEEE/ACM international symposium on cluster, cloud and grid computing–CCGrid’15*, IEEE
77. Singh R, Sharma U, Cecchet E, Shenoy P (2010) Autonomic mix-aware provisioning for non-stationary data center workloads. In: *Proceedings of the 7th international conference on autonomic computing–ICAC’10*, ACM, pp 21–30
78. Solis Moreno I, Garraghan P, Townend P, Xu J (2014) Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. *IEEE Trans Cloud Comput* 2(2):208–221
79. Somasundaram TS, Govindarajan K (2014) CLOUDRB: a framework for scheduling and managing high-performance computing (HPC) applications in science cloud. *Future Gener Comput Syst* 34:47–65

80. Spicuglia S, Björkqvist M, Chen LY, Serazzi G, Binder W, Smirni E (2013) On load balancing: a mix-aware algorithm for heterogeneous systems. In: Proceedings of the 4th international conference on performance engineering–ICPE’13, ACM, pp 71–76
81. Tai J, Zhang J, Li J, Meleis W, Mi N (2011) ArA: adaptive resource allocation for cloud computing environments under bursty workloads. In: Proceedings of the 30th international conference on performance computing and communications–IPCCC’11, IEEE, pp 1–8
82. Tickoo O, Iyer R, Illikkal R, Newell D (2010) Modeling virtual machine performance: challenges and approaches. *SIGMETRICS Perform Eval Rev* 37(3):55–60
83. Van den Bossche R, Vanmechelen K, Broeckhove J (2010) Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: Proceedings of the 3rd international conference on cloud computing–CLOUD’10, IEEE, pp 228–235
84. Warneke D, Kao O (2011) Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE Trans Parallel Distrib Syst* 22(6):985–997
85. Weingärtner R, Bräscher GB, Westphal CB (2015) Cloud resource management: a survey on forecasting and profiling models. *J Netw Comput Appl* 47:99–106
86. Wu F, Wu Q, Tan Y (2015) Workflow scheduling in cloud: a survey. *J Supercomput* 71(9):3373–3418
87. Yang H, Luan Z, Li W, Qian D (2012) MapReduce workload modeling with statistical approach. *J Grid Comput* 10(2):279–310
88. Yin J, Lu X, Chen H, Zhao X, Xiong NN (2014) System resource utilization analysis and prediction for cloud based applications under bursty workloads. *Inform Sci* 279:338–357
89. Zhan ZH, Liu XF, Gong YJ, Zhang J, Chung HSH, Li Y (2015) Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput Surv* 47(4):63:1–63:33
90. Zhang F, Cao J, Li K, Khan SU, Hwang K (2014) Multi-objective scheduling of tasks in cloud platforms. *Future Gener Comput Syst* 37:309–320
91. Zhang Q, Zhani MF, Boutaba R, Hellerstein JL (2014) Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE Trans Cloud Comput* 2(1):14–28
92. Zhao Y, Fei X, Raicu I, Lu S (2011) Opportunities and challenges in running scientific workflows on the cloud. In: Proceedings of the international conference on cyber-enabled distributed computing and knowledge discovery–CyberC’11, IEEE, pp 455–462
93. Zhu Q, Agrawal G (2010) Resource provisioning with budget constraints for adaptive applications in cloud environments. In: Proceedings of the 19th international symposium on high performance distributed computing–HPDC’10, ACM, pp 304–307

Reproducibility of Software Bugs

Basic Concepts and Automatic Classification

Flavio Frattini, Roberto Pietrantuono and Stefano Russo

Abstract Understanding software bugs and their effects is important in several engineering activities, including testing, debugging, and design of fault containment or tolerance methods. Dealing with hard-to-reproduce failures requires a deep comprehension of the mechanisms leading from bug activation to software failure. This chapter surveys taxonomies and recent studies about bugs from the perspective of their reproducibility, providing insights into the process of bug manifestation and the factors influencing it. These insights are based on the analysis of thousands of bug reports of a widely used open-source software, namely MySQL Server. Bug reports are automatically classified according to reproducibility characteristics, providing figures about the proportion of hard to reproduce bug their features, and evolution over releases.

1 Introduction

Software is commonly characterized by the presence of *defects*—imperfections that cause systems to improperly deliver the service they are intended for, resulting in what is called a *failure*. *Bugs* are usually meant as defects in the code, thus with a more narrow meaning than defects. Bugs have been classified according to various characteristics from the perspective of software engineering, usually with the aim of supporting product and process improvement activities by defect analysis [1]. Several schemes are available in the literature: some relevant examples are the HP

F. Frattini (✉) · R. Pietrantuono · S. Russo
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione,
Università degli Studi di Napoli Federico II, Via Claudio, 21, 80125 Napoli, Italy
e-mail: flavio.frattini@unina.it

R. Pietrantuono
e-mail: roberto.pietrantuono@unina.it

S. Russo
e-mail: stefano.russo@unina.it

classification [2], the IEEE 1044 classification [3], and Orthogonal Defect Classification (ODC) [4].

As software systems grow in size and complexity and are increasingly used for mission- and business-critical applications, attention is being devoted to the bug manifestation process and its features with the aim of gaining a deeper understanding of complex erroneous behaviors. Knowledge of the factors influencing the chain by which a bug is activated, propagates into the system, and reaches its external interface,¹ serves, for instance, to reproduce the exposure of subtle bugs to then locate and remove them.

Not all bugs are easily reproducible.² In fact, software may behave differently under apparently identical conditions. Attempting to recreate the same conditions and repeating the steps that led to an observed failure is the usual way to try to reproduce a bug. However, some bugs require rare combinations and/or relative timings of inputs, or a specific state to be reproduced; or there may be a long delay between the fault activation and the failure occurrence. In other cases, the activation of a fault makes the system traverse several error states. In all such cases it is difficult to identify the inputs and the conditions to reproduce the failure. Moreover, concurrent programs are known to suffer from the *probe effect*, an “alteration in the frequency of run-time errors observed when delays are introduced”: this effect can mask synchronization errors [6]. Even when the input values and the system internal state for an observed failure are known, there are cases where the failure occurs only under specific environmental conditions, and “as the tester tries to reproduce them, the environment changes, making the failure disappear” [7].

In this chapter, we survey taxonomies and experimental studies which led to the current understanding of bugs reproducibility (Sect. 2). Then, we describe a procedure to analyze a bug repository from the reproducibility perspective (Sect. 3). The procedure is applied to thousands of bugs reported in the MySQL database management system. Results of the classification and prediction process are presented, providing insights into the process of bug manifestation and the factors influencing it (Sect. 4). This chapter ends with a brief discussion of the results (Sect. 5).

2 Studies on Bug Reproducibility

Reproducibility is the basic criterion of several bugs classifications. In the 1980s, Gray distinguished **Bohrbugs** and **Heisenbugs** [7]: Bohrbugs exhibit a deterministic behavior, hence they can be detected and removed with testing and debugging—these are also known as **hard** or **solid** ones, for which the failure occurrence is always reproducible; Heisenbugs cause transient failures, which may not manifest on a software

¹We follow the well-established notion of *fault–error–failure chain* [5]: a *software fault* (bug) is a defect in the application code; when activated, faults cause *errors*; errors may lead to *failures*.

²We use the expression “bug reproducibility”—widely used in the literature—to indicate the reproducibility of the failure caused by the bug.

re-execution under the same input, thus appearing as “nondeterministic”. These are known as **soft** or **elusive** faults, whose activation is not systematically reproducible; they may be extremely difficult to identify through testing. Gray named the former class alluding to the physicist Niels Bohr, who developed the atomic model, and the latter class referring to the physicist Werner Heisenberg, who formulated the uncertainty principle.

Grottke and Trivedi [8, 9] recognized that the term Heisenbug had originally been coined in the 1960s by Lindsay (while working with Gray), referring to “bugs in which clearly the behavior of the system is incorrect, and when you try to look to see why it’s incorrect, the problem goes away”; this is a different definition than the one published later by Gray. The class of bugs defined based on the notion of non-determinism is not the same as soft faults (as claimed by Gray).

The nomenclature that Grottke and Trivedi revised in the past decade introduces the category of **Mandelbugs** in lieu of Heisenbugs: by alluding to the mathematician Benoit Mandelbrot and his work on fractal geometry, the name somehow suggests a form of chaotic system behavior [8]. Unlike Heisenbugs, Mandelbugs are defined in terms of *inherent* faults properties, that is, faults *able* to cause failures which are not systematically reproducible.³

Four factors of complexity in the failure reproduction process are pinpointed by Grottke, Nikora, and Trivedi as responsible for a bug to be classified as Mandelbug [10]: (i) a time lag between the fault activation and the failure occurrence; (ii) interactions of the software application with hardware, operating system, or other applications running concurrently on the same system; (iii) influence of the timing of inputs and operations; (iv) influence of the sequencing of operations.

Several studies show that Bohrbugs are much more common than Mandelbugs: for instance, 463 out of 637 bugs are classified as Bohrbugs in [11]; 547 over 852 bugs are considered as always reproducible in [12]. Other studies examine bugs according to characteristics attributable to Bohr- or Mandelbugs (factors such as concurrency, resource management), but adopting a different terminology [13–15].

Mandelbugs are often related to unusual hardware conditions (rare or transient device faults), limit conditions (out of storage, counter overflow, lost interrupt, etc.), or race conditions [7]. Changing the environment and removing the chaotic state—i.e., resetting the program to a proper state—is likely to enable the application to work. This explains why and how some fault tolerance methods work. For example, checkpointing is a technique that periodically saves a snapshot of an application in permanent storage, enabling it to restart after a failure from an internal state that should allow the proper execution. Similarly, replicating the execution in two

³Gray states: “Heisenbug may elude a bugcatcher for years of execution. The bugcatcher may perturb the situation just enough to make it disappear. This is analogous to Heisenberg’s Uncertainty Principle in physics.” Indeed, Heisenberg ascribed the uncertainty principle to the disturbance triggered by the act of measuring (observer effect). However, this argument is recognized to be misleading by modern physicists: the principle states a fundamental property of conjugate entities; it is not a statement about the observational success of the technology. Curiously, Mandelbugs are closer than Heisenbugs to the principle these were originally meant to resemble.

different environments can result in the proper execution of a replica even if the other fails.

A third category of bugs introduced in [8] are **aging-related bugs** (ARBs): they cause the accumulation of errors in the running application or in the system-internal environment, which result in an increased failure rate and/or degraded performance. They are viewed as a class of *Mandelbugs*. It is worth noting that ARBs are hard to reproduce due to both the system internal state and the time necessary for the failure to manifest. They might be easy to activate, but the time they take to manifest themselves as failures make them hard to observe during testing.

In [16–20], we focused on the aging problem and on aging-related bugs, distinguishing memory-related problems (e.g., leaks), storage-related (e.g., fragmentation), wrong management of other resources (e.g., handles, locks), and numerical errors, observing an impact of approximately 5%.

Other researchers focused on the ephemerality of bugs. Chandra and Chen [21] distinguish environment-independent and environment-dependent bugs, and further classify the latter as transient and non-transient. **Environment-independent bugs** occur independently of the operating environment; if a bug of this kind occurs with a specific workload, it will always occur with that workload. **Environment-dependent bugs** depend on the operating environment: the subset of **transient bugs** may appear only in some executions; on the contrary, **non-transient bugs** occur always, in a specific environment. Environmental bugs are also examined in [22], where authors focus on how to reproduce transient bugs by varying factors of the execution environment. Clearly, there is an overlap between Mandelbugs/Heisenbugs and environment-dependent transient bugs: in both cases the main characteristics are the apparent aleatory occurrence and the difficulty of reproduction. In [23], environment-dependent bugs are categorized to conjecture a possible fault injection framework for emulating environmental influence on systems failure.

Concurrency bugs are discussed in [13]: the authors select randomly 105 such bugs from the repositories of 4 open-source applications, showing that their reproducibility is very hard due to the large number of variables involved or to the dynamics of memory accesses. Fonseca et al. [24] consider concurrency bugs from MySQL Server to understand their effects and how they can be detected or tolerated. In this case, it is shown that concurrency bugs are likely to remain latent and to corrupt data structures, but only later cause actual failures. In [14], it is shown that concurrency bugs in operating system code commonly requires more effort to be identified. Moreover, the analysis reveals that semantic bugs are the dominant root cause and, as software evolves, their number increases while memory-related bugs decrease. *Aging-related concurrency bugs* are shown in [18] to cause performance decrease over time in a hard-to-predict way, and the failure rate does not depend on the instantaneous and/or mean accumulated work.

Performance bugs are analyzed by Nistor et al. [25]. It is shown that: their fixing may introduce new functional bugs; they appear more difficult to fix than nonperformance bugs; most performance bugs are discovered through code reasoning rather than because of users experiencing failures due to bugs. In [26], some rules are extracted from a set of bugs in order to identify performance problems in MySQL,

Apache, and Mozilla applications. *Security bugs* are studied in [27] with reference to the Firefox software. It is shown that they require more experienced debuggers and their fixes are more complex than the fixes of other kinds of bugs. For open source software, an empirical study based on the automated analysis of 29,000 bugs [28] shows that most bugs are semantic and that bugs related to memory are still the major component, despite the recent introduction of detection tools.

These studies examine several high-level factors that can be (directly or indirectly) attributed to the bug manifestation process, considering aspects such as concurrency, memory, timing, interaction with the operating system or other applications, performance, security, resource leaks, wrong error handling. Overall, the literature highlights the relevance of the topic, but there is insufficient knowledge of essential bug reproducibility characteristics. With respect to other bug characteristics (e.g., detection or closing times, bug location, i.e., source code, fixing commits, severity, etc.), that are more amenable to be analyzed automatically, a relevant problem is the absence of approaches to automatically distinguish bugs according to some reproducibility characteristic. In the following, we attempt to address this issue by proposing an automatic classification from reports, in order to enable future analyses on wider datasets, so as to improve the knowledge about bug reproducibility similarly to other defect analysis research areas.

3 Analysis of Bug Reproducibility

We describe a procedure to analyze bug reports from the *reproducibility* perspective, i.e., considering how they can be exposed and reproduced. We refer to a system model where we distinguish the *application* under analysis, its *execution environment*, the *workload*, and the *user* submitting it. The analysis is meant to discriminate bugs depending on whether, under a given workload, they are *always reproducible*, or *not always reproducible*, i.e., they may occur or not depending on the state of the execution environment. The latter is considered as the hardware resources where the application is deployed (processors, I/O devices, network), and software running concurrently on each node—including operating systems, middleware layers, virtualization layers, and other applications sharing the hardware resources.

The user (not necessarily a human) interacts with the application by submitting workload requests and getting the results. We assume a workload request represented as a generic request for service (e.g., a query to a DBMS), characterized by a type (e.g., query type, like INSERT), and by a set of input parameters (e.g., values of an INSERT), in turn characterized by a type and a value. To accomplish a well-defined task, the user can submit a sequence of serial/concurrent requests. We denote with *environment* the union of the *execution environment* and the *user*.

According to this model, we define two categories of bug manifestation.

- **Workload-dependent (WL):** *the bug manifestation is “workload-dependent” if resubmitting (at most a subset of) the workload requests that caused a failure always produces the same failure, for every valid state of the environment (i.e.,*

for every state of the environment in which the traversed application states are allowed to occur) and for every admissible user inputs' timing/ordering in each request of the sequence. In this sense, we talk about *always reproducible bugs*.

- An **example** of this kind of bug is the bug 13894 of MySQL Server,⁴ which reports “Server crashes on update of CSV table” and also includes the sequence of statements that crashes the server. The reporter specifies that every time the sequence is repeated the MySQL Server crashes.
- **Environment-dependent (ENV)**: *the bug manifestation is “environment-dependent” if resubmitting (at least a subset) of workload requests that caused a failure, there exist at least one (valid) state of the environment or user inputs' timing/ordering⁵ causing the same failure to not be reproduced.* Note that, unlike Mandelbugs, these do not include complex but “deterministic” bugs, namely those bugs requiring a very complex workload but that, under such workload, always reappear: in this categorization, such bugs fall in the WL category. We talk about *not-always-reproducible bugs*.
 - As an **example** consider the bug 18306 of MySQL Server “MySQL crashes and restarts using subquery”; the report is about a delete operation; the reporter specifies that “The DELETE query works X times and then mysql crashes.” Thus, the bug does not manifest every time a specific load is submitted to the system, further conditions are to be forced in order to reproduce the bug, instead.

The analysis aims at providing insights into the presence of workload-dependent or environment-dependent bugs and of their evolution over several software versions. The following procedure is followed: (i) first, *manual classification* of bugs as either WL or ENV is performed, by inspecting bug reports of the target software; (ii) then, we build predictors for *automatic classification* that takes bug reports as input, process the text contained in the report by text mining techniques, and then automatically classify the report as either WL or ENV; (iii) based on the predicted values, further insight about WL versus ENV bugs on an enlarge dataset and on various versions of the software is obtained; (iv) the most discriminating textual features are also examined, so as to figure out the characteristics of a bug more related to the bug manifestation type. The analysis steps are summarized in the next sections.

3.1 Manual Classification

Each problem report is manually inspected to check it documents a real and unique bug: documentation and build issues, problems turning out to be operator errors,

⁴MySQL Bugs—<https://bugs.mysql.com>.

⁵*Valid* means admissible, compatible environment state with reference to the input requests; in the case of user, it means that the same workload request(s) could be submitted in different timing/ordering producing the same result.

and requests for software enhancements (not erroneously marked as such) are put in a NOT_BUG class and discarded. Then, reports containing insufficient details to classify the bug (e.g., a bug closed as soon as it was reported, corrected by a new minor release) are assigned to the class UNKNOWN, and discarded.

The remaining reports are searched for the following indications:

- (i) what inputs are required (the failure-causing workload); reports often provide it in the test case and/or in the steps to repeat the failure occurrence;
- (ii) what is the application and the environment configuration;
- (iii) if the corresponding failure is observed to be always repeatable (i.e., workload-dependent) or not (i.e., environment-dependent) by the bug reporter or assignee;
- (iv) in the case not always reproducible, what are the conditions hiding the bug manifestation (i.e., if they are related to the execution environment, or to particular user actions, such as timing of inputs).

Based on this, we assign a bug report to the workload-dependent (WL) or environment-dependent (ENV) classes. It is worth noting that the manual analysis is based exclusively on fixed bugs (as in [12, 15, 16]), since, for bugs that have not yet been fixed, the reports may contain inaccurate or incomplete information. While this allows relying on more stable information, results will not refer to non-closed bugs, which could, in principle, have different patterns of WL or ENV bugs. Moreover, although results of the manual classification are cross-checked by the authors, we cannot exclude, as any paper where manual inspection is needed, possible classification mistakes that can affect the results.

3.2 Automatic Classification

The bug classification aims at automatically assigning a bug to a class by analyzing its report. It consists of two steps: text processing and classification.

3.2.1 Text Processing

The automatic classification is carried out by means of predictors, using bug report textual description as input features. Some preliminary steps are required to render text suitable for processing by classifiers. Specifically, in text mining, each term occurring in the document is a potential dimension; to avoid dealing with a useless large number, we apply common reduction methods [29, 30]:

1. **Tokenization:** a textual string is divided into a set of tokens; a token is a block of text considered as a useful part of the unstructured text (it often corresponds to a word, but it might also be an entire sentence). Tokenization includes filtering out meaningless symbols, like punctuation, brackets, and makes all letters lowercase.

2. **Stop-word removal**; this consists of removing the terms such as propositions, articles, conjunctions, which do not convey much useful information and may appear frequently—thus biasing the classification algorithms.
3. **Stemming**: reducing common words to a single term, e.g., “computerized”, “computerize”, and “computation” are all reduced to “compute”.

3.2.2 Classifiers

To classify the bugs, we adopt two classifiers widely used in the literature of defect prediction, namely *Naïve Bayes* and *Bayesian Network*.

A Naïve Bayes (NB) classifier estimates the a posteriori probability of the hypothesis H to be assessed (e.g., “bug is ENV”), that is the probability that H is true given an observed evidence E . This is given the probability to observe E under the hypothesis H multiplied by the a priori probability of the hypothesis H (i.e., when no evidence is available) over the probability of evidence E :

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}. \quad (1)$$

The evidence E consists of *features* used to classify, namely the *attributes* of the instances to classify, which are extracted through the mentioned text mining technique. A fundamental assumption of a Naïve Bayes classifier is that each feature E_i is conditionally independent of any other feature E_j , $j \neq i$. Under this assumption, the a posteriori probability can be obtained as:

$$P(H|E) = \left[\prod_i P(E_i|H) \right] \frac{P(H)}{P(E)}. \quad (2)$$

This assumption is apparently oversimplifying, since features usually exhibit some degree of dependence among each other. Nevertheless, the Naïve Bayes classifier performs well even when this assumption is largely violated [31].

A Bayesian network (*BayesNet*) is a directed acyclic graphical model representing a set of random variables and their conditional dependency (graph nodes and edges, respectively). A conditional dependency exists between two nodes if the corresponding random variables are not conditionally independent. It is assumed that:

$$P(\text{node}|\text{parents plus any other nondescendants}) = P(\text{node}|\text{parents}). \quad (3)$$

The joint probability distribution for a set of random variables X_1, \dots, X_n is:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i|X_{i-1}, \dots, X_1) = \prod_{i=1}^n P(X_i|X_i\text{'s parents}). \quad (4)$$

Equation 4 is used to compute the probability of a hypothesis H represented by a node of the network, given the conditional probability distributions of each node, and given a set of observed values.

Classifiers are evaluated by means of the common metrics *precision*, *recall*, and *F-measure*. We adopt a *k-fold cross-validation*, with $k = 3$ and 10 repetitions: the set is split into three disjoint sets, two of which are used for training, and the third one for testing. Accuracy metrics are then calculated. The procedure is repeated until each subset has been used as test set. Accuracy metrics of each step are averaged. All the steps are repeated ten times; each time the three sets are generated by randomly selecting reports from the set. Finally, the metrics values of each repetition are averaged to obtain the final accuracy evaluation.

For each data sample in the test set, the predicted class is compared with the actual class of the sample. Given a target class (e.g., ENV), samples of the test set belonging to the *target class* are *true positives* if they are correctly classified, and are *false negatives* otherwise. Similarly, samples belonging to the other class (i.e., WL) are denoted as *true negatives* if they are correctly classified, and as *false positives* otherwise. From these, we compute:

- **Precision (Pr):** Percentage of *true positives* (TP) that are classified as belonging to the target class (*true positives* and *false positives* (FP)):

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}). \quad (5)$$

- **Recall (Re):** Percentage of *true positives* that actually belong to the target class (*true positives* and *false negatives* (FN)):

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}). \quad (6)$$

- **F-measure (F):** Harmonic mean of *precision* and *recall*:

$$F\text{-measure} = (2 \cdot \text{Pr} \cdot \text{Re})/(\text{Pr} + \text{Re}). \quad (7)$$

The higher the precision and the recall (ideally, $\text{Pr} = \text{Re} = 1$), the higher the quality of the predictor, since it avoids false positives and false negatives.

4 Case Study: Analysis of MySQL Bugs

The procedure described in Sect. 3 is here applied to the MySQL Data Base Management System. This is chosen as case study since it is a modern complex software system widely adopted in business-critical contexts, and for which detailed bug reports are publicly available.

We use bug reports related to *MySQL Server* version 5.1 for manual classification and training. Then, the prediction is applied to bug reports from other versions of the software. We consider both versions preceding the 5.1 (MySQL Server 4.1 and 5.0)

and versions following the 5.1 (5.5 and 5.6). The aim is to figure out if there is a trend for the types of bugs over various versions, and to understand which modifications in the software are likely the cause of such a trend, if any.

4.1 Manual Classification

As a preliminary step, the following types of reports are excluded manually from subsequent inspection:

- reports not marked as *closed*, so as to proceed with descriptions only of solved bugs, relevant for the analysis;
- *duplicate* bugs, that is, from the description we deduced that the reported problem was caused by the same bug of another report already classified;
- bugs marked as *enhancement* or *feature request* in the “severity” field.

This step produces a set of 560 bug reports, which are manually inspected according to the steps reported in Sect. 3.1, considering the following reports’ sections:

- the textual description of the steps to repeat the failure;
- the textual discussion and comments of developers/users working on that bug;
- the final patch that has been committed, along with the description note in the change log;
- the attached files (e.g., test cases, environment configuration files).

The manual inspection identifies:

- 402 workload-dependent (WL) bugs;
- 86 environment-dependent (ENV) bugs;
- 44 reports classified as NOT_BUG;
- 28 reports classified as UNKNOWN, because they contained insufficient details.

We have that 82 % of classified bugs are WL, and the remaining 18 % are ENV. As in other studies, workload-dependent bugs are the large majority [10–12]. Nevertheless, the minority of environment-dependent bugs are those hard to reproduce and to fix.

4.2 Automatic Classification

The automatic classification is performed on the set of bugs manually categorized as WL or ENV; it includes the two steps of text processing and classification.

4.2.1 Text Processing

Text processing of reports is performed to identify the features and their occurrence. To identify the features the three operations of tokenization, stop-word removal, and

stemming are performed, as described in Sect. 3.2. 21,477 features are identified. For each of them it is counted the number of occurrences in each report.

4.2.2 Classifiers

The classifiers are trained and their accuracy is evaluated by means of threefold cross-validation. Results on the Naïve Bayes and Bayesian Network classifiers are reported in Table 1.

Results show that the Bayesian Network classifier presents a lower precision than the Naïve Bayes classifier, but recall is better. Overall, the F-measure reveals that the Bayesian Network is slightly better than the Naïve Bayes classifier.

We also report the confusion matrices, which show, for each class, the number of bugs correctly classified and the incorrectly classified instances. Tables 2 and 3 are related to the Naïve Bayes the Bayesian Network classifiers, respectively.

The tables show that most of the incorrect classifications are related to environment-dependent bugs. In the case of the Naïve Bayes classifier, 73 % of environment-dependent bugs were not correctly classified. This may be due to the reduced number of examples for training. However, the classification improves when using the Bayesian Network, both for WL bugs and for ENV bugs.

It is also interesting to consider which features are the most discriminating. The classifiers predict the type of a bug (i.e., workload-dependent or environment-dependent) based on the terms used in its report. During the training, the probability, for each feature, that a bug is of a certain type given that the feature appears in the description is computed. In Table 4, we report the most significant features for the prediction process.

Table 1 Results of the training

	Class	Precision	Recall	F-measure
<i>Naïve Bayes</i>	WL	0.94	0.81	0.87
	ENV	0.41	0.73	0.53
	Weighted	0.86	0.80	0.82
<i>Bayesian Network</i>	WL	0.90	0.90	0.90
	ENV	0.44	0.46	0.45
	Weighted	0.83	0.83	0.83

Table 2 Confusion matrix of Naïve Bayes classifier

Correct	Incorrect	
113	27	WL
7	19	ENV

In some cases there is just a root indicating a set of words; as an example, *concurr* implies that there may be words such as concurrent, concurrency, etc. Note that most features are commonly linked to hard-to-reproduce conditions, such as memory

Table 3 Confusion matrix of Bayesian Network

Correct	Incorrect	
125	15	WL
14	12	ENV

Table 4 Most significant features for prediction

Feature		
Thread	Memory	Concurr
Deadlock	Alloc	Wait
Run	Race	Start
Schedul	Valgrind	

issues, deadlocks, or race conditions. Valgrind is a tool for solving memory issues; thus, the feature with the same name is also related to memory issues.

It is worth noting that only 259 features, over the total 21,477, are actually used by the classifiers.

4.3 Prediction Results

The prediction of bug types is performed for the two versions preceding the one used for the training, and for the two following it: that is for MySQL Server versions 4.1, 5.0, 5.5, and 5.6. Results are shown in Figs. 1 and 2 for the Naïve Bayes classifier and the Bayesian Network, respectively. In the figures, results achieved by means of reports’ manual inspection (version 5.1) are marked differently.

As expected, workload-dependent bugs are more than environment-dependent ones. This is even more evident using the Bayesian network. For preceding versions, just 6.7 % of bugs are ENV. The small number of environment-dependent bugs is

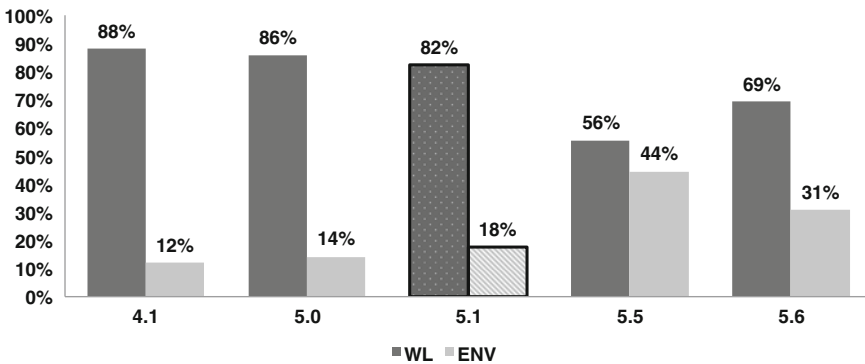


Fig. 1 Results on bug type prediction with Naïve Bayes classifier

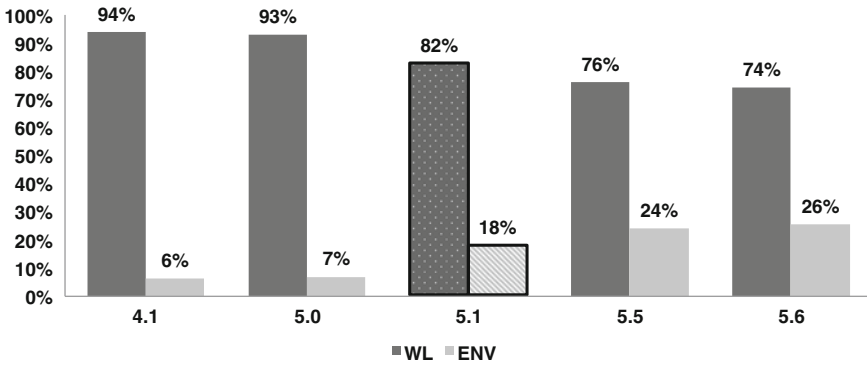


Fig. 2 Results of bug type prediction with *Bayesian Network*

probably related to the type of software considered. The DBMS, in its early versions, is not made up of many subsystems, each influencing the condition for which a fault is activated. Environment-dependent bugs are more common for operating systems, instead [12].

It is also worth noting the increasing trend of environment-dependent bugs’ percentage over versions, in line with the common opinion that the prevalence of simple bugs decreases with time [12]. While in version 4.1 just 6 % of the bugs are predicted as ENV, this percentage increases up to 26 % for version 5.6. This may be due to the greater complexity starting from version 5.5. In the documentation, it is specified that one of the main changes in later versions, with respect to previous ones, consists in multiple background I/O threads used to improve I/O performance. The status of such threads may represent an environmental condition hard to reproduce. Interestingly, we also found that *thread* is the most discriminating feature.

5 Discussion and Conclusion

Reproduction of software failures to locate and fix the corresponding bugs is not an easy task, but it is very important to make software reliable. We discussed the problem of bug manifestation and introduced the two classes of workload-dependent and environment dependent bugs. For the former class, by resubmitting the same workload requests that caused a failure, the same failure can be produced. For the latter, there exists at least one (valid) state of the environment or user inputs’ timing/ordering that may render the same failure unreproducible even when resubmitting the same workload request that caused the failure.

The analysis of the literature and the manual inspection of 560 bug reports showed that environment-dependent bugs are, commonly, in small number with respect to workload-dependent bugs. They are more difficult to fix, however. Also, studies in the scientific literature are usually related to few hundreds of bugs, given the difficulty of

inspection by hand. Thus, we applied text mining techniques and Bayesian classifiers in order to automate the classification process.

Results from classification models show that automatic classification can be performed with an F-measure up to 83 %, but with environment-dependent bugs being more difficult to discriminate. Future works will target the improvement of environment-dependent bug discrimination by exploiting other sources of information (e.g., complexity metrics). Results from the prediction of MySQL reports confirm that, for large datasets, workload-dependent bugs are more numerous. Nevertheless, it is worth noting that the number of environment-dependent bugs presents an increasing trend from one version to another. This may be due to the addition of more software components that may cause the necessity for particular conditions of each of them in order to produce a failure.

More bug reports from different kinds of software (e.g., web servers, operating systems, apart from DBMS) should be analyzed in order to understand the trend of the two types of bugs. Also, how these and other classifiers can be improved in order to increase the quality of the prediction.

References

1. Carrozza G, Pietrantuono R, Russo S (2014) Defect analysis in mission-critical software systems: a detailed investigation. *J Softw Evol Process* 27(1):22, 49
2. Grady RB (1992) *Practical software metrics for project management and process improvement*. Prentice Hall, Englewood Cliffs
3. IEEE Computer Society IEEE Standard Classification for Software Anomalies, IEEE Std 1044–2009
4. Chillarege R, Bhandari IS, Chaar JK, Halliday MJ, Moebus DS, Ray BK, Wong M-Y (1992) Orthogonal defect classification—a concept for in-process measurements. *IEEE Trans Softw Eng* 18(11):943–956
5. Avizienis A, Laprie J-C, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans Dependable Secure Comput* 1(1):11–33
6. Gait J (1986) A probe effect in concurrent programs. *Softw Pract Exp* 16(3):225, 233
7. Gray J (1985) Why do computers stop and What can be done about it? Tandem Tech Report TR-85.7
8. Grottke M, Trivedi KS (2005) A classification of software faults. In: *Supplemental proceedings 16th IEEE international symposium on software reliability engineering (ISSRE)*, pp 4.19–4.20
9. Grottke M, Trivedi KS (2007) Fighting bugs: remove, retry, replicate, and rejuvenate. *Computer* 40(2):107–109
10. Grottke M, Nikora A, Trivedi KS (2010) An empirical investigation of fault types in space mission system software. In: *Proceedings IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp 447–456
11. Chillarege R (2011) Understanding Bohr-Mandel bugs through ODC triggers and a case study with empirical estimations of their field proportion. In: *Proceedings 3rd IEEE international workshop on software aging and rejuvenation (WoSAR)*, pp 7–13
12. Cotroneo D, Grottke M, Natella R, Pietrantuono R, Trivedi KS (2013) Fault triggers in open-source software: an experience report. In: *Proceedings 24th IEEE international symposium on software reliability engineering (ISSRE)*, pp 178–187
13. Lu S, Park S, Seo E, Zhou Y (2008) Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. *SIGARCH Comput Architect News* 36(1):329–339

14. Tan L, Liu C, Li Z, Wang X, Zhou Y, Zhai C (2014) Bug characteristics in open source software. *Empirical Softw Eng* 19(6):1665–1705
15. Carrozza G, Cotroneo D, Natella R, Pietrantuono R, Russo S (2013) Analysis and prediction of mandelbugs in an industrial software system. In: *Proceedings IEEE 6th international conference on software testing, verification and validation (ICST)*, pp 262–271
16. Cotroneo D, Natella R, Pietrantuono R (2013) Predicting aging-related bugs using software complexity metrics. *Perform Eval* 70(3):163–178
17. Bovenzi A, Cotroneo D, Pietrantuono R, Russo S (2011) Workload characterization for software aging analysis. In: *Proceedings 22nd IEEE international symposium on software reliability engineering (ISSRE)*, pp 240–249
18. Bovenzi A, Cotroneo D, Pietrantuono R, Russo S (2012) On the aging effects due to concurrency bugs: a case study on MySQL. In: *Proceedings 23rd IEEE international symposium on software reliability engineering (ISSRE)*, pp 211–220
19. Cotroneo D, Natella R, Pietrantuono R (2010) Is software aging related to software metrics? In: *Proceedings IEEE 2nd international workshop on software aging and rejuvenation (WoSAR)*, pp 1–6
20. Cotroneo D, Orlando S, Pietrantuono R, Russo S (2013) A measurement-based ageing analysis of the JVM. *Softw Test Verif Reliab* 23:199–239
21. Chandra S, Chen PM (2000) Whither generic recovery from application faults? A fault study using open-source software. In: *Proceedings international conference on dependable systems and networks (DSN)*, pp 97–106
22. Cavezza DG, Pietrantuono R, Russo S, Alonso J, Trivedi KS (2014) Reproducibility of environment-dependent software failures: an experience report. In: *Proceedings 25th IEEE international symposium on software reliability engineering (ISSRE)*, pp 267–276
23. Pietrantuono R, Russo S, Trivedi K (2015) Emulating environment-dependent software faults. In: *2015 IEEE/ACM 1st international workshop on in complex faults and failures in large software systems (COUFLESS)*, pp 34–40
24. Fonseca P, Cheng L, Singhal V, Rodrigues R (2010) A study of the internal and external effects of concurrency bugs. In: *Proceedings international conference on dependable systems and networks (DSN)*, pp 221–230
25. Nistor A, Jiang T, Tan L (2013) Discovering, reporting, and fixing performance bugs. In: *Proceedings 10th conference on mining software repositories (MSR)*, pp 237–246
26. Jin G, Song L, Shi X, Scherpelz J, Lu S (2012) Understanding and detecting real-world performance bugs. In: *Proceedings 33rd ACM SIGPLAN conference on programming languages design and implementation (PLDI)*, pp 77–88
27. Zaman S, Adams B, Hassan AE (2011) Security versus performance bugs: a case study on Firefox. In: *Proceedings 8th conference on mining software repositories (MSR)*, pp 93–102
28. Li Z, Tan L, Wang X, Lu S, Zhou Y, Zhai C (2006) Have things changed now?: an empirical study of bug characteristics in modern open source software. In: *Proceedings 1st workshop on architectural and system support for improving software dependability (ASID)*, pp 25–33
29. Lamkanfi A, Demeyer S, Soetens QD, Verdonck T (2011) Comparing mining algorithms for predicting the severity of a reported bug. In: *Proceedings 15th European conference on software maintenance and reengineering (CSMR)*, pp 249–258
30. Menzies T, Marcus A (2008) Automated severity assessment of software defect reports. In: *Proceedings IEEE international conference on software maintenance (ICSM)*, pp 346–355
31. Domingos P, Pazzani M (1997) On the optimality of the simple Bayesian classifier under zero-one loss. *Mach Learn* 29(23):103–130

Constraint-Based Virtualization of Industrial Networks

Waseem Mandarawi, Andreas Fischer, Amine Mohamed Houyou,
Hans-Peter Huth and Hermann de Meer

Abstract In modern industrial solutions, Ethernet-based communication networks have been replacing bus technologies. Ethernet is no longer found only in inter-controller or manufacturing execution systems, but has penetrated into the real-time sensitive automation process (i.e., close to the machines and sensors). Ethernet itself adds many advantages to industrial environments where digitalization also means more data-driven IT services interacting with the machines. However, in order to cater to the needs of both new and more automation-related communication, a better restructuring of the network and resources among multitenant systems needs to be carried out. Various Industrial Ethernet (IE) standards already allow some localized separation of application flows with the help of Quality of Service (QoS) mechanisms. These technologies also expect some planning or engineering of the system which takes place by estimating worst-case scenarios of possible traffic generated by all assumed applications. This approach, however, lacks the flexibility to add new services or to extend the system participants on the fly without a major redesign and reconfiguration of the whole network. Network virtualization and segmentation is used to satisfy these requirements of more support for dynamic scenarios, while keeping and protecting time-critical production traffic. Network virtualization allows slicing of the real physical network connecting a set of applications and end devices into logically separated portions or Slices. A set of resource demands and constraints

W. Mandarawi · A. Fischer · H. de Meer (✉)
Chair of Computer Networks and Computer Communications,
University of Passau, Innstraße 43, 94032 Passau, Germany
e-mail: Waseem.Mandarawi@uni-passau.de

A. Fischer
e-mail: Andreas.Fischer@uni-passau.de

H. de Meer
e-mail: demeer@uni-passau.de

A.M. Houyou · H.-P. Huth
Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany
e-mail: amine.houyou@siemens.com

H.-P. Huth
e-mail: hans-peter.huth@siemens.com

is defined on a Slice or Virtual Network level. Slice links are then mapped over physical paths starting from end devices through forwarding devices that can guarantee these demands and constraints. In this chapter, the modeling of virtual industrial network constraints is addressed with a focus on communication delay. For evaluation purposes, the modeled network and mapping criteria are implemented in the Virtual Network Embedding (VNE) traffic-engineering platform ALEVIN [1].

Keywords Field buses · Local area networks · Scheduling · Telecommunication network topology · Fieldbus technology · Industrial communication systems · Optimized datagram transfer · Real-time Ethernet system · Synchronous scheduling · Topology-based addressing · Auto configuration · Real-time Ethernet · Real-time communication · Synchronous scheduling

1 Introduction

In VNE, high level virtual networks requested by users are usually called Virtual Network requests (VNRs). In this chapter, the term ‘demands’ refers to the resource capacity needs and constraints defined for individual network entities in the VNR.

Resource-constrained Network Virtualization is a concept that has been adopted in industrial scenarios by Huth et al. in [17]. The network then has to define the end-nodes hosting a certain application or tenant, which has a certain set of requirements—called “demands”—that need to be treated separately or in an isolated manner. These requirements could include specific demands for computing resources such as CPU and network bandwidth resources, QoS demands such as communication latency, and security demands. Network virtualization allows for a demand-based efficient allocation of network resources to applications. It extends traditional server virtualization that shares a single physical host among multiple user virtual machines to allocate a complete Virtual Network of multiple virtual nodes and a set of links among them. This mathematical problem is known as the VNE problem [14]. In VNE, an algorithm tries to embed a set of virtual network requests (VNRs) by efficiently mapping each virtual node to a physical host, and each virtual link to one or more physical paths. Each virtual node and link carries a set of demands that should be satisfied in the mapping process. These demands can be classified in two main categories: consumable demands such as computing and network bandwidth resources, and constraint-based demands. The constraint-based demands can be either performance constraints such as communication latency, or security constraints such as the security level of a physical node.

Defining and modeling the specific requirements of industrial networks for the VNE problem is a challenging domain that has not been widely covered in research. However, many models have been presented to calculate performance bounds of small network samples using network calculus methods. Many VNE algorithms have been developed to consider different constraints in the optimized mapping but with a generic view that is not easily applicable to the real networks and does not consider the specific nature of industrial networks. These models also do not consider the real

properties of network entities and how the output of the mapping algorithms could be applied to configure the network devices.

In industrial networks, many applications share a large network of hundreds or thousands of nodes. These applications have different topologies and classes of traffic that might share the same network entities. The topology and communication requirements of an industrial environment depend on the applications running in this environment. All types of topologies could be used by industrial applications: line, star, ring, and mesh. All types of traffic might be found in an IE network: traditional user data, bandwidth-consuming video, and critical control data. The industrial network must be able to give priority to different types of traffic and deliver real-time network services: low latency and jitter and minimal packet loss when the network infrastructure is under load [10]. Most control operations in industrial applications can tolerate latencies of 10–50 ms [10]. The capability to share a network with other applications, yet maintain the priority of the critical traffic, is a key differentiating factor of IE [10]. The IE networks should also provide redundant paths, avoiding the situation of a single network entity failure taking down the entire network. Two network topologies most often used to achieve higher availability are ring and redundant star [10]. The introduction of network virtualization can relax these design criteria of the network topology, since these kinds of requirements could still be defined per application as a Virtual Network.

The network slicing approach adopted introduces a network controller, where the requested Virtual Networks and resource demands are submitted. The controller runs the algorithmic basis for deciding whether and how to embed the Virtual Network and has the potential to configure each real network device in the physical substrate. The details of the network slicing concept are given in Sect. 2.

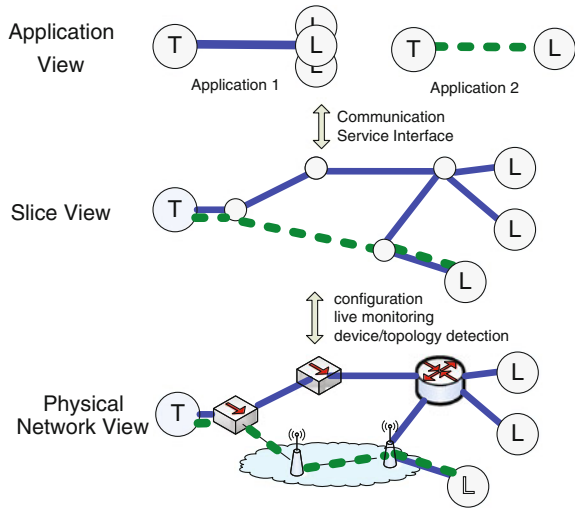
In Sect. 3, the model used for the application of VNE to industrial networks using Network Slices is covered. A general overview of VNE is presented in Sect. 3. Section 4 classifies the different types of communication constraints and describes some specific constraints of industrial environments and how these constraints are modeled by VNE research. A simple approach for modeling delay is developed and presented by the authors in this Section. Section 5 introduces the ALEVIN VNE platform and describes how the delay model is implemented. Section 6 introduces the evaluation approaches of VNE algorithms. Section 7 summarizes this chapter.

2 Network Slices in Industrial Networks

2.1 Overall System Architecture

To model the requirements of multiple applications that belong to the same service class in an Industrial Network, a network abstraction layer is introduced, the ‘Slice layer’ (Huth et al. [17]). The Slice layer (see Fig. 1) provides a graph view of the network. Applications can only see edge nodes and intelligent traffic-engineering algo-

Fig. 1 Layers of network abstraction (Huth et al. [17])



rithms can operate on the graph without knowing details of the underlying substrate implementation. Important architectural components are clean interfaces, meaning they must provide clearly defined responsibilities and enforce a ‘separation of concerns’ (Laplante et al. [24]).

Applications (including planning and management tools) use a ‘Communication Service Interface’ (CSI) for setting up and using the network. The Slice view is provided and controlled from a central component, the Slice Manager (SMGR). The SMGR is responsible for a network, or a part of a network. If a Network Virtualization (Slice) must be constructed, the SMGR receives the necessary specifications from the application via the CSI. The SMGR then calculates an optimal routing subgraph and finally commissions the required rules to the physical network using a ‘driver layer’. The driver layer has knowledge of the concrete methods for accessing the devices and this layer also provides device abstraction for the Slice view in the SMGR. The driver layer forms the base for including a multitude of different standards or vendor-specific device interfaces. It is an important component for making the Slice layer agnostic to the underlying device and protocol heterogeneity.

Unlike traditional approaches, this layered approach does not mandate specific interfaces. The SMGR can handle all of the interfaces simultaneously by means of driver ‘plug-ins’. Because of this, the SMGR constitutes an upgrade-friendly, unified solution, providing the means to integrate both legacy equipment and specialized industrial elements or upcoming programmable/software defined networks under one control platform. Devices, whether they are end devices or network elements, must be able to perform routing or forwarding according to the policies defined by the SMGR. In this architecture, the needed functionality along with signaling interface to the SMGR are bundled in a logical function called a Slice Enforcement Point (SEP), which is shown in Fig. 2. The SEP can be implemented on that device or use other means (e.g., Simple Network Management Protocol (SNMP)) to remotely perform its tasks. End devices that are not under the control of a SEP, hereafter called ‘legacy devices’, can be attached to an adjacent network element with a SEP (‘edge

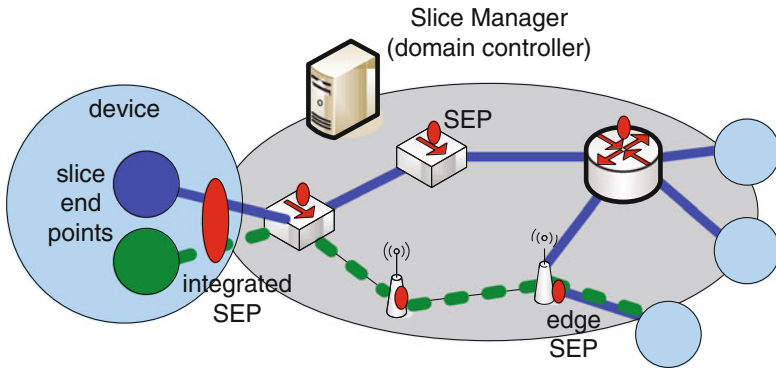


Fig. 2 Architecture birds view (Huth et al. [17])

SEP'as in Fig. 2). This edge SEP controls the corresponding network interfaces and enforces the needed policies for the legacy devices. This solution is similar to a port-based VLAN, which indicates a migration path from today's Ethernet technology toward our Slice solution. Devices that are specifically dedicated to support the new approach will have their own local SEP ('integrated SEP ', Fig. 2) so that Slices can reside in that device.

2.2 The Slice Manager

The SMGR controls all devices it is responsible for and provides the interface for managing Slices. Figure 3 shows the internal functional architecture of the SMGR.

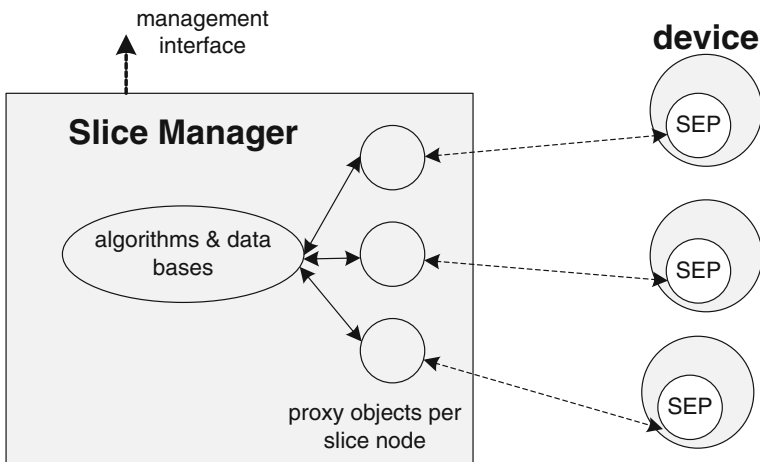


Fig. 3 Simplified SMGR architecture (Huth et al. [17])

The SMGR should expose both a management interface and an interface for accessing devices in the network. The management interface allows controlling (i.e., establishing, tearing down, and changing) Slices without intervention by an application. The management interface can also be accessed by planning tools or by an operator. The interface to the physical devices follows an object-oriented paradigm with one proxy object per device managed by this SMGR instance. These proxy objects also provide a standardized abstract view of a device and they contain drivers for accessing the device. For example, an industrial Ethernet device and a legacy-Ethernet device have different proxy implementations and the algorithms in the SMGR only see different device properties but access to the devices (viz. the interface) is of the same type for both.

In order to manage and optimize Slices, the SMGR must not only know all devices and their capabilities, but also the topology of the network. One way to accomplish this is to collect all neighborhood information from the devices and infer the network graph from the information collected. As SNMP and LLDP are already frequently used in industrial devices, this info is usually present even in today's devices.

Notice that Fig. 3 does not show an application interface. The device interface (which uses SEP signaling) could be re-used as an application signaling channel for several reasons: (a) it is easy for applications to discover a local SEP, so there is no need for another service discovery, (b) access control becomes easier as it can already be performed in the SEP, and (c) there is no need for an additional asynchronous and multi-client interface, a fact that simplifies implementation of the SMGR.

2.3 The Slice Enforcement Point

As explained before, the SEP is a functionality bound to a device. It is responsible for signaling to the SMGR or the applications, and it controls the network interfaces of that device. For example, the SEP must be able to manipulate forwarding or routing tables and to set QoS rules. The actual implementation can reuse existing control interfaces such as SNMP (Case et al. [7] and Yang et al. [31]). However, a small dedicated agent forming the SEP is beneficial because it can be tailored for that purpose and it may add local intelligence that, e.g., enables quick recovery in failure cases without contacting the SMGR. Notice that due to the object-oriented architecture of the SMGR, a mix of different SEP implementations is of course possible.

2.4 The Applications Interface to Communication Services

In order to use Slices, applications need a communication interface (User Plane) and signaling means (Control Plane) for attaching to a Slice and for specifying properties such as bandwidth constraints or QoS. For the user plane, a virtual layer-2 interface

could be used, similar to solutions found in server virtualization environments. The signaling can use the next SEP as an entry point, which is either located on a device or on an adjacent edge device (switch or router).

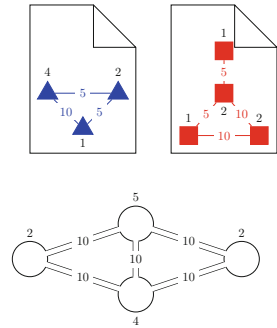
Specification of a Slice includes at least a specification of the required QoS, but in order to enable online traffic-engineering, a specification of the traffic is also beneficial. Slices could be used for aggregate flows of an application and not for single flows, although the latter is conceptually possible. Pure QoS, industrial applications may also have tight reliability requirements. Furthermore, some of them will have a high importance (i.e., a safety application). Thus, a notion of resilience and importance to the specification of traffic requirements should be added. A traffic-engineering algorithm can use this information to, e.g., establish redundant paths or to resolve resource shortages.

2.5 Use Cases

The following sample use cases illustrate how the Slice system can be used. First, consider a data acquisition application accessing many sensors and a server collecting the sensor data. In the planning step, a traffic matrix is calculated and a Slice is defined. The Slice definition can be forwarded to the SMGR. The SMGR then calculates an optimal path for the Slice and commissions it to the network devices. After that, the end devices (sensors and server) simply attach to the predefined Slice. In the ‘attach’ process, a Virtual Network interface is created by the device SEP and the interface is bound to the Slice in question. A second example is an automation application that uses a controller unit connected to actuators and reading sensors. The application requires hard real-time communication and is programmed by a planning tool. This tool can estimate and generate the traffic matrix and also the Slice specification. The latter is then pushed into the SMGR. Unlike today, the planning tool does not need full knowledge of the topology and capabilities of the network. The SMGR in turn, knowing the properties of the network, can construct an optimal Slice, eventually reusing specific means from one of the many IE standards [21]

While the first two examples assume pre-planning, dynamic Slice setup in the running system is also feasible. For instance, consider that remote support needs to access a robot for maintenance. In this case, the network operator may install means for granting access to the remote service over the firewall. In order to facilitate this, the operator installs a Slice on the fly. This Slice might support best-effort communication and an upper bandwidth limit. Additionally, this Slice only exposes the devices needed for the remote service. These sample use cases also illustrate that the ‘normal’ use of the Slice system is to construct semi-static or longer living Slices for aggregates (i.e., flows belonging to one application). In many cases, the Slices will be instantiated by means of a configuration/management interface rather than application signaling.

Fig. 4 A VNE instance



3 Slice Embedding

With Network Virtualization becoming more and more widespread, the resource assignment problem (VNE) has gained significant attention in the last years (Fischer et al. [14]). The premise of this problem is that a given number of VNRs are to be realized on a single physical (or ‘substrate’) network. Slice embedding is a particular form of VNE. The term Slice introduced in the previous section denotes a Virtual Network plus allocated resources and policy rules associated with that Virtual Network. If physical network resources are scarce, the optimal assignment of resources to the demands of a Slice becomes an interesting problem. Indeed, the general VNE problem has been shown to be NP-hard (Zhu et al. [32]). Typically, node and link resources are considered; CPU and bandwidth are usually taken as examples. For the VNE problem, it is assumed that resources and demands match each other. A demand imposed by a Slice requires a corresponding resource provided by the substrate network. Moreover, it is assumed that demands and resources remain static over the course of time. This is, of course, a significant simplification from reality, where particular demands may vary wildly (e.g., via traffic spikes). Placement of virtual nodes is generally assumed to be unrestricted, apart from the generic constraint that a substrate node must have sufficient available resources to host a particular virtual node. Those nodes are interconnected by virtual links. Virtual links differ from physical links in that they can be stretched over a path in the substrate network to allow two virtual nodes that are adjacent in the VNR to be mapped to two substrate nodes that are not directly connected to each other. Figure 4 depicts an example instance of the VNE problem. Here, a substrate network with four nodes should be used to host two VNRs with three and four nodes, respectively. Node and link resources/demands are indicated with vertex and edge weight. Though a solution exists in the example shown in the figure, it may not be immediately obvious, even though the number of resources involved is still very small. It should be clear that larger problems with realistic network sizes can be very challenging to solve.

Formally, the problem can be described as a graph problem. Given a substrate network represented by a graph $SN = (N, L)$ with resource capacity functions for nodes and links, $cap_n : N \rightarrow \mathbb{R}$ and $cap_l : L \rightarrow \mathbb{R}$, along with a number of VNRs $VNR^i = (N^i, L^i)$ with respective demand functions for nodes and links, $dem_n^i :$

$N^i \rightarrow \mathbb{R}$ and $dem_l^i : L^i \rightarrow \mathbb{R}$, is it possible to assign substrate resources such that all VNRs can be satisfied? Formulated as an optimization problem: What is the minimum amount of substrate resources that must be spent to realize the given set of VNRs? The NP-hardness of this problem extends in two dimensions. Focusing on nodes, the problem is a variation of the bin-packing problem, where substrate nodes are bins and virtual nodes are weights to pack into the bins. Focusing on links instead, the problem is a variation of the unsplittable flow problem, provided that virtual links are only allowed to use a single path in the substrate network. It is the combination of those two problems that makes VNE challenging and interesting at the same time.

4 Modeling of Communication Constraints

4.1 Performance Constraints

4.1.1 Delay

The end-to-end communication delay between end devices is the most critical performance constraint in industrial environments. In real-time systems, both maximum and minimum delays are critical. This adds an additional performance metric, jitter. The jitter of the delay models the range in which the delay might fluctuate. Different types of delays might be caused by different network entities. Since IE is the modern technology in industrial environments, the main focus of this chapter is on the latencies in a switched Ethernet network, and how to calculate the cumulative latency over a network path on which a Slice link is mapped. However, latencies in other networks such as wireless networks are considered by other researches. According to [30], switched Ethernet networks may have the following sources of delay for an Ethernet frame in practice:

1. Store and forward (transmission delay): the switch stores the received data in memory until the entire frame is received. The switch then transmits the data frame through the appropriate out port: $LSF = FS/BR$ (FS is the frame size in bits, and BR is the bit rate in bits/s). For example, for the maximum size of an Ethernet frame (1500 bytes) at 100 Mbps bit rate, this latency is 120 μ s.
2. Switch fabric processing (processing delay): the functions of the switch such as maintaining the MAC address table and VLAN. In modern industrial switches, this delay is about 5 μ s.
3. Wireline transmission (propagation delay): about 2/3 of the speed of light (3×10^8 m/s) in fiber optics. For example, the latency for a 100 km link is 500 μ s.
4. Frame queuing (queuing delay): the delay caused by other frames still waiting for transmission in the switch.

Queuing delay is non-deterministic since it depends on the exact traffic patterns that traverse the switch and the scheduling policies. The traffic patterns define the maximum frame size, class of service, and time distribution and rate of frames. However, a maximum queuing delay in an Ethernet switch can be estimated in a simple non-preemptive and non-prioritized queuing. For example, in an N port switch, the worst case in an egress port happens when a maximum size frame from each other ingress port is scheduled for transmission through this egress port at the same time. In this case, $N-1$ frames would be in the queue to be transmitted [30]. The maximum queuing delay is then $(FS(N-1)/BR)$. For example, in the case of a 16-port switch, and for the maximum size Ethernet frame (1500 bytes) and 100 Mbps bit rate, the latency is 1.8 ms [30].

Modeling delay requirements in VNE is covered by some researchers. However, the general approach is to assign delay demands to virtual links and assign certain delay values as a cost factor to links. The delay is then considered in the optimal embedding algorithm. In [18], Ivaturi et al. introduced a mixed integer programming formulation for delay aware VNE (VHub-Delay). The original VHub algorithm tries to reduce the inner-nodal distance between virtual nodes mapped on the substrate network (SN). VHub-Delay is an improvement of VHub. During the mapping, delay and bandwidth capacity of edges are considered. The evaluation of VHub-Delay shows that VHub-Delay achieves a lower Delay Ratio over paths. The metric Delay Ratio is the ratio of the delay to the length of a certain virtual link path.

In [29], Shengquan et al. introduced a delay aware algorithm based on a multi-agent approach. The multi-agent architecture partitions a VNR into k parts and computes a Node Rank for each part. The algorithm VNE-DC (VNE with delay constraints) tries to map virtual nodes onto physical nodes, considering the delay constraint and trying to minimize the bandwidth costs. After a successful mapping of nodes, links are determined by the k -shortest path algorithm. The k -shortest path algorithm chooses latency satisfying paths for determining links. The authors consider two types of delays, the propagation and queuing delay. The authors compared VNE-DC to the similar Hybrid-VNE algorithm that does not consider the delay. The results show that VNE-DC has a small advantage in acceptance ratio (of Virtual Networks) and bandwidth cost.

In [3], Basta et al. present a solution for QoS-aware mapping with minimum costs. The authors describe the effect of service differentiation on Virtual Networks. Service differentiation can be performed either by the Physical Infrastructure Provider (PIP) or the Virtual Network Operator (VNO). Three models are introduced. In the first model, the VNO is responsible for guaranteeing the service and providing resiliency in its own domain. In the second model, the VNO guarantees the service quality in its domain and the PIP guarantees the resiliency of the virtual links. In the last model, the PIP is responsible for guaranteeing the service and resiliency. All models consider three QoS classes (Gold, Silver, and Bronze) with specific values for maximum delay. The 'Gold' class is defined for delay sensitive services such as gaming applications and guarantees a maximum delay of 20 ms. The 'Silver' class is defined for less time sensitive applications such as VoIP and guarantees a maximum delay of 70 ms. The 'Bronze' class guarantees a maximum delay of 170 ms, and can be used for

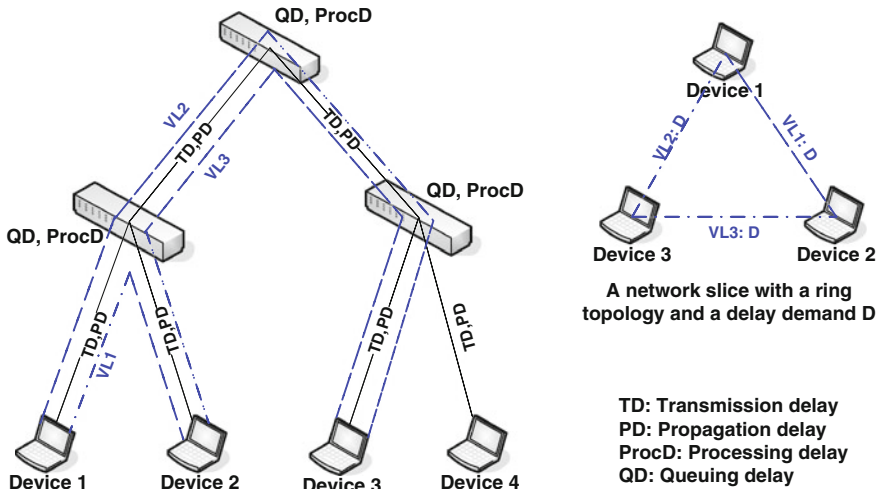


Fig. 5 Assigning different delay types to network entities

non-critical applications such as web-browsing. The paths are categorized in these classes. To verify if the path satisfies the delay demand, the path delay is calculated as the sum of the end-to-end delays of each link. The evaluation shows that handling QoS and resiliency by the PIP and not by the VNO improves the overall network utilization.

Beck et al. [5] present a nonlinear delay model, based on queuing theory. Delay is modeled, for substrate links, as a function. Four different delay types are considered: processing, transmission, propagation, and queuing delay. As an optimization strategy for VNE, the increased queuing delay that results from intense traffic is considered in the model. Considering traffic intensity during the mapping stage, leads to a different result and better resource utilization than traditional VNE approaches. The total delay of a link is the sum of all delay types assigned to this link. Queuing delay is calculated through a specific formula that considers the average number of packets waiting in the queue of a link and the queue size of a system.

In the model being developed by the authors, the aforementioned types of delays for both nodes and links are modeled in the ALEVIN VNE platform. In this model, the propagation and transmission delays are link properties since they depend on the transmission medium and the bit rate. The processing and queuing delay are node properties. Figure 5 reflects this model. To calculate the end-to-end delay between two nodes, the different types of delays along the physical path between the two nodes are accumulated. However, only intermediate forwarding nodes in the path are considered in the delay calculation. The processing and queuing delay in the end application devices are neglected.

4.1.2 Reliability and Availability

Links and nodes may exhibit different quality expressed in terms of packet loss probability or link availability (e.g., mean time between failures). Traffic flows on the other hand may wish to have a certain availability and a maximum loss ratio. The layout of a Slice should take this into consideration by choosing appropriate paths or even by introducing redundancy. Many researches address the resilience-aware VNE (RVNE) by developing algorithms to handle single node or link failures. According to Markopoulou et al. [26], 20% of all failures happen during a period of scheduled maintenance activity. Of the unplanned failures, almost 30% are shared by multiple links and are most likely due to router-related and optical equipment-related problems, while 70% affect a single link at a time. Table 1 compares some approaches for resilient VNE according to the following criteria:

- Failure model: Types of failures that the approach provides resilience for.
- Strategy: Resilience mechanisms such as capacity splitting, protection cycles, and backup paths.
- Stages: Embedding stages.
- Solution: Mathematical model used by the embedding algorithm.
- Backup: Specifies whether this approach depends on reserving backup resources.

4.2 Security Constraints

Security constraints are notably different from either resource constraints such as CPU and bandwidth, or performance constraints such as delay and reliability, in that they are typically qualitative in nature. A virtual node might require a trusted hypervisor environment. A virtual link might require encryption. Such requirements are not optional, but have to be realized by specific hardware that is capable of providing the desired level of security. The demands posed by the virtual nodes and links refer to properties, rather than resources, of the physical environment. A physical node might be equipped with a trusted hypervisor. A physical link might use encryption for data transfer.

For the VNE problem, these constraints introduce qualitative differences between nodes and links in the substrate network. Nodes that have a trusted hypervisor will be differentiated from nodes that lack this particular capability. In the following, it will be explained how these constraints can be modeled appropriately for the VNE problem.

4.2.1 Node Capabilities

Node capabilities refer to particular properties of substrate nodes that can be demanded by virtual nodes. Two particular types of node capabilities will be dis-

Table 1 Comparing some approaches to RVNE

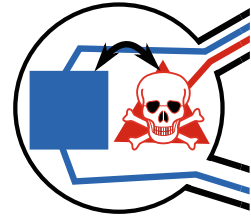
Approach	Failure model	Strategy	Stages	Solution	Backup
Oliveira et al. [27]	Single physical link failure	Multipath capacity splitting, capacity recovery using remaining paths	Nodes, links, capacity	Meta-heuristic	No
Jarray et al. [20]	Single physical link failure	Protection-cycle	Nodes, links, protection	Column generation, Hongbo graph cycles algorithm	Minimized
Jarray et al. [19]	Single physical node failure	Protection-cycle	N/A	Column generation	Minimized
Chen et al. [8]	Single physical entity failure	Load balancing, reconfiguration of backup resources	N/A	Linear programming heuristic	Yes
Basta et al. [3]	Single physical entity failure	QoS classes with backup baths, minimum cost	N/A	Mixed Integer Linear Problems (MILPs)	Yes

cussed here as examples: Virtual Machine Introspection (VMI) and Trusted Hypervisors. VMI is a technology that allows the hypervisor to gain outside information about a virtual machine. For security, this can be used to monitor virtual machines and react to potentially malicious behavior (Garfinkel et al. [15]).

A Trusted Hypervisor represents the extension of the Trusted Platform Module (TPM) principle to a virtualization environment. A virtual node with a high security requirement may demand to be hosted only on hypervisors providing a trusted computing environment to protect against adversaries that try to attack the hypervisor directly.

Both types of node capabilities can be expressed as a property that is attached to the substrate node and demanded by the virtual node. Unlike resources, these properties are not spent when a virtual node is mapped to a substrate node. They only limit the number of potential mapping candidates for each virtual node. This means that a virtual node with the ‘Trusted Hypervisor’ constraint attached may only be mapped to substrate nodes that provide the ‘Trusted Hypervisor’ property.

Node capabilities can be mixed and matched. In conventional VNE scenarios, there is only a single node constraint, namely CPU, that is present on each node (either as a demand or as a resource). In our scenario, we can have some nodes

Fig. 6 A cross-VM attack

providing (or demanding) a particular property, or a set of properties. Thus, in our example there will be substrate nodes that have no particular properties, nodes that provide VMI, nodes that provide a Trusted Hypervisor, and nodes that provide both. The same applies for demands of virtual nodes. A mandatory property is the limitation of machine resources (disk/memory space, CPU load).

4.2.2 Isolation

Virtualization can be used for isolation purposes to support security of virtual services. However, it will not solve all security problems (Cohen et al. [11]). Threats that are specific to virtualized environments (e.g., cross-Virtual Machine (VM) attacks, cf. Ristenpart et al. [28]) have already been discussed in the literature. Two examples of isolation will be discussed here: Isolation between different VNRs and isolation within a single VNR. The first one can be used to avoid cross-VM attacks. In particular, when virtual nodes of different customers are co-hosted on the same physical node, attacks may be possible (cf. Fig. 6). A malicious VM might try to gain information about a co-hosted VM, e.g., via side-channel attacks. To avoid these kinds of attacks, two VMs that are potential adversaries should not be co-hosted on the same hardware. In a scenario where multiple heterogeneous Virtual Networks are supposed to co-exist on the same hardware, this means that security-critical virtual nodes should not be co-hosted along with nodes that have low security. VNE must take such additional constraints into account.

Another requirement is isolation within a topology. In the physical world, firewalls are used to segment parts of the network and control network traffic closely. This concept can be transferred into a virtualized environment. A hardware firewall that is present in the substrate network can be used by the Virtual Networks to segment and isolate parts of individual Virtual Networks with high security demands from less secure parts. In this case, virtual nodes have to specify that they are part of a network segment that should be protected by a firewall. Special substrate nodes have to be tagged as possessing firewall capabilities. A VNE algorithm then has to identify segments in both the virtual and substrate networks, making sure that any pair of virtual nodes is connected via a firewall in the substrate network.

5 Modeling Industrial Constraints in the ALEVIN VNE Platform

ALEVIN is a Java-based open-source virtual embedding platform developed in the VNREAL project [4, 13]. ALEVIN enables the creation of a substrate network and a set of Virtual Networks using three different methods: manually via a GUI, imported from a data file, or a randomly generated scenario with predefined ranges for the network sizes. The resources are modeled in ALEVIN as properties of substrate nodes and links. The demands of Virtual Networks are also modeled as properties of virtual nodes and links. ALEVIN provides an abstract architecture that makes it easy for the developer to add new pairs of resources/demands. When the mapping algorithm identifies the substrate nodes and links on which the Virtual Network will be mapped, each resource in each substrate network entity is occupied with the capacity demanded by the mapped Virtual Network entity. ALEVIN also provides the infrastructure to develop and use different VNE algorithms. A set of algorithms are already implemented in ALEVIN such as subgraph isomorphism (Lischka et al. [25]) and node ranking (Cheng et al. [9]). The subgraph isomorphism algorithm is based on subgraph isomorphism detection. It maps nodes and links during the same stage. The algorithm uses Dijkstra's algorithm to find the shortest path between two substrate nodes in which each link satisfies all the demands of the mapped virtual link. The node ranking algorithm is a two-stage algorithm. In the node mapping stage, a rank is calculated for each substrate node based on the CPU capacity and the bandwidth of outgoing links. The rank is also calculated for virtual nodes. The virtual node with the greatest node rank is mapped on the substrate node with greatest node rank if the substrate node satisfies its demands. The link mapping stage calculates k-shortest paths for each virtual link using the algorithm in by Eppstein et al. [12]. Each path is then verified for all the demands of the virtual link until a satisfying path is found.

The Slices are modeled in ALEVIN as Virtual Networks that specify virtual links among end devices. Each Slice defines certain topology and demands. ALEVIN converts the Slice to a VNR with the required topology and copies the demands to each Slice link to the Virtual Network. The node mapping stage is not actually performed since the virtual nodes are identical to the substrate nodes. The network path constraints are modeled in ALEVIN by introducing a new type of resource/demand pairs for which a different method is used in the mapping procedure. Instead of occupying the capacity of the resources, these constraints are verified for candidate paths. The discussed delay model has been implemented in ALEVIN using delay resources that are assigned to physical nodes and links according to the specific type of the delay and to delay demands assigned to virtual links. ALEVIN is also developed to support the verification of path constraints during the mapping of virtual links. A generic architecture is built to intercept the link mapping stage of any embedding algorithm. In the new architecture, each candidate path found by the link mapping algorithm is verified for each path constraint of the mapped virtual link. This verification procedure is already used in ALEVIN to check the residual resource capacities of all

network entities along the path and decide if this path can map the virtual link. The new architecture adds the verification of path constraints to the available verification procedure. For example, the delay constraint of a virtual link is compared to the accumulated delays (all delay types) and for all network entities along the candidate path. However, considering these constraints in the optimized mapping algorithm has not yet been addressed. This will add multiple dimensions to the problem and will greatly increase its complexity and execution time for large scenarios.

6 Evaluation Scenarios

In this section, the general evaluation approaches of VNE algorithms are discussed. Possible extensions of these approaches to meet the requirements of industrial networks are proposed. To evaluate VNE algorithms in industrial networks, four different issues should be considered:

- The topologies of the substrate and Virtual Networks: Traditional VNE approaches use random topologies with varying network sizes and connectivity. Evaluation of industrial networks should be able to consider the specific topologies used in these networks such as the ring and redundant tree. The test scenarios should be able to create specific topologies and support the creation of Slices with different topologies.
- The classes of Virtual Networks: Evaluation of VNE in industrial networks should be able to create different application classes with different requirements. For example, different classes of delay demands should be included in a test scenario and a delay class should be assigned to each Slice to simulate the nature of industrial networks where different applications with specific delay requirements share the network. The distribution of these classes in the Virtual Network could then be varied to evaluate the effect of this variation on the evaluation metrics.
- The parameters of the substrate network resources and Virtual Networks demands/constraints: Defining the parameters of the resources/constraints used for evaluating VNE algorithms in industrial networks requires a deep analysis of these networks. To define the delay demand ranges for the test scenarios, the latency requirements of the industrial automation applications should be considered. According to Kerschbaum et al. [22], each automation application requires a different QoS in terms of communication latency. These requirements are divided into three classes: 100, 10, and 1 ms.
- The objectives of the algorithm and the evaluation metrics. Those issues will be discussed in detail below.

Fischer et al. [14] described the main embedding objectives and evaluation metrics that have been defined in VNE approaches. Metrics are used to compare the quality of different VNE approaches. The main objectives and the main related metrics in VNE approaches are described below.

- QoS-compliant embeddings:

- Path length: Average number of substrate links that map a virtual link.
- Stress level: Number of virtual entities mapped over a substrate entity.
- Throughput: Real bandwidth between virtual nodes.
- Delay: Communication latency between virtual nodes.
- Jitter: Variance in the communication latency between virtual nodes.
- Maximizing the economic benefit of VNRs:
 - Cost: Sum of all substrate resources utilized for embedding the VNRs.
 - Revenue: Sum of all resource requirements of VNRs.
 - Cost/Revenue: Ratio between reserved substrate resources and virtual resources provided.
 - Acceptance ratio: Number of successfully mapped VNRs.
 - Active substrate nodes: Number of substrate nodes that have to be running to realize the embedding. This metric is related to operation costs such as energy usage.
- Survivable embeddings: Creating backup resources in the substrate network. Backup nodes/links can be set up either for the whole Virtual Network or just for certain heavily-loaded nodes/links that have high failure probability. Recovery from failures should not interrupt the Virtual Network.
 - Number of backups: Number of available backup resources.
 - Path redundancy: Level to which the paths in multipath embeddings are disjoint.
 - Redundancy cost: Number of additional resources used to provide resiliency.
 - Recovery blocking probability: Ratio of unrecoverable failures vs. all possible failures.
 - Number of migrations : Number of virtual nodes that have to be migrated in case of failure.
- Another generic metric that is used to evaluate the VNE algorithms is the algorithm runtime for different sizes of both the substrate and Virtual Networks.

These general evaluation objectives and metrics should be adapted to evaluate the VNE algorithms in industrial networks. These networks have different requirements from traditional networks since they host different application classes in one substrate network with different requirements such as delay guarantees. The following evaluation metrics are examples of new metrics that might be used to evaluate VNE algorithms in industrial networks:

- The acceptance ratio of VNRs for different ranges of the demanded delays. In this metric, only the delay constraints should be defined in the mapping scenario to make sure that all rejections of Virtual Networks are due to the strict delay constraints.

- Comparing the execution time of the algorithms when they do or do not consider the delay constraints with an increased network size. The evaluation parameters should guarantee a 100 % acceptance ratio to check the performance overhead of the algorithms without resulting in rejection.
- Average difference between the demanded delay of a virtual link and the actual delay of the path mapped. The average should be calculated for all the paths of all accepted (successfully mapped) Virtual Networks.

7 Summary

Industrial networks have special requirements that cannot be easily planned. Network Virtualization can provide this planning by efficiently allocating network resources to different industrial applications considering the specific constraints for each application. To apply Network Virtualization in Industrial Networks, the concept of ‘Slice’ is introduced. The Slice defines a Virtual Network that specifies the application resource requirements and constraints and the virtual links among end devices that are used by the application. The VNE algorithms can then be used to efficiently map this Slice on the Industrial Network and satisfy all the specified constraints. These algorithms can provide online and automated reconfiguration of the Industrial Network to fit the changing application requirements.

References

1. Alevin2: a tool for the evaluation of algorithms for embedding virtual networks. <http://sourceforge.net/p/alevin/wiki/home/>
2. Bassiri B, Heydari SS (2009) Network survivability in large-scale regional failure scenarios. In: Proceedings of the 2nd canadian conference on computer science and software engineering, C3S2E '09, pp 8387, New York, NY, USA. ACM
3. Basta A, Barla IB, Hoffmann M, Carle G (2013) QoS-aware optimal resilient virtual networks. In: IEEE international conference on communications (ICC), pp 2251–2255, 9–13 June 2013
4. Beck MT, Linnhoff-Popien C, Fischer A, Kokot F, de Meer H (2014) A simulation framework for virtual network embedding algorithms. In: 2014 16th international telecommunications network strategy and planning symposium (Networks), pp 1–6, 17–19 Sept 2014
5. Beck MT, Linnhoff-Popien C (2014) On delay-aware embedding of virtual networks. In: The sixth international conference on advances in future internet, AFIN
6. Bui M, Jaumard B, Harter IBB, Develder C (2014) Scalable algorithms for QoS-aware virtual network mapping for cloud services. In: 2014 international conference on optical network design and modeling, pp 269–274, 19–22 May 2014
7. Case J (2002) Introduction and applicability statements for internet standard management framework. In: RFC 3410, IETF, Dec 2002
8. Chen Q, Wan Y, Qiu X, Li W, Xiao A (2014) A survivable virtual network embedding scheme based on load balancing and reconfiguration. In: Network operations and management symposium (NOMS), 2014 IEEE, p 17, May 2014

9. Cheng Xiang Su, Sen Zhang Zhongbao, Hanchi Wang, Fangchun Yang, Yan Luo, Jie Wang (2011) Virtual network embedding through topology-aware node ranking. *SIGCOMM Comput Commun Rev* 41(2):38–47
10. Cisco (2010) Industrial ethernet: a control engineers guide. In: Cisco white paper
11. Cohen F (2010) The virtualization solution. *IEEE Secur Priv IEEE Comput Soc* 8:60–63
12. David Eppstein (1998) Finding the k shortest paths. *SIAM J Comput* 28(2):652–673
13. Fischer A, Botero JF, Duelli M, Schlosser D, Hesselbach X, De Meer H, Margaria T, Padberg J, Taentzer G, Hellbrck H, Luttenberger N, Turau V (eds) (2011) ALEVIN—a Framework to develop, compare, and analyze virtual network embedding algorithms. In: Electronic communications of the EASST, proceedings of the workshop on challenges and solutions for network virtualization (NV2011), EASST, 37, pp 1–12
14. Fischer A, Botero JF, Beck MT, De Meer H, Hesselbach X (2013) Virtual network embedding: a survey. *IEEE Commun Surv Tutorials* 15:1888–1906
15. Garfinkel T, Rosenblum M (2003) A virtual machine introspection based architecture for intrusion detection. In: Proceedings of network and distributed systems security symposium, pp 191–206
16. Harrison C, Cook D, McGraw R, Hamilton JA (2012) Constructing a cloud-based IDS by merging VMI with FMA. In: IEEE 11th international conference on trust, security and privacy in computing and communications (TrustCom), pp 163–169, 25–27 June 2012
17. Huth HP, Houyou AM (2013) Resource-aware virtualization for industrial networks. In: 4th international conference on data communication networking (DCNET 2013), Reykjavik, Iceland, July 2013
18. Ivaturi K, Wolf T (2014) Mapping of delay-sensitive virtual networks. In: 2014 international conference on computing, networking and communications (ICNC), pp 341–347, 3–6 Feb 2014
19. Jarray A, Yihong S, Karmouch A (2013) p-Cycle-based node failure protection for survivable virtual network embedding. In: IFIP networking conference, pp 1–9, 22–24 May 2013
20. Jarray A, Yihong S, Karmouch A (2013) Resilient virtual network embedding. In: ICC'13, pp 3461–3465
21. Jasperneite J, Imtiaz J, Schumacher M, Weber K (2009) A proposal for a generic real-time ethernet system. *IEEE Trans Industr Inform* 5(2):75–85
22. Kerschbaum S, Hielscher KS, Klehmet U, German R, Fischbach K, Krieger U (eds) (2014) A framework for establishing performance guarantees in industrial automation networks. In: Measurement, modelling, and evaluation of computing systems and dependability and fault tolerance, vol 8376. Springer International Publishing, pp 177–191
23. Khan A, Herker S, An X (2013) Survey on survivable virtual network embedding problem and solutions. In: ICNS 2013, the ninth international conference on networking and services, pp 99–104
24. Laplante P (2007) What every engineer should know about software engineering. In: CRC Press. ISBN 0849372283
25. Lischka J, Karl H (2009) A virtual network mapping algorithm based on subgraph isomorphism detection. In: Proceedings of the 1st acm workshop on virtualized infrastructure systems and architectures, VISA'09, pp 81–88, New York, NY, USA. ACM
26. Markopoulou A, Iannaccone G, Bhattacharyya S, Chuah C-N, Ganjali Y, Diot C (2008) Characterization of failures in an operational ip backbone network. In: *IEEE/ACM transactions on networking* 16(4):749–762
27. Oliveira RR, Marcon DS, Bays LR, Neves MC, Buriol LS, Gasparly LP, Barcellos MP (2013) No more backups: toward efficient embedding of survivable virtual networks. In: ICC'13, pp 2128–2132
28. Ristenpart T, Tromer E, Shacham H, Savage S (2009) Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: CCS '09: proceedings of the 16th ACM conference on computer and communications security, ACM, pp 199–212
29. Shengquan L, Chunming W, Min Z, Ming J (2013) An efficient virtual network embedding algorithm with delay constraints. In: 2013 16th international symposium on wireless personal multimedia communications (WPMC), pp 1–6, June 2013

30. Siemens Canada Limited: Application note 8: Latency on a switched Ethernet network. 2014.03.21. <https://w3.siemens.com/mcms/industrial-communication/en/ruggedcommunication/Documents/AN8.pdf>
31. Yang L, Dantu R, Anderson T, Gopal R (2004) Forwarding and control element separation (ForCES) framework. In: RFC 3746, IETF, April 2004
32. Zhu Y, Ammar M (2006) Algorithms for assigning substrate network resources to virtual network components. IN: Proceedings of FOCOM 2006, 25th IEEE international conference on computer communications, pp 1–12

Component-Oriented Reliability Assessment Approach Based on Decision-Making Frameworks for Open Source Software

Shigeru Yamada and Yoshinobu Tamura

Abstract At present, the open source software (OSS) development paradigm is rapidly spreading. In order to consider the effect of each software component on the reliability of a system developed in a distributed environment such as an open source software project, we apply AHP (Analytic Hierarchy Process) and ANP (Analytic Network Process) which are well-established decision-making methods. We also propose a method of reliability assessment based on the software reliability growth models incorporating the interaction among the components. Moreover, we analyze actual software fault count data to show numerical examples of software reliability assessment for a concurrent distributed development environment. Furthermore, we consider an efficient and effective method of software reliability assessment for actual OSS projects.

Keywords Open source software · Software reliability · Decision-making · AHP · ANP · Software reliability growth model · NHPP · Stochastic differential equation · Software component · Cloud computing · Big data

1 Characteristics of OSS

At present, OSS systems serve as key components of critical infrastructures in our society. OSS projects possess a unique feature known as software composition by which components are developed by teams that are geographically dispersed throughout the world. Successful OSS projects includes Apache HTTP server, MySQL database server, OpenStack cloud software, Firefox Web browser, and GNU/Linux

S. Yamada (✉)

Department of Social Management Engineering, Tottori University,
Minami 4-101, Koyama, Tottori-shi 680-8552, Japan
e-mail: yamada@sse.tottori-u.ac.jp

Y. Tamura

Department of Electronic and Information System Engineering,
Yamaguchi University, Tokiwadai 2-16-1, Ube-shi, Yamaguchi 755-8611, Japan
e-mail: tamura@yamaguchi-u.ac.jp

© Springer International Publishing Switzerland 2016

L. Fiondella and A. Puliafito (eds.), *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering,
DOI 10.1007/978-3-319-30599-8_23

operating system. However, poor handling of quality issues and customer support has limited the progress of OSS because the development cycle of OSS has no testing-phase. Further, mobile OSS has been gaining a lot of attention in the embedded system area, i.e., Android, BusyBox, Firefox OS, etc. However, the installer software developed by third-party developers have an effect on the reliability of a mobile device. Therefore, it is difficult for many companies to assess the reliability of a mobile OSS project, because a mobile OSS includes several software versions. Another closely related issue is that of software vulnerability posed by the open source nature of the code, raising the possibility of security loopholes. For the above-mentioned reasons, it is difficult for the software managers to assess the reliability of OSS.

We compare the characteristics of software development under the OSS and the proprietary software paradigms as follows:

OSS

1. The specification continuously changes with each version upgrade.
2. OSS has no specific testing-phase.
3. Several versions of OSS are uploaded to the website of the OSS project.
4. It is difficult to clearly distinguish between developers and users.
5. Many OSS possess different licenses.

Proprietary Software

1. The specification is fixed in the initiation phase.
2. The proprietary software has a specific testing-phase.
3. The delivery of software is specified as the software release time.
4. The user cannot access the source code. The maintenance is performed by the software engineers.
5. The user of proprietary software must contract with the software development company.

In particular, OSS have several licenses as follows:

1. GPL (GNU General Public License)
2. LGPL (GNU Lesser General Public License)
3. BSD License
4. X11 License
5. Apache Software License

There are many software licenses in a variety of OSS project areas. Therefore, it is known that it is difficult for software managers to use OSS for commercial software.

At present, many OSS are developed under several open source projects. For example, the Firefox Web browser, Firefox OS, and Thunderbird mailer are developed and managed under the Mozilla.org project. Also, Apache HTTP server, Tomcat, and Flex are developed and managed under the Apache software foundation. These OSS projects control the development phase of many OSS projects. Moreover, the specification of OSS continuously changes with the version upgrade, because the software of several versions is uploaded to the website of OSS project.

It is difficult for software managers to assess OSS reliability because of the differences among the development style of OSS and traditional software. Therefore, it is important to assess and manage OSS reliability by considering the characteristics of the OSS development paradigm.

In particular, OSS have several versions of the development process as follows:

1. Bug Fix Version (most urgent issue such as patch)
2. Minor Version (minor revision by the addition of a component and module)
3. Major Version (significant revision for specification)

Also, the version number of OSS is generally described as the “(Major version number. Minor version number. Revision number. Build number)”, e.g., (2.1.2103.1104). There are several versions for each OSS. Therefore, it is known that it is difficult for software managers to select the appropriate OSS, because several OSS are available from the website of an open source project.

This chapter is devoted to the reliability of OSS with the above-mentioned characteristics. Three methods of reliability assessment for OSS are presented. Also, several numerical examples for each method of reliability assessment are given using actual fault data of OSS. These methods may be useful for software managers to assess the reliability of a software system developed using the OSS paradigm.

2 OSS Reliability Assessment Based on NHPP Model and AHP

In order to consider the effect of each software component on the reliability of the entire system under such distributed development environment, we apply the AHP which is known as a decision-making method. Moreover, we propose the method of reliability assessment based on an SRGM incorporating the interaction among each software component.

2.1 Component-Oriented Reliability Analysis for OSS

2.1.1 NHPP Models

Many SRGMs have been used as a conventional methods to assess the reliability, quality control, and testing-process control of software development [1–4]. Among others, nonhomogeneous Poisson process (NHPP) models have been discussed by many research papers, since these NHPP models can be easily applied during the software development. In this section, we discuss NHPP models to analyze software fault-detection count data. Considering the stochastic characteristics associated with the fault-detection procedures in the testing-phase, we treat $\{N(t), t \geq 0\}$ as a nonnegative counting process where random variable $N(t)$ indicates the cumulative number

of faults detected up to testing-time t . The fault-detection process $\{N(t), t \geq 0\}$ is described as follows [1]:

$$\Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!} \exp[-H(t)] \quad (n = 0, 1, 2, \dots). \quad (1)$$

In Eq. (1), $\Pr\{A\}$ indicates the probability of event A , and $H(t)$ is called the mean value function which represents the expected cumulative number of faults detected in the testing-time-interval $(0, t]$.

According to the growth curve of the cumulative number of detected faults, we assume that the software reliability in each software component is assessed by applying the following SRGMs based on the NHPP [1]:

Exponential SRGM

Inflection S-shaped SRGM

The NHPP models have been discussed by many research papers, since the models can be easily applied in actual software projects. Moreover, we apply the method of maximum-likelihood to estimate the model parameters. Below are the expressions for various software reliability assessment measures from the NHPP models.

2.1.2 Exponential SRGM

The mean value function of the exponential SRGM is given as follows:

$$E_i(t) = a_i(1 - e^{-b_i t}) \quad (a_i > 0, b_i > 0), \quad (2)$$

where $E_i(t)$ represents the expected cumulative number of faults detected up to the module testing-time t ($t \geq 0$) is the mean value function for the i -th software component. In Eq. (2), a_i ($i = 1, 2, \dots, n$) is the expected number of initial inherent faults for the i -th software component, and b_i ($i = 1, 2, \dots, n$) the software failure rate per inherent fault for the i -th software component.

2.1.3 Inflection S-shaped SRGM

The mean value function of the inflection S-shaped SRGM is given as follows:

$$D_i(t) = \frac{a_i(1 - e^{-b_i t})}{(1 + c_i \cdot e^{-b_i t})} \quad (a_i > 0, b_i > 0, c_i > 0), \quad (3)$$

where $a_i (i = 1, 2, \dots, n)$ is the expected number of initial inherent faults for the i -th software component, and $b_i (i = 1, 2, \dots, n)$ the software failure rate per inherent fault for the i -th software component. Moreover, $c_i (i = 1, 2, \dots, n)$ represents the inflection rate for the i -th software component.

2.1.4 Goodness-of-Fit Evaluation Criteria for Applied Model

We compare the model goodness-of-fit of two conventional SRGM's for the observed data set. We use the following goodness-of-fit evaluation criteria, i.e., the Akaike's information criterion (AIC) and the mean square error (MSE). Suppose that K data pairs $(t_k, y_k) (k = 1, 2, \dots, K)$ are observed during the system testing-phase, where y_k is the cumulative number of software failures observed in the time-interval $(0, t_k]$.

(i) AIC

AIC helps us to select the optimal model among ones estimated by the method of maximum-likelihood. It is given by

$$\text{AIC} = -2 \cdot (\text{the logarithmic maximum-likelihood}) + 2 \cdot (\text{the number of free model parameters}). \tag{4}$$

Differences among AIC values are significant, not their value themselves. It can be judged that the model having the smallest AIC fits best to the actual data set when their differences are greater than or equal to 1. However, there is no significant difference among two models in the case where the differences of AIC's are less than 1.

(ii) MSE

The mean square error can be obtained by dividing the sum of square errors between the observed value, y_k , and its estimate, \hat{y}_k , by the number of pairs of data, n , i.e.,

$$\text{MSE} = \frac{1}{n} \sum_{k=1}^n (y_k - \hat{y}_k)^2. \tag{5}$$

\hat{y}_k in Eq.(5) is obtained from the estimated mean value function $\hat{H}(t)$ as $\hat{y}_k = \hat{H}(t_k) (k = 1, 2, \dots, n)$. A small mean square error indicates that the selected model fits the observed one well.

We compare the two conventional SRGM's using the above-described goodness-of-fit evaluation criteria. Concretely speaking, AIC is the first goodness-of-fit evaluation criterion, and MSE is the secondary goodness-of-fit measure, i.e., we select the appropriate model when the difference of values in AIC is greater than or equal to 1, otherwise we select the appropriate model based on the value of MSE.

2.1.5 Software Component Assessment Based on AHP

The AHP developed in the 1970s is utilized widely in Europe and the United States for management issues, energy problems, decision-making, and urban planning. The AHP is considered to be one of the most effective methods for decision-making support [5].

When considering the effect of debugging process on an entire system in the development of a software reliability assessment method for distributed development environment, it is necessary to grasp the deeply intertwined factors, such as programming path, size of a component, skill of fault reporter, and so on.

Also, it is rare that collected data sets entirely contain all the information needed to assess software reliability, although these data sets for deeply intertwined factors are collected from the bug tracking system. Therefore, it is difficult to estimate the effect of each component on the entire system using the collected data sets only.

In this chapter, we propose a reliability assessment method based on the AHP to estimate the effect of each component on the entire system in a complicated environment. Specifically, we can assess the importance level of faults detected for each component, the size of a component, the skill of fault reporter, and so on as evaluation criteria for the AHP.

Let $w_i (i = 1, 2, \dots, n)$ be the weight parameters for evaluation criteria of the AHP. Then, the pair comparison matrix is given as follows:

$$A = \begin{bmatrix} \frac{w_1}{w_1} & \frac{w_1}{w_2} & \dots & \frac{w_1}{w_n} \\ \frac{w_2}{w_1} & \frac{w_2}{w_2} & \dots & \frac{w_2}{w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{w_n}{w_1} & \frac{w_n}{w_2} & \dots & \frac{w_n}{w_n} \end{bmatrix}. \quad (6)$$

We can obtain the weight parameter α_i for each evaluation criterion from the above pair comparison matrix using the following geometric average:

$$\alpha_i = \sqrt[n]{\prod_{j=1}^n x_{ij}},$$

$$x_{ij} = \frac{w_i}{w_j}. \quad (7)$$

Therefore, the total weight parameter for each evaluation criterion is given by the following equation:

$$\beta_i = \frac{\alpha_i}{\sum_{i=1}^n \alpha_i} \tag{8}$$

Using the weight parameter β_i in Eq. (8), we can obtain the total weight parameter p_i which represents the level of importance for each component.

2.2 Reliability Analysis for Entire OSS

2.2.1 Logarithmic Execution Time Model

The operating environment of OSS possesses characteristics of the susceptible to various application software. Therefore, it is different from a conventional software system developed by an identical organization. Then the expected number of detected faults continues to increase from the effect of the interaction among application software, i.e., the number of detected faults cannot converge to a fixed value.

As mentioned above, we apply the logarithmic Poisson execution time model based on the assumption that the number of detected faults tends to infinity. Thus, we consider the following structure of the mean value function $\mu(t)$ for the entire system because an NHPP model is characterized by its mean value function:

$$\mu(t) = \frac{1}{\theta - P} \ln[\lambda_0(\theta - P)t + 1] \tag{9}$$

$(0 < \theta, 0 < \lambda_0, 0 < P < 1),$

where λ_0 is the intensity of initial inherent failure, θ the reduction rate of the failure intensity rate per inherent fault. Moreover, we assume that the parameter P in Eq. (9) represents the following weighted average in terms of the weight parameter p_i estimated by the AHP and the software failure rate per inherent fault b_i for the i -th software component in Eq. (2) or Eq. (3):

$$P = \sum_{i=1}^n p_i \cdot b_i, \tag{10}$$

where n represents the number of software components and $p_i (i = 1, 2, \dots, n)$ the weight parameter for each component.

2.2.2 Reliability Assessment Measures

We can give the following expressions as software reliability assessment measures derived from the NHPP model given by Eq. (9):

- *Software reliability*

The software reliability can be defined as the probability that a software failure does not occur during the time-interval $(t, t + x]$ ($t \geq 0, x \geq 0$) after the testing-time t . The software reliability is given by

$$R(x|t) = \exp[\mu(t) - \mu(t + x)],$$

$$(t \geq 0, x \geq 0). \quad (11)$$

- *Instantaneous mean time between software failures*

The instantaneous mean time between software failures (MTBF_I) measures the frequency of software failure occurrence, and is given by

$$\text{MTBF}_I(t) = \frac{1}{\frac{d\mu(t)}{dt}}. \quad (12)$$

- *Cumulative mean time between software failures*

The cumulative mean time between software failures (MTBF_C) is given as follows:

$$\text{MTBF}_C(t) = \frac{t}{\mu(t)}. \quad (13)$$

3 OSS Reliability Assessment Based on NHPP Model and ANP

In order to consider the effect of each software component on the reliability of an entire system in a distributed development environment, we apply the ANP (analytic network process) which is a popular decision-making method. Moreover, we propose a method of reliability assessment based on the SRGM incorporating the interaction among software components. Also we discuss a method of reliability assessment for OSS projects as a typical case of a distributed development environment.

3.1 Component-Oriented Reliability Analysis for OSS

3.1.1 Reliability Assessment Based on SRGM

We can give useful expressions for various software reliability assessment measures from the NHPP models with specified mean value functions. Based on Sects. 2.1.2, 2.1.3, and 2.1.4, we can similarly select the suitable SRGM for each software component.

3.1.2 Weight Parameter for Each Component Based on ANP

To consider the effect of debugging process on entire system in the development of a software reliability assessment method for distributed development environment, it is necessary to grasp the deeply intertwined factors, such as programming path, size of a component, skill of fault reporter, and so on.

Also, it is difficult to consider that collected data sets entirely contain the information in terms of software reliability, although these data sets for deeply intertwined factors are collected from bug tracking system. Therefore, it is difficult to estimate the effect of each component on the entire system using the collected data sets only.

In this section, we apply the reliability assessment method based on the ANP [6, 7] to estimate the effect of each component on the entire system in a complicated situation. Specifically, we apply the importance level of faults detected for each component, the skill of the fault reporter, and the skill of the fault repairer as evaluation criteria for the ANP.

For example, Fig. 1 shows the network structure in this section. “Severity”, “Assigned to” and “Reporter” are evaluation criteria, whereas “general”, “other”, “xfce4”, “xfdesktop”, “xffm”, and “xfwm” are components. The super-matrix is given as follows:

$$S = \begin{bmatrix} A_1 & 0 \\ B_{21} & A_2 \end{bmatrix} \tag{14}$$

$$\left(A_1 = 0, B_{21} = \begin{bmatrix} v \\ 0 \end{bmatrix}, A_2 = \begin{bmatrix} 0 & W \\ U & 0 \end{bmatrix} \right).$$

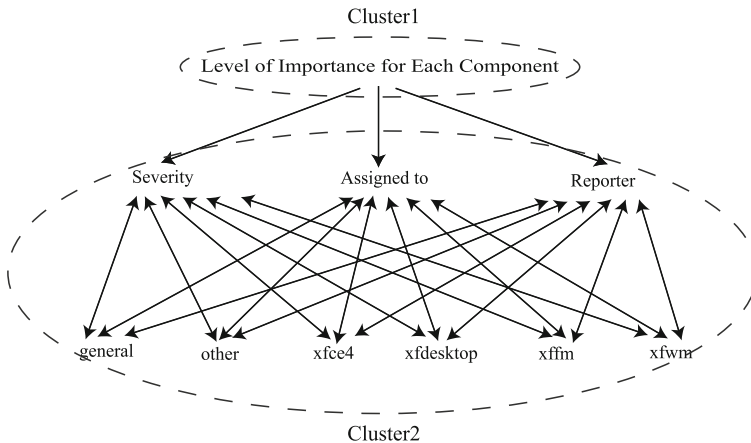


Fig. 1 Network structure of ANP

Then, A_i is the evaluation matrix in cluster i , and B_{21} the evaluation matrix from cluster 1 to cluster 2. Moreover, v is the level of importance of each component, U the evaluation matrix which influences from each component to evaluation criteria, and W the evaluation matrix which influences from evaluation criteria to each component.

First, in order to standardize a matrix, the maximum eigenvalue λ_1 and λ_2 of the partial matrix in a diagonal block A_1 and A_2 are calculated as

$$\overline{A_i} = \frac{1}{\lambda_i} A_i \quad (i = 1, 2), \tag{15}$$

$$\overline{B_{21}} = \frac{1}{\lambda_1} B_{21}. \tag{16}$$

Then, let λ_i be 1 if $A_i = 0$. Then the super-matrix is given as follows:

$$S = \begin{bmatrix} \overline{A_1} & 0 \\ \overline{B_{21}} & \overline{A_2} \end{bmatrix}. \tag{17}$$

And, $[\overline{B_{21}} \ \overline{A_2}]$ is extracted from Eq. (17). The number of the positive factor of the i -th line of this matrix is set to n_{2i} , and the matrix which divided each factor of the i -th line by n_{2i} is set to $[\widehat{B_{21}} \ \widehat{A_2}]$. Thereby, we can calculate \widehat{b}_2 as follows:

$$\widehat{b}_2 = \widehat{B_{21}} u_1. \tag{18}$$

When a cluster 1 consists of one element, it is estimated that it is $u_1 = 1$. \widehat{b}_2 is an evaluation value given from cluster 1 to cluster 2.

The parameters p_i representing the level of importance of each component for the entire system reliability can be estimated using u_2 expressed with Eq. (19) from the above-mentioned results:

$$\widehat{b}_2 + \widehat{A_2} u_2 = u_2. \tag{19}$$

3.2 Reliability Assessment for Entire System

3.2.1 Inflection S-shaped SRGM

We apply the inflection S-shaped SRGM for reliability assessment of the entire system. Thus, we consider the following structure of mean value function $S(t)$ because an NHPP model is characterized by its mean value function:

$$S(t) = \frac{a(1 - e^{-bt})}{1 + C \cdot e^{-bt}} \quad (a > 0, b > 0, C > 0), \tag{20}$$

where a is the expected number of initial inherent faults, and b the software failure rate per inherent faults. Moreover, we assume that C represents the following weighted average in terms of weight parameter p_i estimated by ANP and inflection rate c_i in Eq. (3):

$$C = \frac{\sum_{i=1}^n p_i \cdot c_i}{\sum_{i=1}^n p_i} = \sum_{i=1}^n p_i \cdot c_i, \tag{21}$$

where n represents the number of software component, p_i the weight parameter for each component, and c_i the inflection rate for the i -th software component.

3.2.2 Software Reliability Assessment Measures

We can give the following expressions as software reliability assessment measures derived from the NHPP model given in Eq. (21):

- *The expected number of remaining faults*

The expected number of faults remaining in the system at testing-time t , which is obtained as the expectation of random variable $\{N(\infty) - N(t)\}$, is given as follows:

$$N_c(t) \equiv E[N(\infty) - N(t)] = a - S(t). \tag{22}$$

- *Software reliability*

The software reliability can be defined as the probability that a software failure does not occur during the time-interval $(t, t + x] (t \geq 0, x \geq 0)$ after testing-time t . The software reliability is given by

$$R_c(x|t) = \exp[S(t) - S(t + x)] \quad (t \geq 0, x \geq 0). \tag{23}$$

4 OSS Reliability Assessment Based on Stochastic Differential Equation Model and AHP for Big Data on Cloud Computing

4.1 Software Reliability Considering Big Data on Cloud Computing

Many software reliability growth models (SRGM's) [1–4] have been applied to assess the reliability for quality management and testing progress control of software development. On the other hand, the effective methods assisting dynamic testing management for a new distributed development paradigm as typified by cloud computing have not been studied as extensively [8–10]. Also, several research papers [11–15]

have addressed the area of cloud computing and mobile clouds. However, these papers focus on security, service optimization, secure control, resource allocation technique, etc. Thus, research papers on the reliability of big data and cloud computing have presented. When considering the effect of the debugging process on the entire system in the development of a method of reliability assessment for the software developed by third-party developers, it is necessary to understand the role of installer software, network traffic, the installed software, etc. Therefore, it is very important to consider the status of network traffic on the reliability assessment from the following standpoint:

- For mobile devices, the network access devices are frequently used by many software applications installed via the installer software.
- Using the installer software, various third-party software applications are installed via the network.
- In case of open source, the weakness of reliability and computer network security becomes a significant problem.

There are also some interesting research papers on cloud hardware, cloud service, mobile clouds, and cloud performance evaluation [16, 17]. However, most of them have focused on the case studies of cloud service and cloud data storage technologies. The effective methods of dynamic reliability assessment considering the environment such as cloud computing and OSS are limited [18]. In particular, it is very important to consider the status of fault-detection and big data as they influence the reliability assessment for cloud computing from the following standpoint:

- Cloud computing has a particular maintenance phase such as the provisioning processes.
- Big data, as the results of the huge and complicated data using the internet network, causes system-wide failures because of the complexity of data management.
- Various mobile devices are connected via the network to the cloud service.
- Data storage areas for cloud computing are reconfigured via the various mobile devices.

From above reasons, it is important to consider the indirect influences of big data on reliability. We have proposed several methods of software reliability for cloud computing in the past [19, 20]. However, the effective methods of reliability assessment considering both big data factors and faults are limited, because it is very difficult to describe the indirect influence of big data and fault data as the reliability assessment measures as shown in Fig. 2. Thus, we propose a new approach to describe the indirect effect on reliability using three kinds of Brownian motions.

From the points discussed above, we consider that all factors of *big data*, *cloud computing*, and *network access* have an effect on cloud computing, directly and indirectly. In other words, big data and cloud computing have deep and complex influences on reliability. Therefore, it is very important to consider big data from the point of view of reliability for cloud computing, i.e., to ensure stable operation of

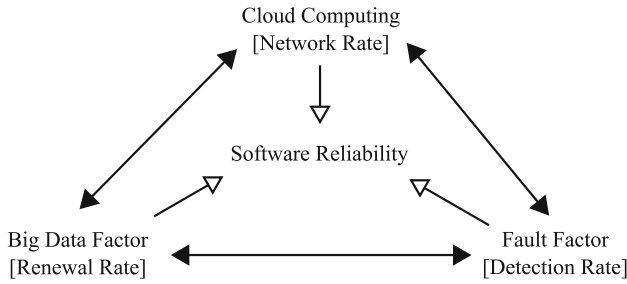


Fig. 2 The relationship among big data, cloud computing, network, and reliability

cloud computing by offering several reliability assessment measures considering the big data in terms of all factors of *cloud computing*, *mobile clouds*, and *open source software*. We now introduce *3V's model* defined by Gartner Group, Inc. [21] for the big data.

4.2 Weight Parameter Based on AHP

In case of considering the effect of debugging process on an entire system in the development of a method of software reliability assessment for big data on cloud computing, it is necessary to grasp the deeply intertwined factors, such as the characteristics of big data, the application of cloud computing, the system reliability, and so on.

We discuss a method of reliability assessment based on the AHP in terms of estimating the effect of each factor on the entire big data on cloud computing in a complex situation. In particular, we can apply the *3V's model* for describing big data, to the evaluation criteria of the AHP. The *3V's model* in the big data means Volume, Velocity, and Variety. The *3V's model* is defined by Gartner Group, Inc. [21]. The Volume, Velocity, and Variety are very important to assess the big data in terms of the external factor for reliability.

In Sect. 2.1.5, using the weight parameter $\beta_i (i = 1, 2, 3)$ in Eq. (8), we can obtain the total weight parameter $p_i (i = 1, 2, 3)$ which represents the alternative of AHP. In this paper, the Volume, Velocity, and Variety of *3V's model* are applied to the evaluation criteria of the AHP. Moreover, we consider three probability as the alternative of AHP, i.e., the changing rate of network traffic per unit time, the renewal rate of data per unit time, and the detection rate of fault per unit time. Figure 3 shows the basic concept of factor analysis for big data by using AHP in this chapter.

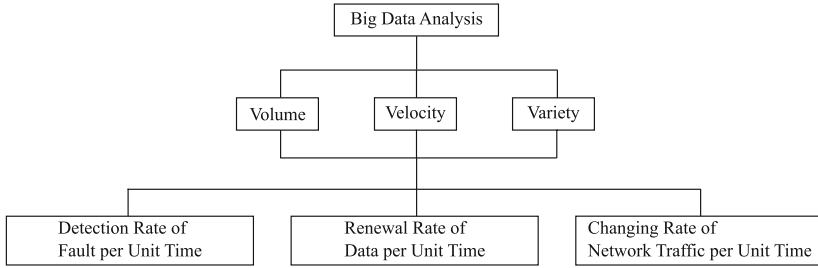


Fig. 3 The basic concept of factor analysis for big data

4.3 Stochastic Differential Equation Modeling

Let $M(t)$ be the cumulative number of faults detected by operation time t ($t \geq 0$) in the cloud software. Suppose that $M(t)$ takes on continuous real values. Since latent faults in the cloud software are detected and eliminated during the operation phase, $M(t)$ gradually increases as the operational procedures go on. Thus, under common assumptions for software reliability growth modeling [1], the following linear differential equation can be formulated:

$$\frac{dM(t)}{dt} = b(t)\{R(t) - M(t)\}, \tag{24}$$

where $b(t)$ is the software fault-detection rate at operation time t and a nonnegative function, $R(t)$, means the amount of changes of requirements specification [22]. Also, $R(t)$ is defined as follows:

$$R(t) = \alpha e^{-\beta t}, \tag{25}$$

where α is the number of faults latent in the cloud OSS, and β the changing rate of requirements specification. It is assumed that the fault-prone requirements specification of OSS grows exponentially in terms of t [22]. Thus, the OSS shows a reliability regression trend if β is negative. On the other hand, the OSS shows a reliability growth trend if β is positive.

Considering the characteristic of the big data on cloud computing, the software fault-reporting phenomena keep an irregular state in the operation phase, due to the network access of several users. In case of OSS, we have to consider that OSS fault-prone specification depends on the operation time. In particular, cloud computing has the unique characteristics of provisioning process. At present, the amount of data used by cloud users becomes large. Then we consider the big data in order to assess the reliability for cloud computing. Therefore, we extend Eq. (24) to the following stochastic differential equation modeling with three Brownian motions [23, 24]:

$$\frac{dM_1(t)}{dt} = \{b_1(t) + \sigma_1 v_1(t)\} \{R_1(t) - M_1(t)\}, \tag{26}$$

$$\frac{dM_2(t)}{dt} = \{b_2(t) + \sigma_2 v_2(t)\} \{R_2(t) - M_2(t)\}, \tag{27}$$

$$\frac{dM_3(t)}{dt} = \{b_3(t) + \sigma_3 v_3(t)\} \{R_3(t) - M_3(t)\}, \tag{28}$$

where σ_1, σ_2 and σ_3 are a positive constant representing a magnitude of the irregular fluctuation, $v_1(t), v_2$ and $v_3(t)$ a standardized Gaussian white noise. We assume that $M_1(t)$ and $R_1(t)$ are related with the software fault-detection rate $b_1(t)$ depending on the failure occurrence phenomenon. Also, $M_2(t)$ and $R_2(t)$ are related with the software fault-detection rate $b_2(t)$ depending on the big data. Moreover $M_3(t)$ and $R_3(t)$ are related with the software fault-detection rate $b_3(t)$ depending on the network of cloud computing. Considering the independent of each noise, we can obtain the following integrated stochastic differential equation:

$$\frac{dM(t)}{dt} = \{b(t) + \sigma_1 v_1(t) + \sigma_2 v_2(t) + \sigma_3 v_3(t)\} \{R(t) - M(t)\}. \tag{29}$$

We extend Eqs. (26)–(28) to the following stochastic differential equations of an Itô type [25]:

$$dM_1(t) = \left\{ b_1(t) - \frac{1}{2}\sigma_1^2 \right\} \{R_1(t) - M_1(t)\}dt + \sigma_1 \{R_1(t) - M_1(t)\}d\omega_1(t), \tag{30}$$

$$dM_2(t) = \left\{ b_2(t) - \frac{1}{2}\sigma_2^2 \right\} \{R_2(t) - M_2(t)\}dt + \sigma_2 \{R_2(t) - M_2(t)\}d\omega_2(t), \tag{31}$$

$$dM_3(t) = \left\{ b_3(t) - \frac{1}{2}\sigma_3^2 \right\} \{R_3(t) - M_3(t)\}dt + \sigma_3 \{R_3(t) - M_3(t)\}d\omega_3(t), \tag{32}$$

where $\omega_i(t)$ is i -th one-dimensional Wiener process which is formally defined as an integration of the white noise $v_i(t)$ with respect to time t . Similarly, we can obtain the following integrated stochastic differential equation based on the independent of each noise:

$$\begin{aligned} dM(t) = & \left\{ b(t) - \frac{1}{2}(\sigma_1 + \sigma_2 + \sigma_3)^2 \right\} \{R(t) - M(t)\}dt \\ & + \sigma_1 \{R(t) - M(t)\}d\omega_1(t) + \sigma_2 \{R(t) - M(t)\}d\omega_2(t) \\ & + \sigma_3 \{R(t) - M(t)\}d\omega_3(t). \end{aligned} \tag{33}$$

We define the three-dimension processes $[\omega_1(t), \omega_2(t), \omega_3(t)]$ as follows [26]:

$$\tilde{\omega}(t) = (\sigma_1^2 + \sigma_2^2 + \sigma_3^2)^{-\frac{1}{2}} \{ \sigma_1 \omega_1(t) + \sigma_2 \omega_2(t) + \sigma_3 \omega_3(t) \}. \tag{34}$$

Then, three Wiener processes, $\tilde{\omega}(t)$, are Gaussian processes and have the following properties:

$$\Pr[\tilde{\omega}(0) = 0] = 1, \tag{35}$$

$$E[\tilde{\omega}(t)] = 0, \tag{36}$$

$$E[\tilde{\omega}(t)\tilde{\omega}(t')] = \text{Min}[t, t'], \tag{37}$$

where $\Pr[\cdot]$ and $E[\cdot]$ represent the probability and expectation, respectively.

Using Itô's formula [23, 24], we can obtain the solution of Eq. (33) under the initial condition $M(0) = 0$ as follows [25]:

$$M(t) = R(t) \left[1 - \exp \left\{ - \int_0^t b(s)ds - \sigma_1\omega_1(t) - \sigma_2\omega_2(t) - \sigma_3\omega_3(t) \right\} \right]. \tag{38}$$

Using solution process $M(t)$ in Eq. (38), we can derive several software reliability measures.

Moreover, we define the software fault-detection rate per fault in case of $b(t)$ defined as

$$\begin{aligned} b(t) &\doteq \frac{\frac{dI(t)}{dt}}{a - I(t)} \\ &= \frac{b}{1 + c \cdot \exp(-bt)}, \end{aligned} \tag{39}$$

where $I(t)$ means the mean value functions for the inflection S-shaped SRGM, based on a nonhomogeneous Poisson process (NHPP) [1], a the expected total number of latent faults for SRGM, and b the fault-detection rate per fault. Generally, the parameter c is defined as $\frac{(1-l)}{l}$. We define the parameter l as the value of fault factor estimated using the method of AHP.

Therefore, the cumulative numbers of detected faults are obtained as follows:

$$\begin{aligned} M(t) &= R(t) \left[1 - \frac{1 + c}{1 + c \cdot \exp(-bt)} \right. \\ &\quad \left. \cdot \exp \left\{ - bt - \sigma_1\omega_1(t) - \sigma_2\omega_2(t) - \sigma_3\omega_3(t) \right\} \right]. \end{aligned} \tag{40}$$

In the proposed model, we assume that the parameter σ_1 depends on the failure occurrence phenomenon. Also, we assume that the parameter σ_2 depends on the network changing rate per unit time resulting from the cloud computing. Moreover, we assume that the parameter σ_3 depends on the renewal rate per unit time resulting from the big data.

5 Numerical Examples

In the above discussions, we have presented several methods of OSS reliability assessment based on AHP and ANP. In this section, we show several numerical examples OSS reliability assessment based on AHP and ANP.

5.1 Reliability Assessment for Application Software on X Window System

As an example, we show several numerical examples based on Sect. 3. We focus on the Xfce desktop environment which is one of the software developed under OSS project. Xfce is a lightweight desktop environment for UNIX-like operating system. It aims to be fast and lightweight, while still being visually appealing and easy to use. The data used in this section is collected in the bug tracking system of the website of Xfce [27].

We show the cumulative number of detected faults in each component for actual data in Fig. 4. The estimated results of weight parameter $p_i (i = 1, 2, \dots, n)$ for each component based on ANP in Sect. 3 are shown in Table 1. Especially, the applied evaluation criterion are the importance level of faults detected for each component (Severity), the fault repairer (Assigned to), and the fault reporter (Reporter). From Table 1, we find that the level of importance for “other” component is largest. On the other hand, we find that the level of importance for “general” component is the smallest.

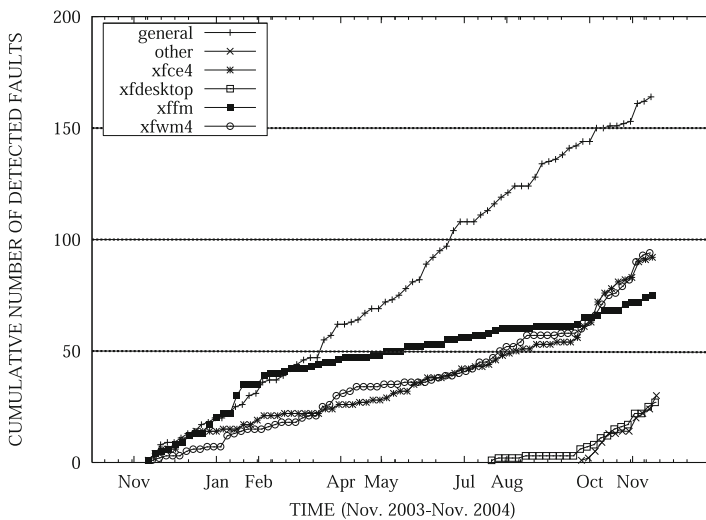


Fig. 4 The cumulative number of detected faults in each component for actual data

Table 1 The estimated results of the weight parameter for each component based on ANP

Component	Weight parameter p_i
general	0.0550295
other	0.444284
xfce4	0.093274
xfdesktop	0.1827720
xffm	0.122944
xfwm	0.101697

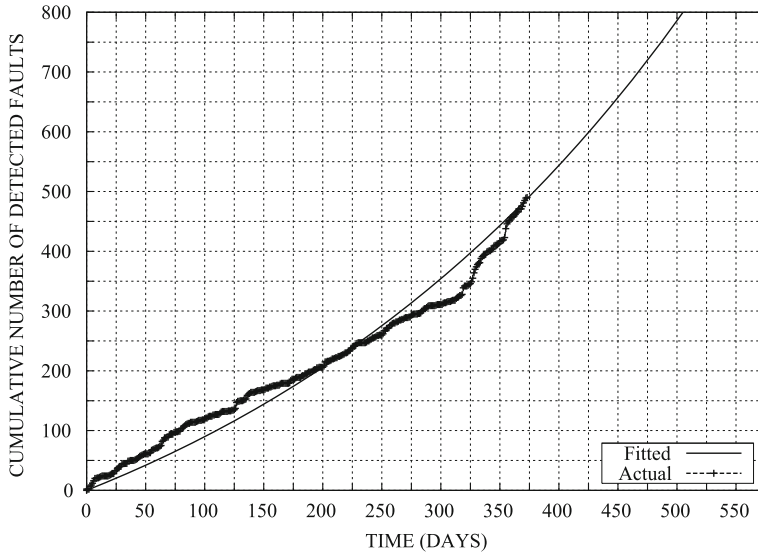


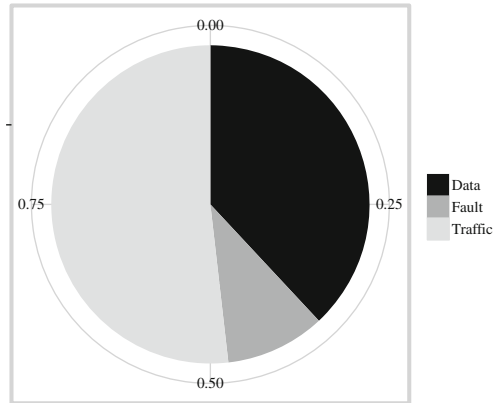
Fig. 5 The estimated cumulative number of detected faults, $\hat{S}(t)$

Based on the estimation results in Table 1, we show the estimated cumulative number of detected faults in Eq. (20), $\hat{S}(t)$, for Xfce desktop environment in Fig. 5.

5.2 Reliability Assessment for Cloud Computing

As an example, we show several numerical examples based on Sect. 4. The OSS is closely watched from the point of view of the cost reduction and the quick delivery. There are several OSS projects in area of cloud computing. In particular, we focus on OpenStack [28] in order to evaluate the performance of our method. In this section, we show numerical examples of reliability assessment measures using the data sets for OpenStack of cloud OSS. The data used in this chapter are collected in the bug tracking system on the website of OpenStack OSS project.

Fig. 6 The estimated results of the alternative based on AHP



The estimated results of the weight parameters of the alternative for each factor based on AHP discussed in Sect. 2 are shown in Fig. 6. Then, the estimated consistency index of AHP is 0.0678. From Fig. 6, we can find that the level of importance for the changing rate of network traffic is largest. On the other hand, we can find that the level of importance for the fault-detection rate is smallest. These results mean that the cloud computing keeps the stable throughput because the cloud software becomes stable.

The sample path of the estimated number of detected faults for the fault and network factors in Eq. (40) is shown in Fig. 7. Similarly, the sample path of the estimated number of detected faults for the fault and big data factors in Eq. (40) is shown in Fig. 8. Moreover, the sample path of the estimated number of detected faults for the network and big data factors in Eq. (40) is shown in Fig. 9. From Figs. 7, 8 and 9, we can confirm that the noise of network factor becomes large in the early operation phase of cloud computing. On the other hand, we can confirm that the noise of fault factor becomes small in all operating phase of cloud computing. Therefore, we find that the cloud computing environment in this case keeps in stable condition in terms of the reliability.

From above-mentioned results, we have found that our model can describe the characteristics of the big data on cloud computing according to the changes of the fault, the changing rate per unit time of network traffic, and the renewal rate per unit time of big data. The proposed method will be useful to assess the reliability of the characteristics of big data on cloud computing.

6 Concluding Remarks

In this chapter, we have focused on a software development paradigm based on an OSS project. In order to consider the effect of software systems, each software component on the reliability of an entire system under distributed development

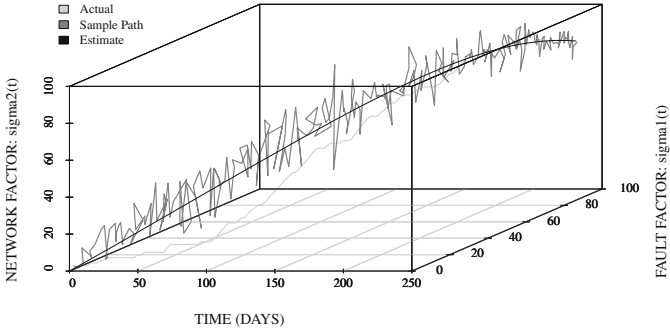


Fig. 7 The sample path of the number of detected faults (fault and network factors)

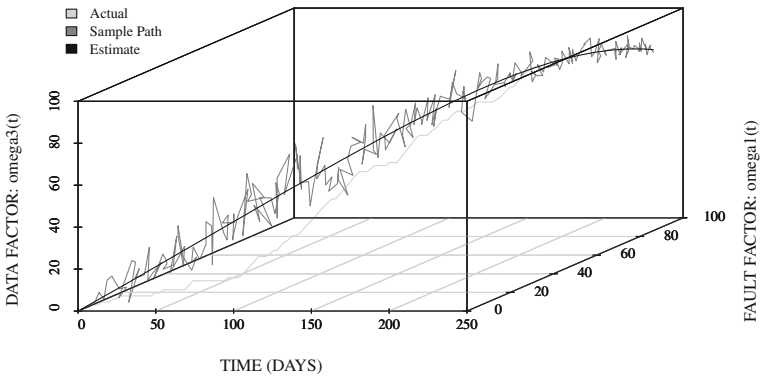


Fig. 8 The sample path of the number of detected faults (fault and big data factors)

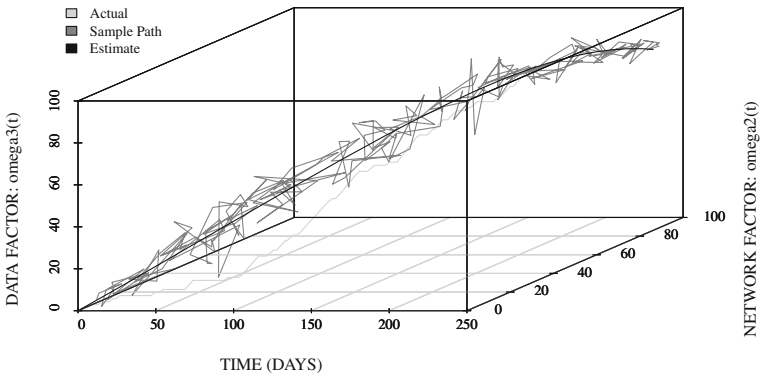


Fig. 9 The sample path of the number of detected faults (network and big data factors)

environment such as OSS project, we have applied AHP and ANP which are known as one of the method of decision-making. Also we have discussed a method of reliability assessment based on the software reliability growth models incorporating the interaction among software components. Moreover, we have considered an efficient and effective method of software reliability assessment for the actual OSS project.

We have focused on the cloud computing with big data. In particular, we have applied AHP which is known as one of the method of decision-making in order to consider the characteristics of cloud computing under big data. Using the AHP, we have proposed the method of reliability assessment incorporating the interaction among 3V's on big data. The AHP and stochastic differential equation models applied in this chapter have the simple structure. Therefore, we can easily apply our method to actual cloud computing project under big data.

When considering the effect of external factors on an entire system in the development of software reliability assessment methods for cloud computing, it is necessary to grasp the deeply intertwined factors. In this chapter, we have shown that the proposed method can grasp such deeply intertwined factors by using the weight parameters of evaluation criteria for 3V's model of big data in AHP. Moreover, we have given several software reliability assessment measures based on the proposed method. Also, we have analyzed actual data to show numerical examples of software reliability assessment for the cloud computing. Our methods may be useful as the OSS reliability assessment approach.

References

1. Yamada S (2014) Software reliability modeling: fundamentals and applications. Springer, Heidelberg
2. Lyu MR (ed) (1996) Handbook of software reliability engineering. IEEE Computer Society Press, Los Alamitos
3. Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York
4. Kapur PK, Pham H, Gupta A, Jha PC (2011) Software reliability assessment with OR applications. Springer, London
5. Satty T (1980) The analytic hierarchy process. McGraw-Hill, New York
6. Kinoshita E (2000) Introduction to AHP. JUSE Press, Tokyo
7. Kinoshita E (2000) Theory of AHP and its application. JUSE Press, Tokyo
8. Li X, Li YF, Xie M, Ng SH (2011) Reliability analysis and optimal version-updating for open source software. *J Inf Softw Technol* 53(9):929–936
9. Ullah N, Morisio M, Vetro A (2012) A comparative analysis of software reliability growth models using defects data of closed and open source software. In: Proceedings of the 35th IEEE software engineering workshop, Greece, 2012, pp 187–192
10. Cotroneo D, Grottke M, Natella R, Pietrantuono R, Trivedi KS (2013) Fault triggers in open-source software: an experience report. In: Proceedings of the 24th IEEE international symposium on software reliability engineering, Pasadena, CA, 2013, pp 178–187
11. Park J, Yu HC, Lee EY (2012) Resource allocation techniques based on availability and movement reliability for mobile cloud computing. In: Distributed computing and internet technology. Lecture notes in computer science, vol 7154. Springer, Berlin, pp 263–264

12. Suo H, Liu Z, Wan J, Zhou K (2013) Security and privacy in mobile cloud computing. In: Proceedings of the 9th international wireless communications and mobile computing conference, Cagliari, Italy, 2013, pp 655–659
13. Khalifa A, Eltoweissy M (2013) Collaborative autonomic resource management system for mobile cloud computing. In: Proceedings of the fourth international conference on cloud computing, GRIDs, and virtualization, Valencia, Spain, 2013, pp 115–121
14. Gabner R, Schwefel HP, Hummel KA, Haring G (2011) Optimal model-based policies for component migration of mobile cloud services. In: Proceedings of the 10th IEEE international symposium on network computing and applications, Cambridge, MA, USA, 2011, pp 195–202
15. Park N (2011) Secure data access control scheme using type-based re-encryption in cloud environment. In: Semantic methods for knowledge management and communication. Studies in computational intelligence, vol 381. Springer, Berlin, pp 319–327
16. Iosup A, Ostermann S, Yigitbasi MN, Prodan R, Fahringer T, Epema DHJ (2011) Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans Parallel Distrib Syst* 22(6):931–945
17. Khalifa A, Eltoweissy M (2013) Collaborative autonomic resource management system for mobile cloud computing. In: Proceedings of the fourth international conference on cloud computing, GRIDs, and virtualization, Valencia, Spain, 2013, pp 115–121
18. Cotroneo D, Grottke M, Natella R, Pietrantuono R, Trivedi KS (2013) Fault triggers in open-source software: an experience report. In: Proceedings of the 24th IEEE international symposium on software reliability engineering, Pasadena, CA, 2013, pp 178–187
19. Tamura Y, Miyahara H, Yamada S (2012) Reliability analysis based on jump diffusion models for an open source cloud computing. In: Proceedings of the IEEE international conference on industrial engineering and engineering management, Hong Kong Convention and Exhibition Centre, Hong Kong, 2012, pp 752–756
20. Tamura Y, Yamada S (2010) Reliability analysis methods for an embedded open source software. Mechatronic systems, simulation, modelling and control, IN-TECH, Vienna Austria, European Union, March 2010
21. Pettey C, Goasduff L (2011) Gartner special report: examines how to leverage pattern-based strategy to gain value in Big Data, 2011 Press Releases, Gartner Inc., 27 June 2011
22. Yamada S, Fujiwara T (2001) Testing-domain dependent software reliability growth models and their comparisons of goodness-of-fit. *Int J Reliab Qual Saf Eng* 8(3):205–218
23. Arnold L (1974) Stochastic differential equations-theory and applications. Wiley, New York
24. Wong E (1971) Stochastic processes in information and systems. McGraw-Hill, New York
25. Yamada S, Kimura M, Tanaka H, Osaki S (1994) Software reliability measurement and assessment with stochastic differential equations. *IEICE Trans Fundam E77-A(1)*:109–116
26. Mikosch T (1998) Elementary stochastic calculus, with finance in view. Advanced series on statistical science and applied probability, vol 6. World Scientific, Singapore
27. Xfce Desktop Environment, Xfce Development Team, <http://www.xfce.org>
28. The OpenStack project, OpenStack, <http://www.openstack.org/>

Measuring the Resiliency of Extreme-Scale Computing Environments

Catello Di Martino, Zbigniew Kalbarczyk and Ravishankar Iyer

Abstract This chapter presents a case study on how to characterize the resiliency of large-scale computers. The analysis focuses on the failures and errors of Blue Waters, the Cray hybrid (CPU/GPU) supercomputer at the University of Illinois at Urbana-Champaign. The characterization is performed by a joint analysis of several data sources, which include workload and error/failure logs as well as manual failure reports. We describe *LogDiver*, a tool to automate the data preprocessing and metric computation that measure the impact of system errors and failures on user applications, i.e., the compiled programs launched by user jobs that can execute across one or more XE (CPU) or XK (CPU+GPU) nodes. Results include (i) a characterization of the root causes of single node failures; (ii) a direct assessment of the effectiveness of system-level failover and of memory, processor, network, GPU accelerator, and file system error resiliency; (iii) an analysis of system-wide outages; (iv) analysis of application resiliency to system-related errors; and (v) insight into the relationship between application scale and resiliency across different error categories.

C. Di Martino (✉)
Bell Labs - Nokia,
600 Mountain Ave, New Provincence, NJ 07974, USA
e-mail: lelio.di_martino@nokia.com

Z. Kalbarczyk · R. Iyer
Coordinated Science Laboratory, University of Illinois at Urbana Champaign,
1307 W Main St, Urbana, IL 61801, USA
e-mail: kalbarcz@illinois.edu

R. Iyer
e-mail: rkiyer@illinois.edu

1 Introduction

Failures are inevitable in large-scale, high-performance computing systems (HPCs). Error resiliency (i.e., the ability to compute through failures) is strategic for providing sustained performance at scale. In this context, it is important to measure and quantify what makes current systems unreliable. Such analyses can also drive the innovation essential for addressing resiliency in exascale computing.

Analysis of data from today's extreme-scale HPC systems provides an unparalleled understanding of the resilience problem as it relates to current and future generations of extreme-scale systems, including fault and error types, probabilities, and propagation patterns.

Although methods for the design and evaluation of fault-tolerant systems have been extensively researched, little is known about how well these strategies work in the field. A study of production systems is valuable not only for accurate evaluation but also to identify reliability bottlenecks in system design.

Current large-scale supercomputers are equipped with an unprecedented number of sensors generating events and numerical data for continuous collection and analysis. Because of the extreme volume of data generated, in practice only a share of the data is collected and analyzed, the purpose being to guide improvements of system maintenance and resiliency. Computer logs represent a valuable source of data to conduct resiliency analyses at different levels, i.e., to determine how systems and applications can withstand run-time errors. The logs are machine-generated, often human-readable files reporting sequences of entries (log events) generated by the hardware, operating system and daemons, middleware, network devices, and applications in relation to regular and anomalous activities that occurred during the system operational phase. The collected data contain a large amount of information about naturally occurring errors and failures. Analysis of this data can provide understanding of actual error and failure characteristics and insight into analytical models.

An important and often neglected step in addressing the resiliency challenge of extreme-scale machines is to understand how system errors and failures impact applications and how resiliency is affected by application characteristics. In HPC systems to date, application resilience to failure has been accomplished by the brute-force method of checkpoint/restart. This method allows an application to make forward progress in the face of system faults, errors, and failures independent of the root cause or end result. It has remained the primary resiliency mechanism because we lack a way to identify faults and to project results early enough to take meaningful mitigating action. Because we have not yet operated at scales at which checkpoint/restart cannot help, vendors have had little motivation to provide the instrumentation necessary for early identification. However, as we move from petascale to exascale, mean time to failure (MTTF) will render the existing techniques ineffective. An in-depth characterization of the application failures caused by system-related issues is essential to assessing the resiliency of current systems and central to guiding the design of resiliency mechanisms to handle the extreme-scale machines of the future.

Modern supercomputers are equipped with fault-tolerant infrastructures that are capable of protecting job and application executions from failures due to hardware or software problems. The important question is: what are the errors and failures that affect the resiliency of the system and of the jobs and applications executing on supercomputers? This chapter presents an example measurement-based resiliency analysis of Blue Waters, the sustained petascale hybrid machine at the University of Illinois. The major findings of the study described in this chapter are as follows:

- Software is the cause of 74.4 % of system-wide outages (SWOs). Close to two-thirds (62 %) of software-caused SWOs resulted from failure/inadequacies of the failover procedures, such as those invoked to handle Lustre failures. More research is needed to improve failover techniques, including the way failure recovery software modules are tested in large-scale settings.
- Hardware is highly resilient to errors. Out of 1,544,398 machine check exceptions analyzed, only 28 (0.003 %) resulted in uncorrectable errors, showing the value of the adopted protection mechanisms (Chipkill and ECC). The GPU accelerator DDR5 memory, protected only with ECC, is 100 times more sensitive to uncorrectable errors than DDR3 node RAM. This finding shows the need for better techniques to protect the memory of GPU accelerators memory when both CPUs and GPU accelerators are used to create future large-scale hybrid systems.
- On average, an application failure caused by a system-related issue occurs every 15 min. In total, those applications run for 17,952,261 node hours, i.e., about 9 % of the total production node hours. While the number of failed applications is low compared to the number of executed applications, our measurements show that the impact of errors on applications is nonnegligible. Considering an average power consumption of 2 KW/blade [1], failed applications that are not recovered through checkpoint/restart add potentially \$421,878 to the Blue Waters energy bill (based on a cost of 0.047 c/KW).
- 37 % of failed applications fail during/after failover operations. Although the system can survive a sustained number of failures while preserving high availability, the level of resiliency perceived by the user may suffer. Our data shows that failures tolerated at the system level can still be harmful to applications. System-level resiliency mechanisms should interplay more profoundly with the workload management system and application-level resiliency mechanisms; this will avoid workload disruptions and reduce the impact of system outages on user activities.
- While our measurements show that 1.53 % of the applications failed because of system problems, the variation in failures is quite large. We measured an increase of 20x in the application failure probability (from 0.008 to 0.162) when scaling XE applications from 10,000 to 22,000 nodes. Similarly, for XK applications we measured an increase from 0.02 to 0.129 in the application failure probability when scaling the applications from 2000 to 4224 nodes. In both platforms, small-scale applications (i.e., those not exceeding a blade) have a failure probability due to system-related issues on the order of $1E - 5$.
- Our measurements show that the probability of application failure can be modeled with a cubic function of the node hours for XK applications and a quadratic func-

tion of the node hours for XE applications. This finding emphasizes the need for (i) dedicated resiliency techniques to be deployed for preventing error propagation from the hardware to the application in the hybrid nodes, and (ii) effective assessment techniques at extreme scale to harness hybrid computing power in future machines.

2 Blue Waters Overview

Blue Waters is a sustained petaflop system capable of delivering approximately 13.1 petaflops (at peak) for a range of real-world scientific and engineering applications. The system, schematized in Fig. 1, includes the following components:

- The Mainframe, consisting of 276 Cray liquid-cooled cabinets hosting 27,648 nodes and 1.47 PB of RAM across 197,032 RAM DIMMs. Each cabinet consists of an L1 cabinet controller, several fan trays, power conversion electronics, breakers, a blower and chiller, and related piping. Each cabinet is organized into 3 chassis, and each chassis hosts 8 blades;
- 22,640 XE6 compute nodes (based on AMD Opteron processors) with a total of 724,480 integer cores;
- 4,224 GPU hybrid nodes equipped with Nvidia K20X GPU accelerators and AMD Opteron processors;
- 784 service nodes with a total of 5,824 available cores;
- The high-speed Cray Gemini network to provide node connectivity;
- The online storage system, consisting of 36 Cray Sonexion 1600 racks, equipped with 17,280 active disks, 864 hot spares, and 396 SSDs (used to store file system

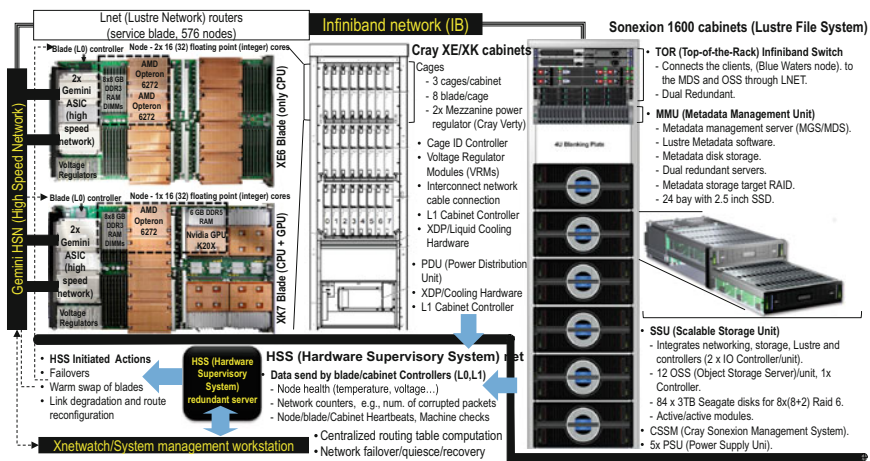


Fig. 1 Blue Waters main components and resiliency features

metadata) that provide access to 26 petabytes (36 raw) of usable storage over a Lustre distributed file system.

CPU Nodes. Compute nodes are hosted in 5,660 Cray XE6 blades (see Fig. 1), 4 nodes per blade. A compute node consists of 2 16-core AMD Opteron 6276 processors at 2.6 GHz. Each Opteron includes 8 dual-core AMD Bulldozer modules, each with an 8 × 64 KB L1 instruction cache; a 16 × 16 KB L1 data cache; an 8 × 2 MB L2 cache (shared between the cores of each Bulldozer module), and a 2 × 8 MB L3 cache (shared among all the cores). Each compute node is equipped with 64 GB of DDR3 RAM in 8 GB DIMMs.

GPU Nodes. GPU (hybrid) nodes are hosted in 768 Cray XK7 blades, 4 nodes per blade (see Fig. 1). A GPU node consists of a 16-cores Opteron 6272 processor equipped with 32 GB of DDR3 RAM in 8 GB DIMMs and a Nvidia K20X accelerator. The accelerators are equipped with 2,880 single-precision Cuda cores, 64 KB of L1 cache, 1,536 KB of dedicated L2 cache, and 6 GB of DDR5 RAM memory.

Service Nodes. Service nodes are hosted on 166 Cray XIO blades and 30 XE6 blades, 4 nodes per blade. Each XIO service node consists of a 6-core AMD Opteron 2435 Istanbul working at 2.3 GHz and equipped with 16 GB of DDR2 memory in 4 GB DIMMs protected by ×4 Chipkill (with single symbol error correction and dual symbol error detection capabilities, with 4-bit per symbol). Of the service nodes, there are 582 that act as LNET router nodes for the three Cray Sonexion-1600 file systems (described in the following).

The Gemini High-Speed Network (HSN). Blue Waters, high-speed network consists of a Cray Gemini System Interconnect. Each blade includes a network mezzanine card that houses 2 network chips, each one attached on the HyperTransport AMD bus shared by 2 CPUs and powered by 2 mezzanine dual-redundant voltage regulator modules (Cray Verty). The topology is a 3-dimensional 23 × 24 × 24 reentrant torus: each node has 6 possible links toward other nodes, i.e., right, left, up, down, in, and out.

Table 1 Main components of the Blue Waters' lustre file systems

	Home	Project	Scratch	Total
Sonexion 1600 racks	3	3	30	36
MMU (Metadata Management Servers—running Metadata Storage Servers MDS)	2	2	2	6
SSUs (Scalable Storage Unit)	18	18	180	216
OSSs (Object Storage Servers)	36	36	360	432
OSTs (Object Storage Targets)	144	144	1,440	1,728
Active disks	1,440	1,440	14,400	17,280
Hot spares	72	72	720	864
LNET router nodes	48	48	480	576
Capacity (PB)	3	3	20	26

The Lustre File System. All blades are diskless and use the shared parallel file system for IO operations. Blue Waters includes 3 file systems (project, scratch, and home) and provides up to 26 PB of usable storage over 36 PB of raw disk space. Blue Waters hosts the largest Lustre installation to date. It consists of a parallel file system used to manage data stored in Cray Sonexion 1600 [2] racks. The main components of the Lustre file systems used in Blue Waters are summarized in Table 1.

Each rack of the file system is composed of 6 SSUs (Scalable Storage Unit, for storing data) and 1 MMU (Metadata Management Unit, for storing file system metadata). Each SSU is composed of 2 Object Storage Servers (OSSs, serving the storage requests from clients, namely XE and XK nodes through the Lnet) that act as each other's High Availability (HA) partner. Therefore, there are 12 OSSs in each rack. Each OSS controls 4 Object Storage Targets (OSTs, i.e., raid volumes of disks for storing data).

Each rack comes with 2 InfiniBand (IB) switches for linking the MMU, SSUs, and Lustre clients. The IB cabling within each rack is such that all OSSs on the left side are plugged into 1 Top-Of-Rack (TOR) IB switch, while all OSSs on the right side are plugged into another TOR IB switch. The rack also contains all power supplies, InfiniBand and Ethernet cabling, and a dual Gigabit Ethernet switch for management system connections to individual components. Both SSU and MMU run a customized version of Linux combined with the Cray Sonexion System Manager (CSSM) to provide status and control of all system components, including storage hardware, RAID, operating system, and the Lustre file system.

Each MMU is configured in a 24-bay 2.5-in. drive enclosure with 22×2 TB 10K RPM disk drives. An SSU is housed in drawers containing 84-bay 3.5-in. drive enclosure with 80×2 TB disk drives used to provide data storage in an $8 \times (8 + 2)$ RAID6 target configuration, 2 global hot spares to automatically provide failover for a failed drive, and 2×2 TB solid-state drives in RAID 1 configuration for journaling and logging. An SSU expansion enclosure, which has the same drive configuration as the base SSU, can be added via SAS connections to double the usable capacity for a given bandwidth.

System Software. Compute and GPU nodes execute the lightweight kernel Compute Node Linux (CNL) developed by Cray. The operating system is reduced to a bare minimum to minimize the overhead on the nodes. It includes only essential components, such as a process loader, Virtual Memory Manager, and set of Cray ALPS agents for loading and controlling jobs. Service nodes execute a full-featured version of Linux, the Cray Linux Environment (CLE), which is based on the Suse Linux Enterprise Server 11 kernel 3.0.42.

Jobs are scheduled through a suite of cooperating software that includes (i) Cray ALPS (Application-Level Placement Scheduler) [3] for the placement, launch, and monitoring of all the applications composing a single job, and (ii) Moab and TORQUE [4] for resource management and scheduling decisions. Jobs are assigned at the granularity of the node.

Job and Application Schedulers. Moab and Torque [4] provide batch scheduling and workload management, respectively, for user jobs. Torque provides the low-level functionality to start, hold, cancel, and monitor jobs. Moab applies site policies and

```
1 #!/bin/bash --login
2 #PBS -N job name           #gives a name to the job
3 #select the needed computing resources
4 #PBS -l nodes=NUM NOES:ppn=NUMPER PROCESSING ELEMENT PER NODE
5 #PBS -m bea
6 #PBS -M dimart@illinois.edu      ### set email notification
7 # PBS -l walltime=12:00:00 ## Maximum allowed time (wall clock). MAX=24h
8 basedir=$PBS O WORKDIR      ## The base directory
9 cd $PBS O WORKDIR           ##moves to the base directory
10 module load namd/2.9        ## Load the needed modules, i.e., namd
11 #setup the environment
12 export MPICH_MAX_SHORT_MSG_SIZE=50000
13 export MPICH_PTL_UNEX_EVENTS=80000
14 export MPICH_UNEX_BUFFER_SIZE=100M
15 make output dirs.sh output ##execute a script to prepare the directories
16 aprun -n 32 -N 32 get_data.x #create 32*32 instances of get_data.x,
17                               # 32 times on 8 nodes, each using 1 core.
18                               #launches 4 applications in parallel
19 for i in {1..4}; do
20   cd $basedir/simulation${i}/  ##navigates in the subdirectories
21   aprun -n 2048 -N 32 namd2 +replicas 8 <input${i}> +stdout output${i} &
22 done ##launch a namd2 study on 2048 nodes on the input in the current dir.
23 wait                          ##Wait for all applications to complete before exiting
```

Fig. 2 Example of script for job and applications execution in Blue Waters

extensive optimizations to orchestrate jobs, services, and other workloads across the ideal combination of network, compute, and storage resources. Moab enables true adaptive computing, allowing compute resources to be customized to changing needs and failed systems to be automatically fixed or replaced.

Cray ALPS (Application-Level Placement Scheduler) is used for the placement, launch, and monitoring of the applications that make up a job, i.e., the compiled program that the user can execute across one or more compute or GPU nodes. Parallel jobs can spawn several applications at a time, i.e., different programs on one or multiple nodes that can execute concurrently and/or sequentially. Figure 2 shows a commented snippet of a script for the execution of a job in Blue Waters.

Placement of jobs and applications. When a job is submitted to the batch system (i.e., through Torque), it is held by the batch scheduler in the input queue until an ALPS reservation ID is provided, indicating that a suitable set of nodes are available to fulfill the user computing requirement (e.g., XK nodes and a number of CPUs). To this end, the batch scheduler communicates with ALPS through the Batch and Application Scheduler Interface Layer (BASIL).

Once the batch job has been placed in the scheduling queue, the batch system must initialize the job and create an ALPS reservation for the job, i.e., list of nodes assigned by the scheduler to the job in order to ensure resources will remain available throughout the job’s lifetime. When the resources are available, the batch job is executed by a MOM (node manager, which executes commands other than the ones preceded with the “aprun” keywords that are launched on the node part of the reservation assigned to the job). During the execution of a batch job, there could be several calls to aprun to launch applications on the reserved resources. ALPS recognizes when an application’s launch originates from a specific batch job and assigned resources according to the pool of booked resources.

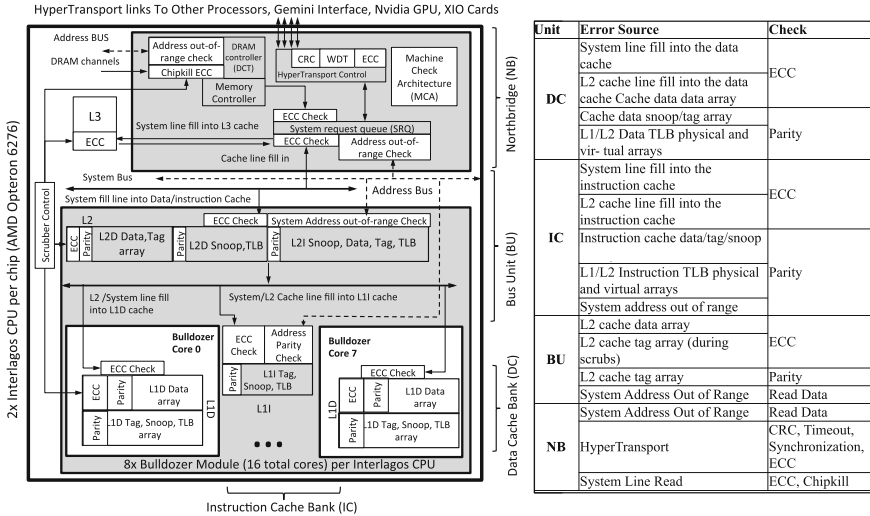
Table 2 Summary of the system specifications and error/failure detection and recovery techniques employed in Blue Waters

Components	Relevant specs	Detection/recovery	Error/failure impact
Node	<ul style="list-style-type: none"> • XE6 (22,640 nodes) • 2x 16 cores, 64 GB RAM 	<ul style="list-style-type: none"> • Node warm-swap • Heartbeat • Coding (ECC/Chipkill/Parity) • Application checkpoint/restart • Node Health Checker • Emergency Power Off (EPO) • Blade Module Controller (BMC) 	<p>Application-level</p> <ul style="list-style-type: none"> • None/not observable • Abnormal termination (e.g., crash) • System-initiated termination (e.g., kill) • User-initiated termination • Application starvation and termination for waittime limit • Application restart/retry/reconnect
	<ul style="list-style-type: none"> • XK7 (4,224 nodes) • 16 cores , 32GB RAM • Nvidia K20X with 6GB RAM 		<p>System-level</p> <ul style="list-style-type: none"> • Node Down • Network Quiesced-reconfigured-resumed • Network Congestion • File System failover • Scheduler Paused • System Wide Outage (total/partial unavailability; service level below threshold) • None
Interconnect	<ul style="list-style-type: none"> • Gemini High-Speed Network (HSN) • LNet (Lustre Network) 	<ul style="list-style-type: none"> • Blade Module Controller *BMC) • Coding (ECC/Parity) • Network reroute • Network Throttling • Retry 	
File system (Lustre)/ storage system (Sonexion 1600)	<p>Lustre</p> <ul style="list-style-type: none"> • Home = 3 PB • Project= 3 PB • Scratch=20 PB <p>Storage nodes</p> <ul style="list-style-type: none"> • Home = 36 nodes • Project = 36 nodes • Scratch = 360 nodes 	<ul style="list-style-type: none"> • Dual redundancy (OST/OSS/ MDS) • multi-level RAID 6 • Lustre client-level assertions • Timeouts • System failover 	
Sys. Software (scheduler, OS...)	<ul style="list-style-type: none"> • Torque/Moab (Job Scheduling) • ALPS (Applications) • CNL (Node OS) 	<ul style="list-style-type: none"> • Dual redundancy • Heartbeat, Timeouts • TCP Connection status • Cray Resiliency Architecture 	

2.1 System Resiliency Features

Table 2 summarizes the resiliency techniques of the principal components of Blue Waters as well as the impact that errors/failures can have on them. Every node in the system is checked and managed by the Hardware Supervisory System (HSS). Core components of the HSS system, highlighted in Fig. 1, are (i) the HSS network; (ii) blade (L0) and cabinet (L1) controllers in charge of monitoring the nodes, replying to heartbeat signal requests, and collecting data on temperature, voltage, power, network performance counters, and run-time software exceptions; and (iii) the HSS manager in charge of collecting node health data and executing the management software. Upon detection of a failure, e.g., a missing heartbeat, the HSS manager triggers failure mitigation operations. They include (i) warm swap of a XE/XK blade to allow the system operator to remove and repair system blades without disrupting the workload; (ii) service node and Lustre node failover mechanisms, e.g., replacement of IO nodes with warm standby replicas; and (iii) link degradation and route reconfiguration to enable routing around failed nodes in the topology.

Nodes are not checkpointed at the system level. Jobs have to be checkpointed by the user based on his/her needs and specifications. Periodic queries to specific software agents are used to monitor jobs and compute node health. In the case of inactivity or a miscommunication by one of the agents, the node state is switched from *up* to *down* by the HSS manager, and the jobs currently running on the node, if not failed, are allowed to complete.



Unit	Error Source	Check
DC	System line fill into the data cache	ECC
	L2 cache line fill into the data cache	
	Cache data snoop/tag array	Parity
IC	L1/L2 Data TLB physical and virtual arrays	ECC
	System line fill into the instruction cache	
	L2 cache line fill into the instruction cache	
BU	Instruction cache data/tag/snoop	Parity
	L1/L2 Instruction TLB physical and virtual arrays	
	System address out of range	ECC
NB	L2 cache data array	Parity
	L2 cache tag array (during scrubs)	
	L2 cache data array	Read Data
NB	System Address Out of Range	Read Data
	HyperTransport	
	System Line Read	CRC, Timeout, Synchronization, ECC

Fig. 3 Machine check architecture for the Opteron 6300 family processor used by Blue Waters

2.1.1 Blade Resiliency Features

All the blades are powered through 4 dual-redundant Cray Verty voltage regulator modules (see Fig. 1), 1 per node, fed by a power distribution unit (PDU) installed in the cabinet and attached to the blade through a dedicated connector. CLE features NodeKARETM (Node Knowledge and Reconfiguration). If a user’s program terminates abnormally, NodeKARE automatically runs diagnostics on all involved compute nodes and removes any unhealthy ones from the compute pool. More details on application-level resiliency features are available in [5].

2.1.2 Processor and Memory Protection

Figure 3 sketches the protection mechanisms enforced in the AMD Opteron processor used in Blue Waters’ nodes. System memory is protected with $\times 8$ Chipkill [6, 7] code that uses 18 8-bit symbols to make a 144-bit ECC word made up of 128 data bits and 16 check bits for each memory word. The $\times 8$ code is a single symbol correcting code, i.e., it detects and corrects up to 8-bit errors. L3, L2, and L1 data caches are protected with ECC, while all other on-chip memory structures (tag caches, TBLs, L2 and L1 instruction caches) are protected with parity. In the case of an uncorrectable error, the operating system is configured to panic. Figure 3 summarizes the main error protection and detection techniques enforced in the machine check architecture of the AMD Opteron 6272 used by Blue Waters nodes.

2.2 File System Resiliency Features

The Lustre file system is accessed through specific IO Routers (LNET routers) bridging the Gemini Network to the InfiniBand network used by Lustre nodes.

Each Lustre rack in the Sonexion cabinets comes with a dual-redundant InfiniBand (IB) switch connecting the Lustre nodes with the LNET routers. Disk modules are organized in RAID 6 and RAID 1 for Object Storage Targets (OST) and Metadata Storage Targets (MDT), respectively. All targets include dual RAID control elements that operate independently, with independent power supplies and cross failover capability. Failures of the RAID arrays are automatically recovered using spare disks without triggering any failover operation.

HSS is responsible for detecting failure of the primary Lustre server node and controlling the failover. Lustre failover requires 2 nodes configured as a failover pair, which must share 1 or more storage devices. For MDT failovers, 2 MDSs are configured to serve the same MDT. Only 1 MDS node can serve an MDT at a time. Two MDSs are configured as an active/passive failover pair. The state is shared using a network Raid 1 technique to ensure short latency between the propagation of state between the primary and standby replica. The primary (active) MDS manages the Lustre system metadata resources. If the primary MDS fails, the secondary (passive) MDS takes over and serves the MDTs and the MGS.

MDS Failover. When clients detect an MDS failure (either by timeouts of in-flight requests or idle-time ping messages), they connect to the new backup MDS and use the Metadata Replay protocol. Metadata Replay is responsible for ensuring that the backup MDS reacquires state resulting from transactions whose effects were made visible to clients but not committed to the disk. Transaction numbers are used to ensure that operations are replayed in the order they were originally performed. All metadata and lock replay must complete before new, non-recovery operations are permitted. Only clients that were connected at the time of MDS failure are permitted to reconnect during the recovery window; this avoids the introduction of state changes that might conflict with what is being replayed by previously connected clients. Recovery fails if the operations of one client directly depend on the operations of another client that failed to participate in recovery (i.e., evicted or failed clients).

OSS Failover. OSTs are configured in a load-balanced, active/active failover configuration. Multiple OSS nodes are configured to be able to access the same OST. However, only 1 OSS node can serve the OST at a time to avoid race conditions. Each OSS serves as the primary node for half the managed OSTs and as a failover node for the remaining OSTs. In this mode, if 1 OSS fails, the other OSS takes over all the failed OSTs. When the OST is in recovery mode, all new client connections are refused until the recovery finishes. The recovery is complete when either all previously connected clients reconnect and their transactions are replayed or a client connection attempt times out. If a connection attempt times out, then all clients waiting to reconnect (and their transactions) are lost. OST failovers can last up to several minutes. The policy enforced in Blue Waters is to declare a failed failover when (i) the failover fails, or (ii) when the recovery time surpasses 30 min.

Client Eviction Failover. Lustre also enforces recovery actions on misbehaving clients by means of client eviction. In general, a server evicts a client when the client fails to respond to a request in a timely manner or fails to ping within 70 s. For example, a client is evicted if it does not acknowledge a glimpse, completion, or blocking lock callback within the timeout. An evicted client must invalidate all locks, which in turn results in all cached inodes becoming invalidated and all cached data being flushed. This enables other clients to acquire locks blocked by the dead client's locks; it also frees resources (file handles, export data) associated with that client. Note that this scenario can be caused by a network partition (i.e., Gemini failures as well as LNET router node failures), as well as an actual client node system failure. Eviction means that all outstanding I/O from a client is lost and unsubmitted changes must be discarded from the buffer cache. Typically, applications do not handle these failures well and exit.

All Lustre failover procedures require power control and management capability to verify that a failed node is shut down before I/O is directed to the failover node. This avoids double mounting the two nodes, and the risk of unrecoverable data corruption. The enforced procedure is called *Shoot The Other Node In The Head* (STONITH). It ensures that the failed node does not prematurely wake up and unexpectedly begin writing to a failed over storage target. The health of MDS and OSS is monitored through the Hardware Supervisory System (HSS) system. If there is a node-failure or service-failure event the HSS starts up the failover process. The status of Lustre automatic failover is recorded in syslog messages.

2.2.1 Network Resiliency Features

Gemini uses error-correcting codes (ECC) to protect major memories and data paths within the device. Fault detection and recovery of the network are managed through the supervisory block that connects Gemini to an embedded control processor (L0) for the blade, which in turn is connected to the Cray Hardware Supervisory System (HSS) network. The firmware on the L0 blade controller detects failed links and power loss to Gemini mezzanine cards. System responses to failures are logged and orchestrated by the System Management Workstation (SMW).

Lane Failover. The availability of 4 lanes in each link allows the network to tolerate up to 2 lane failures and operate in a degraded mode before triggering a system-wide network failover. Each time a lane goes down, an error is written in the logs and a lane recovery (lane failover) is triggered that attempts to recover the lane 10 times in a fixed time window of 60 s.

Link Failover. The failure of all 3 lanes causes the link failure that in turn triggers a link failover. The failover procedure in this case consists of (i) waiting 10 s to aggregate failures; (ii) determining which blade(s) are alive; (iii) quiescing the Gemini network traffic; (iv) asserting a new route in the Gemini chips (performed by the SMW); and (v) cleaning up and resuming Gemini. The total time to execute this procedure is around 30–60 s.

Links can also become unavailable when there is a power loss in a mezzanine, blade, or cabinet and if the cable connection becomes faulty. A faulty cable causes 32 link endpoints to become unavailable. Power loss in a mezzanine, blade, or cabinet causes 64, 64, or 960 end points to fail, respectively.

Rerouting. The algorithm that computes the routing tables fails when it is not able to reroute around failed nodes (e.g., the network is partitioned). In this case, the torus topology is declared unroutable and the system is declared failed (system-wide outage), i.e., the entire system must be halted, fixed, and rebooted. This occurs when the configuration has multiple Gemini router chips disabled.

2.2.2 Job and Application-Level Resiliency Features

Torque and Moab are configured to run in a dual-redundant active/passive configuration. This mechanism enables Torque to continue running even if the scheduler server is brought down. Only the first server to start will complete the full start-up. The second server to start will block very early in the start-up. When the second server cannot obtain the lock, it will spin in a loop and wait for the lock to clear. The sleep time between checks of the lock file is 1 s. In case of failure of the main scheduler server process of Torque, the operating system releases the lock on the shared file and the standby replica takes over. The state of the scheduling queue is managed by an external database that is also replicated using a warm standby replica approach.

The ALPS client provides two main options to cope with system failures at runtime: application relaunch and application reconnect. Application relaunch is a feature that automatically relaunches an application when associated nodes experience certain failures. Relaunch provides the flexibility for users to specify a processing element shrink tolerance so that an application can complete in a possibly degraded mode in spite of compute node failures.

Application reconnect attempts to reconnect the application control fan-out tree around failed nodes and to complete the application run. This option requires the user program to include a specific library and to program well-defined exception handlers. When using this feature, the ALPS client receives HSS (Hardware Supervisory System) compute node failure events and relays the information to the remaining application processes to trigger an application-specific recovery action. Applications that experience these types of failures may be able to complete in a degraded manner.

2.3 Workload

The workload processed by Blue Waters consists of large-scale scientific simulations, including those in the scientific areas shown in Table 3 along with high-level statistics on the workload features. Each area includes different software (code) that is used to create specific studies (applications). Each code is characterized by different features

Table 3 Scientific areas and characteristics of Blue Waters applications

Science area	Runs	Example codes	Struct grid	Unstruct grid	Dense matrix	Sparse matrix	N-body	Monte Carlo	FFT	PIC	Significant IO	Average duration		
												Node hours	Hours	Max nodes
Climate and weather	5,952	CESM, GCRM, CMI/ WRF, HOMME	X	X		X		X			X	1,075	1.63	15,960
Plasmas/ Magnetosphere	18,116	H3D(M), YPIC, OSIRIS, Magtail/ UPIC	X				X		X		X	7,651	2.83	25,670
Stellar Atmospheres, Supernovae	2,336	PPM, MAESTRO, CASTRO, SEDONA, ChaNGa, MS-FLUKSS	X			X	X	X		X	X	332	1.73	22,425
Cosmology	159,320	Enzo, pGADGET	X			X	X					1,332	3.47	22,528
Combustion/ Turbulence	4,422	PSDNS, DJSTUF	X						X			2,143	0.82	16,384
General relativity	69,760	Cactus, Harm3D, LazEV	X			X						510	2.84	11,520
Molecular dynamics	2,693,494	AMBER, Gromacs, NAMD, LAMMPS			X		X		X			131	3.24	17,408
Quantum chemistry	721,042	SIAL, GAMESS, NWChem			X	X	X	X			X	1,117	3.25	20,480
Material science	22,630	NEMOS, OMEN, QMCPACK			X	X	X	X				677	2.09	18,000

(continued)

Table 3 (continued)

Science area	Runs	Example codes	Struct grid	Unstruct grid	Dense matrix	Sparse matrix	N-body	Monte carlo	FFT	PIC	Significant IO	Average duration		
												Nodes hours	Hours	Max nodes
Earthquakes/ Seismology	16,134	AWP-ODC, HERCULES, PLSQR, SPECFEM3D	X	X			X				X	895	1.17	21,296
Quantum chromo dynamics	802,518	Chroma, MILC, USQCD	X		X	X	X		X			1,291	3.05	6,144
Social networks	9,220	EPISIMDEMICS			X							359	0.64	22,528
Engineering/ System of systems	474	GRIPS, Revisit						X				9,465	5.21	5,632
Benchmarks	580,974	linpak, fft, psolve, jacobi, qball, IMB, xperf		X	X	X		X	X	X	X	1,147	1.30	26,864
Visualization	10,374	waves, vmd			X	X					X	11,178	2.86	16,000

The statistics refer to the period from 2013-03-01 to 2014-07-31

summarized in Table 3. Blue Waters jobs may use compute nodes, GPU nodes, or both. The following are representative (for size and duration) of the large-scale applications executed on Blue Waters: (i) NAMD, an application to perform molecular simulations of biomolecules and dynamic evolution of the system with a time step of 1 fs. It is used to determine the precise chemical structure of the HIV virus protein shell on a 64M-atom model, enabling research on new antiretroviral drugs to stop the progression of AIDS. (ii) VPIC, an application for kinetic simulations of magnetic reconnection of high temperature plasmas, executing on 22,528 nodes with 1.25 PFLOPS sustained performance over 2.5 h. (iii) Inertial Confinement Fusion (ICF) involving the simulation of turbulent mixing and combustion in multifluid interfaces, producing 14 TB of data in less than 1 h of execution across 21,417 XE and achieving 1.23 PF of sustained performance. (iv) QMCPACK, an application used to study the high-pressure hydrogen problem. It executes up to 18,000 XE nodes with a sustained performance of 1.037 PF/s for less than 1 h of execution. (v) The largest Weather Research and Forecasting (WRF) simulation ever documented [8].

3 Characterization of Blue Waters Resiliency

3.1 Blue Waters Failures

Blue Waters components are continuously monitored by the HSS. In case of malfunctioning of any component, the HSS starts an alerting process to require the attention of the system administrators. Upon each of those HSS events, system administrators decide whether the event is related to a component or to system failures and act accordingly. In particular, if the event notifies of a failure, the system administrator is required to complete a failure report describing the event.

3.1.1 Failure Data

System failure reports are human-written documents that record each event in the system that required the attention of the system managers. These include failures, maintenance, system upgrades, and bugs reported to the manufacturer. After fixing the problem, the staff updates the error report with the identified cause of the event, the fixes that were applied, and where applicable, the field unit that was replaced. To minimize misdiagnosis, failures and System-Wide Outages (SWOs) are only reported after positive acknowledgment by both Cray and NCSA personnel. Failure reports include the fields reported in Table 4.

The Blue Waters maintenance specialists classify each entry added to the failure report using the categories below:

Table 4 Fields of Blue Waters failure reports

S/N	Numerical id indicating the affected component (e.g., file system or blades)
Component serial number	Numerical id to identify to the specific hardware component related to the recorded event (e.g., one specific blade)
BW#	Numerical id indicating the Blue Water event (e.g., system boot, maintenance time) when the entry was reported
CLSTR#	Unique numerical id indicating the specific maintenance operation performed to resolve the event logged in the report
Failure cause area	Area of impact of the event (e.g., processing blade, storage nodes)
Description	Textual description of the event
SFDC#	Unique numerical id of the event
BUG#	Id of the bug causing the failure (where applicable)
Failure type	Classification of the failure event in specific failure categories
#Compute	Number of computing resource affected by the event
Date start/fixed	Start/fixed date of the event

- **Failure (No Interrupt)**: A failure that is naturally tolerated by the architecture, i.e., does not cause node/system downtime or trigger failover actions (e.g., cooling hardware or performance problems).
- **Interrupt (Failover)**: A critical problem that is successfully handled by the automatic failover mechanisms.
- **Link and Node Failure (Job Failed)**: A failure of one or more nodes and one or more links that causes the failure of one or more user jobs.
- **Link Failure (No Job Failed)**: A failure of a single or multiple node(s) that causes a link failure that is successfully handled by the automatic network failover mechanisms;
- **Link Failure (Job Failed)**: A link failure that causes job loss.
- **Single/Multiple Node Failure**: A failure of single or multiple node(s) that requires repair but does not impact core system functionalities.
- **Interruption (System-Wide Outage)**: A failure that makes the whole system unavailable, e.g., because of a system-wide repair or restart. A system-wide outage occurs if a specific requirement cannot be met, such as (i) the ability to access all data blocks of all files in the file system; (ii) user ability to log in; (iii) full interconnection between nodes; (iv) access to external storage server (esDM, or external data mover); (v) ability to support a user application's submission, scheduling, launch, and/or completion; (vi) ability to recover from file system or network failures through automated failover operations; or (vii) performance (e.g., network bandwidth or file system throughput) above acceptable levels.

3.1.2 Failure Characterization Methodology

Human-generated reports present several challenges. They contain textual descriptions in natural language and cannot be readily analyzed by automatic tools. Failure reports must be filtered from reports on non-failure events to avoid biasing the analysis. Therefore, first steps in the analysis include (i) purging the non-failure entries, (ii) reorganizing the content into a structured database, and (iii) reviewing failure events to remove redundant or overlapping records (in partnership with NCSA and Cray engineers). The characterization process then consists of performing the following analysis: (i) associating failures with their corresponding root cause categories (i.e., hardware, software, network, environment, heartbeat/node down, and unknown), depending on the repaired element or corrective action, (ii) analysis of failure rates and repair times across the root causes; (iii) measurement of how the hardware and software failure rates evolve in the considered measurement window; (iv) evaluation of the hardware error resiliency; and (v) evaluation of system-wide failure and repair time distributions and their statistical properties.

Assignment of Failure Categories. The assignment of an outage to the hardware or software category is guided by the type of corrective action (e.g., replacement using a spare part, or installation of software patches) and based on interactions with system administrators. After assigning categories, we reduced the number of reports from 1,978 to 1,490 entries. These entries include only failure events identified from the reports classified into the following exclusive categories: hardware, software, missing heartbeat, network, environment, and unknown. “Unknown” applies are those failures for which the cause could not be determined; they are considered a separate category, detected by the system but automatically recovered from before they can be diagnosed.

3.2 Blue Waters Errors

Usually, logs contain a large amount of redundant and irrelevant information in various formats. Thus, data filtering, manipulation, and processing must be performed to classify this information and to put it into a flat format that facilitates subsequent analyses. This section provides an overview on how we employed advanced log analysis techniques to analyze the error resiliency of Blue Waters.

3.2.1 Error Data

Errors are captured by many detectors that store information on various automated logs. Examples include system-generated syslogs, machine check logs, and workload logs, including job and application scheduler logs.

System logs include system events logged by the OS and by HSS as well as entries generated by the Sonexion cluster. Events collected in these logs include (i) the time

stamp of the event, (ii) the facility, indicating the type of software that generated the messages, (iii) a severity level, indicating how severe the logged event is, (iv) identification of the node generating the message, (v) the process, including the PID (process identifier) of the process logging the event, and (vi) the event description. *Machine check logs* keep track of errors detected by the machine check architecture in the north bridge of the processor. Machine check data include a time stamp, the ID of the node that experienced the machine check, and information on the type (e.g., correctable/uncorrectable or detected by the hardware scrubber), physical address, error syndrome, and operation involved (e.g., reading from the memory or loading the L2 data cache) encoded into a 64-bit word in the logged data obtained after fetching the content machine check status register [6]. *Torque logs* include information on created, canceled, scheduled, and executed jobs in the system. Each entry in a Torque log consists of 45 fields containing time information on all the phases of the job (creation, queue, execution, and termination times), user, group, queue, resources, the type and the list of used nodes, and wall-time used. *ALPS logs* include information on node reservation management, job/application launching, periodic monitoring and termination of user applications, and detected internal problems and cleanup operations. ALPS logs are redirected by the system console to the syslogs and merged with other system events.

3.2.2 Error Characterization Methodology: The *LogDiver* Approach

The analysis of automated logs requires a mix of empirical and analytical techniques: (i) to handle the large amount of textual data; (ii) to decode specific types of system events and exit statuses obtained from multiple data sources; (iii) to match different sources of information, such as workload and error data to extract signals of interest (e.g., error rates); and (iv) to measure error propagation and application resiliency. *LogDiver* is a tool created to measure both system and application-level resiliency of extreme-scale machines in a holistic manner using different logs. *LogDiver*-based analysis allows us to create a unique dataset encapsulating events that are central in (i) performing resiliency and performability measurements, (ii) supporting the application of machine learning techniques to create application-level error detectors, and (iii) measuring how multiple factors (e.g., application scale and system errors) impact applications. Specifically, this tool does the following:

- Allows a precise identification of the reasons behind application termination,
- Directly relates system errors and failures (e.g., Gemini ECC errors, GPU MMU errors, and Lustre file system failures) to application failures, and
- Provides a unified representation of the workload/error/ failure logs, permitting workload-failure analysis and computation of a range of quantitative performance and dependability metrics.

Such in-depth understanding of application error sensitivity is essential to achieving realistic performance evaluation of current systems and to guiding design of

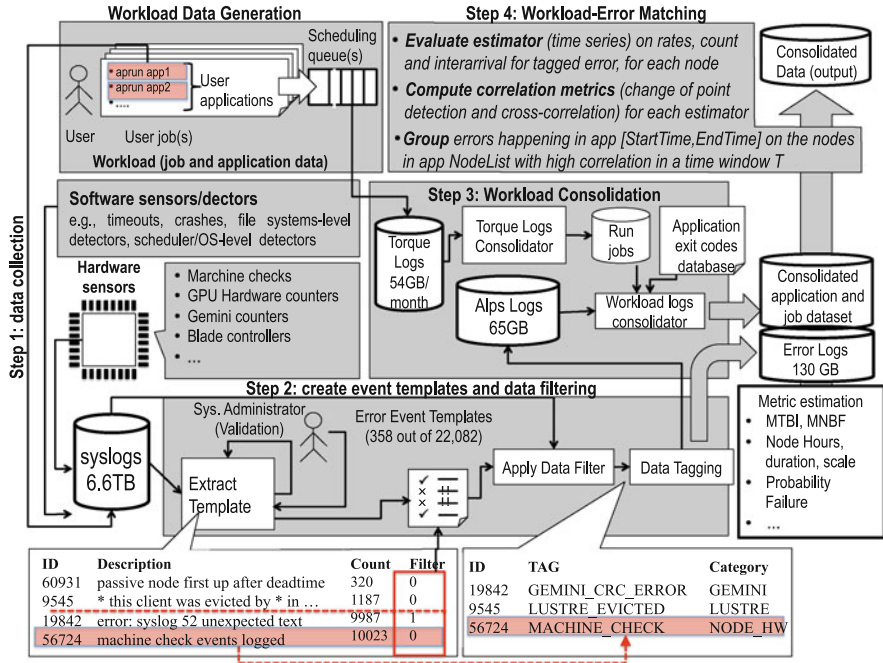


Fig. 4 The LogDiver workflow

resiliency mechanisms. In the following, we briefly describe the *LogDiver* workflow.

LogDiver operates in four main steps, depicted in Fig.4 and summarized in Table 5. Each step produces several output files that are fed downstream to the subsequent step. Data in intermediate output files can also be used by external tools (e.g., MATLAB and SAS to perform workload characterization) to conduct additional analyses beyond what *LogDiver* supports.

All the *LogDiver* steps are fully automated except one, the second step, which aims to understand the content of the logs by generating a list of log message templates. The categorization of error templates requires frequent interactions with technical personnel for validation purposes. The categorization consists of assigning a specific unique numerical template ID, tag, category, and group to each error template. The tag is a textual description of the event of interest, e.g., GPU_MMU_ERROR or LUSTRE_EVICT. The category refers to the subsystem generating the event, e.g., NVIDIA_GPU or LUSTRE. The group corresponds to the subsystem involved in the event, e.g., NODE_HW or STORAGE. For example, in the bottom left corner of Fig.4, we assigned the tag CPU_MACHINE_CHECK and category NODE_HW to the template with ID 56724. The details of the generated list of error templates are reported in [5, 9].

Table 5 Description of the steps of the LogDiver workflow

Step	Objective	Input	Output
Data collection	To collect data from multiple sources including a subset of the data generated by the various hardware sensors deployed in Blue Waters	syslogs, Torque logs, ALPS logs	Parsed logs: data transformed to an internal format that is system agnostic
Event tagging and filtering	To identify, categorize, and filter the error events contained in the collected data	Parsed Logs	(i) a list of categories containing only events of interest, i.e., the error data, referred as message template; (ii) the filtered dataset (error dataset)
Workload consolidation	To create a consolidated dataset that includes information on jobs, application runs, used resources (e.g., type and ID of used nodes), user options (e.g., used resiliency features), in order to enable the matching of workload data with error data, performed in the next step	Parsed Torque and Alps logs	Extended data set of user applications (referred to as “application data” in Fig. 11), which include 46 fields for each application. Important fields are (i) start and end time, (ii) reservation ID, job ID, user, group, application name, (iii) resources data, e.g., number, ID and type of nodes, memory, and virtual memory, (iv) application exit code and job exit code, (v) job- and application required and used wall time, and (vi) command used to launch the application
Workload–Error Matching	To match and collate relevant error data with the consolidated workload data	Filtered error events and application data	(i) <i>apid2Error</i> dataset (Fig. 11a) where each application run by Blue Waters is paired with all the errors that the application experienced while executing; (ii) <i>Coalesced Errors</i> dataset, where all the errors occurring with high correlation on nodes executing the same instance of application (including the service nodes serving XE and XK nodes related to the application) are grouped together to form error tuples

Error-Application Matching. The error-application association is performed by overlapping the workload data with errors occurring between the start time and the end time of the application considered on one of the nodes executing the application or on one of the service/IO nodes serving the application. To correlate application failures with error data, *LogDiver* uses a mix of empirical and analytical techniques that can be classified into two categories: (i) correlation analysis to separate local from global effects across events generated by different nodes and/or different error categories, and (ii) event coalescence to group specific errors occurring with high correlation. The first operation executes a change point detection analysis [10] to determine changes in the mean and variance of different estimators that *LogDiver* evaluates from the error data. Given that software and hardware sensors generate events using different sampling periods and applying different generation rules (e.g., periodic or impromptu), estimators are computed on a uniform representation of the data in the form of (stochastic) point processes. A set of events is generated at random points in time T_i using different representations and transformations based on both the event inter-arrival process and the count of events during specific time intervals. Finally we group together all the error tags that occur with statistically high correlation (estimated using Pearson's lagged cross-correlation coefficients among estimators) and that are generated by the nodes executing the same application. The organization of the output of the workload-error matching is shown in Fig. 7a.

Example of correlation analysis. To determine the symptoms of system-wide outages, we conduct a preliminary analysis based on a cross-correlation analysis to study relations among errors captured by Blue Waters sensors and stored in the system logs.

The example in Fig. 5 is limited to a few sensor signals monitored over about one month, but it is representative of longer and more detailed studies. The top four graphs present: the aggregated rates of error events logged per hour reported by hardware/software sensors corresponding to GPU errors (i.e., GPU double-bit errors and MMU errors), nodes down (at least one of the computing nodes is down because of a failure), HSN Gemini errors (network or communication-related errors, e.g., a corrupted routing table or hung routes), and machine check exceptions (errors captured by hardware mechanisms embedded in the microprocessor, e.g., uncorrectable memory errors). The bottom two graphs in Fig. 5 plot the percentage of application failures (system-caused application failures per hour divided by the number of applications running in the hour) for Blue Waters XE6 and XK7 nodes.

Because the system failures are consequences of the errors reported by the software and/or hardware sensors, we expect to find a correlation between the sensor readings and the observed system failures. Figure 6 shows clear indications (observable visually) of such dependencies. The shaded regions depict examples of dependencies among sensor readings from different subsystems.

- Cases a, g, and i: Multiple node failures (i.e., nodes were down) led to a burst of network (Gemini and Lustre network) errors; nodes detected the interconnect problems; the interconnect was quiescent and then reconfigured; and the system-level failover kicked in, attempting to prevent application failures. However, only

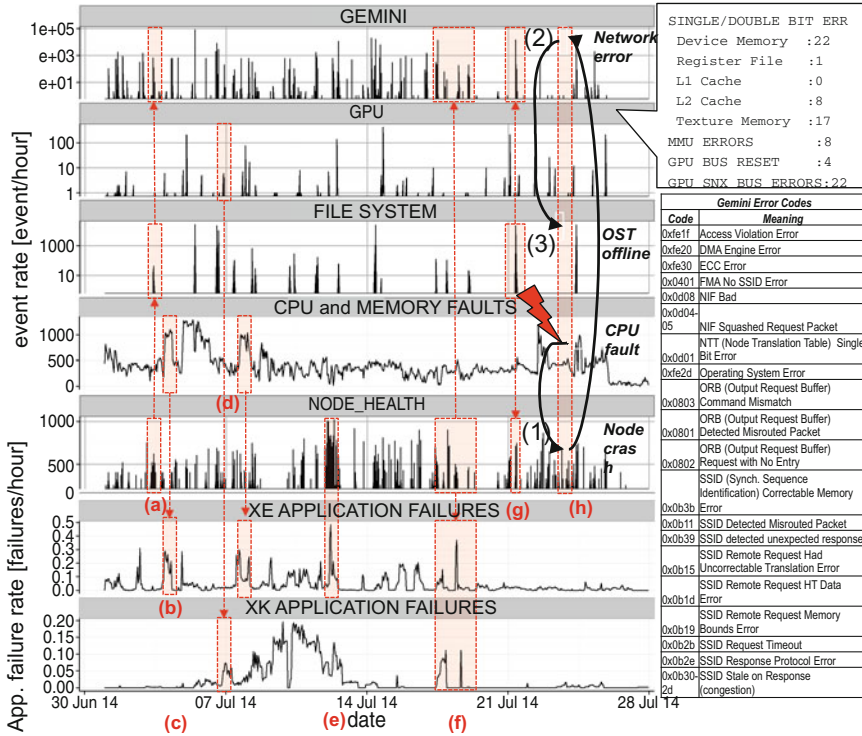


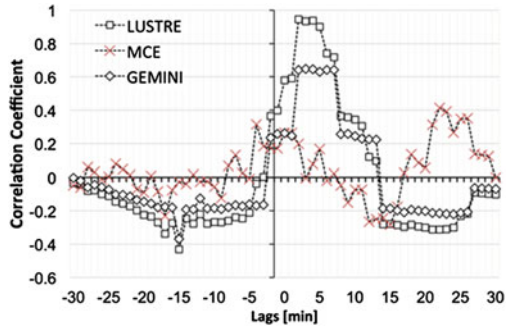
Fig. 5 Examples of potential fault propagation among signals and applications

in Case i was the failover successful in stopping application failures. In Cases a and g, the failover mechanism was unable to prevent failures of multiple applications.

- Cases b and d: A burst of machine check exceptions coincided with malfunctioning of the voltage regulator module powering a node. (The cause of the exception was revealed by an in-depth analysis of the failure.) As a consequence, many applications failed.
- Case c: A burst of GPU errors occurred, e.g., a bus reset and a double-bit error led to failures of multiple applications executing on XK and XE nodes, respectively.

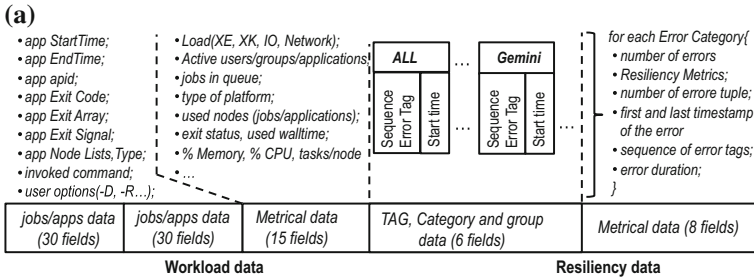
Capturing fault propagation and signal cross-correlation. Figure 6 shows the value of Pearson’s cross-correlation function, computed minute by minute for Case i in Fig. 5. Case i consisted of a node’s failing because of an uncorrectable machine check, causing a Gemini topology reconfiguration and, eventually, a burst of application failures because of file system unavailability. The cross-correlation is computed between the rate of application failures per minute and the rate of error events generated from the sensors considered. The example suggests that *it is possible to identify windows of high correlation between the signals looking at specific patterns in the sensor streams.*

Fig. 6 Cross-correlation analysis among signals for case i in Fig. 5



3.2.3 Event Coalescence

LogDiver employs different coalescence techniques, making it possible to perform analyses at different levels of detail. Specifically, it can coalesce errors generated by (i) the same error category/tag, (ii) the same node, (iii) nodes allocated to the same job and/or application, and (iv) the whole system, considering only console logs. The last type of technique employs hypothesis testing and domain expertise to avoid grouping independent events (e.g., two ECC memory errors on two different nodes). We use domain expertise to create an adjacency matrix of signals that can be mutually influenced, e.g., GPU_MMU errors with GPU voltage level. We group the events (i) temporally when they show high (lagged) cross-correlation values, and (ii) spatially (i.e., events generated across different nodes, blades, and cabinets) only



Size dataset: 5,116,766 (Num. application runs) x 213 fields [6GB]

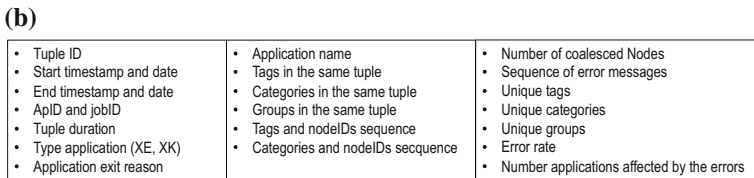


Fig. 7 Organization of the *LogDiver*'s main output

when they are generated by nodes executing the same application. The organization of the output of the event coalescence is shown in Fig. 7b.

3.2.4 *LogDiver* Resiliency Metrics

The last part of the tool is in charge of estimating various metrics of interest. These include: Mean Node Hours Between Failures (MNBF) computed as a ratio between the total number of production node hours and the total number of application failures; Mean Time Between Interrupt (MTBI) computed as a ratio between the total number of production hours and the total number of application failures; probability of an application failure; load figures such as system load, rate of active users/groups/applications, and cumulative number of node hours per user; and application and job success/failure rate. All these metrics are computed with respect to (i) the whole system, (ii) application names, (iii) the user ID, (iv) the node type (i.e., xe and xk nodes), (v) node hours, and (vi) the application/job scale. For each of the metrics mentioned, *LogDiver* estimates both empirical distributions (cumulative and density functions) and synthetic statistics, including mean, standard deviation, and confidence intervals.

4 Analysis of Blue Waters Failure Causes

In this section, we analyze the data and information in the failure reports, addressing how often Blue Waters fails and how much time is needed to fix the causes of a failure. We report on (i) the distribution of the failures' root causes across all the failure categories defined in Sect. 3, and (ii) the root causes of the failures, as identified by the Cray and Blue Waters maintenance specialists. This analysis is based on analysis of manual reports generated by Blue Waters system maintenance engineers over a period of 261 days, from January 3, 2013 to November 17, 2013. Over 1,978 distinct entries (including scheduled maintenance, system expansions, and failure events) were reported during this time.

4.1 *Distribution of Failure Types*

Table 6 provides a breakdown of the failure reports, MTBF, and MTTR across the defined failure categories. In the measured period of 261 days, the system experienced 1,490 failures, of which 1,451 (97.4%) were tolerated and 39 (2.6%) resulted in system-wide outages. Key observations are:

- On average, there was 1 failure (across all categories) every 4.2 h, while the system suffered system-wide outages approximately every 160 h.

- 58.3 % of failures resulted in single/multiple node failures (category 6 in Table II) that caused node unavailability. Such failures were the most common, occurring every 4.2 h; they were allowed to accumulate in order to optimize the cost of field intervention, and they were repaired, on average, in 32 h.
- About one-fourth (25.7 %, categories 2 and 4) of failures were potentially severe events from which the system recovered by means of system-level failover (see Sect. 2) without job failures. Only 2.6 % caused job failures (categories 3 and 5) without resulting in an SWO.
- 11 % of failures (category 1) consisted of noncritical events that were tolerated by the redundant design of Blue Waters without requiring any automatic failover operation. Such events included, for instance, failures of (redundant) power supplies in the blades, failures of cooling hardware (fan trays or water cooling valves), job scheduling performance problems, and resource manager crashes. Such failures caused no machine downtime and little or no unavailability of software services (e.g., the job scheduler). They occurred on average every 35.12 h, and their inter-arrival times showed high variance due to the heterogeneity of the root cause.

4.1.1 Software Related Failures

Software failures contributed to 53 % of the node downtime hours. Figure 8 shows how hardware, software, network, heartbeat, and environment root causes are distributed across the failure categories given in Table 6. As seen in other systems [11], failures with hardware root causes were the predominant cause of single/multiple node failures. They occurred 442 (51 %) times over 868 single/multiple node failures documented in the failure reports, and they constituted 42 % of the total number of failures across all the categories (rightmost bar in Fig. 8). Conversely, failures with software root causes represented 20 % of the total number of failures and only 7.9 %

Table 6 Failure statistics

Failure category	Count	%	MTBF (h)	MTTR (h)	σ_{TBF} (h)	σ_{TTR} (h)
(1) Failure (No interrupt)	164	11	35.17	13.5	70.8	35.3
(2) Interrupt (Failover)	99	6.6	58	14.7	92	42.2
(3) Link and Node failure (Job failed)	19	1.3	297.7	6.1	427.3	5.4
(4) Link failure (No job failed)	285	19.1	19.9	32.7	51.9	91.2
(5) Link failure (Job failed)	19	1.3	291.6	16	444	26.7
(6) Single/Multiple node failure	868	58.2	8.7	26.7	6.3	72
(7) Interruption (system-wide outage)	39	2.62	159.2	5.16	174.2	8.1
All	1490	100	8.8	34.5	13.3	50.5

The last row refers to the statistics calculated across all the failure categories

of the single/multiple node failures. However, an interesting conclusion can be drawn from the relative impact that hardware and software causes have on the total number of node repair hours (hours required to repair failures due to the same root cause, multiplied by the number of nodes involved in the failure) shown in Fig. 9. The key observation is that failures with software root causes were responsible for 53 % of the total node repair hours, although they constituted only 20 % of the total number of failures. Hardware root causes, however, despite causing 42 % of all failures, resulted in only 23 % of the total repair time. As we shall see, hardware problems are well managed by the Cray architecture.

To identify a reason for those differences, we analyzed the distribution of the number of nodes involved in failures with hardware or software root causes. We found that failures with hardware root causes that did not cause system-wide outages propagated outside the boundary of a single blade in only 0.7 % of the cases. Cases in which failures caused by hardware impacted a full blade involved failures in the voltage converter module (VRM) of the mezzanine and/or problems with the cabinet controller. Data show that failures with hardware root causes were limited to a single node or a single blade (i.e., 4 nodes) 96.7 and 99.3 % of the times they occurred, respectively. Conversely, software failures, if they did not cause a system-wide failure, propagated to more than 1 node in 14 % of the cases, i.e., 20 times more often than those caused by hardware. In addition, hardware is easier to diagnose than software. Hardware failures are fixed in bulk to reduce the cost of field intervention, e.g., after a given number of node failures. Although this does not impact the system MTTR (5.16 h), it does result in a larger MTTR for single nodes (32.7 h), as reported in Table 6.

4.1.2 Lustre Failures

Lustre, the CLE OS, and Sonexion/storage software are the top three software root causes among the failure categories. In particular, the Lustre file system and related

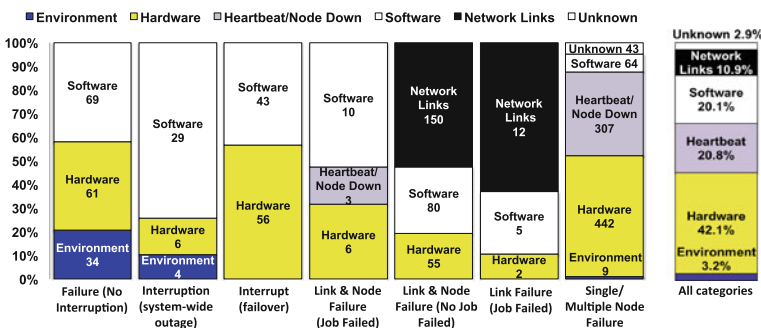


Fig. 8 Breakdown of the failure categories across the failure types considered

Fig. 9 Breakdown of the downtime hours across the failure types considered

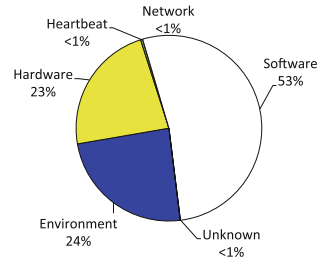
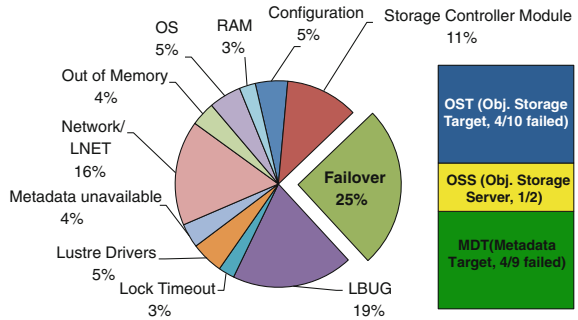


Fig. 10 Breakdown of the Lustre failure root causes



software caused 44 % of the single/multiple node failures attributed to software root causes, while the CLE OS and Sonexion/Storage caused as much as 28 % and 9 % of the total single/multiple node failures, respectively. Lustre includes a rich software stack of more than 250k lines of code and plays a crucial role in Blue Waters, since all the compute nodes are diskless and rely on Lustre and Gemini for the I/O.

Figure 10 shows the breakdown of the root causes of Lustre failures. Blue Waters experienced a total of 104 different failures attributed to Lustre. The failure of any Lustre component triggered a failover operation that was successful 75 % of the time they were invoked. If the automated failover procedures are unable to recover the normal state of the system, the technical staff managing Blue Waters overrides the automated procedures and the system is manually recovered. A common cause of Lustre issues is detection of the so-called LBUG (19 %), i.e., a panic-style assertion in the storage node kernel. The unavailability of the Object Storage Target (OST) or of the metadata (i.e., MDS, MDT, and MGS in Lustre) that provide low-level support for the storage system is another important Lustre failure category. Configuration problems contributed to 5 % of the Lustre failures, but they were not critical and often manifested as performance issues, i.e., Lustre became slow. About 14 % of Lustre failures were due to hardware problems (RAM, 3 %, and Controller Module, 11 %), and the failover mechanism recovered from all of them.

Table 7 System-wide outage statistics

Availability	0.9688
Total time	261 days
Unscheduled downtime	8.375 days
MTBF (SWO)	6.625 days
MIN TBF	3.35 h
MAX TBF	37 days
MTTR	5.12 h
Min TTR	0.58 h
Max TTR	12 h

4.2 System-Wide Outages

Table 7 shows the statistics for 39 system-wide outages (Interruption failure category in Table 6) presented in the failure reports. On average, the system experienced a system-wide outage every 159.22 h. In 90 % of those cases, the system was brought back to full capacity within 3.28 h from the start of the outage, with an MTTR of 5.12 h. The quick repair time contributed to a system availability of 0.9688.

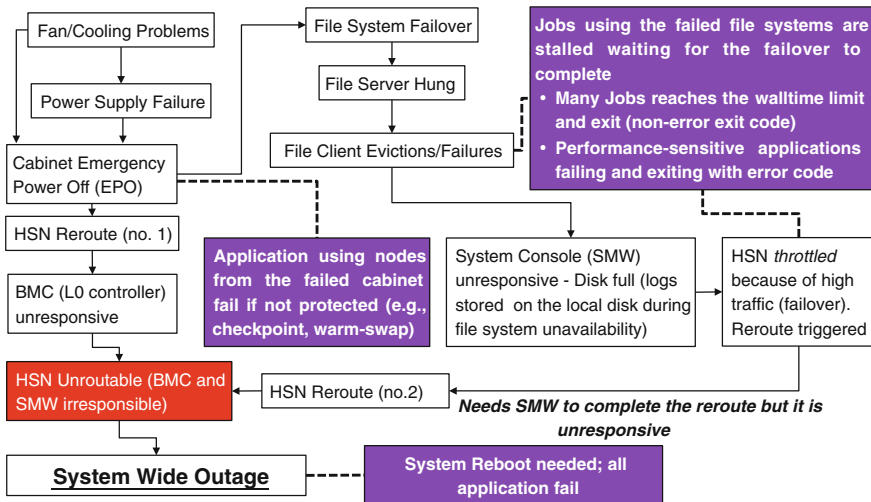


Fig. 11 Example of system-wide outage event and impact on applications

4.2.1 Example of System-Wide Outage and Related Errors

Figure 11 shows a real failure scenario in Blue Waters, highlighting the impact on running applications. The chain of events starts with a voltage regulator failure due to bad cooling, which leads to a cabinet Emergency Power Off (EPO). As a result, the subsequent change of the topology is detected by the HSS, which triggers a network failover that includes a reroute of the HSN Gemini network. However, the network failover is only partially successful and leaves the Blade Module Controllers (BMCs) in an inconsistent state, one in which another HSN reroute would result in a system-wide outage. At this stage, all applications executing on the failed blade are either failed or recovered, depending on the effectiveness of the protection mechanisms (e.g., application-level checkpoint) as well as on the sensitivity of the run code to transient network shutdowns.

In this example, the cabinet EPO also impacts the Lustre file system, causing a mass client disconnect from file servers, which in turn results in file server instability and consequent file system failover (started just after the network reroute). File system slowness (e.g., due to failover and drive rebuild) causes slowness in the job scheduler, impacting application productivity. Applications accessing the file system are suspended waiting for the failover to conclude. Recovery from client failure in Lustre is based on lock revocation and other resources, so surviving clients can continue their work uninterrupted. As a consequence, many applications are quiesced and put on wait for the lock acquisition by the Distributed Lock Manager managing the failover process.

Further, we observe that accumulation of hung I/O threads on the file server from disconnected clients can eventually cause the file server to crash. These cascading failures ultimately result in a nonresponsive System Console and HSN throttle with the following chain of events. The unavailability of the file server causes a surge of logs containing Lustre file system errors to be redirected to the local file system of the system management workstation (SMW). This fills up all the available storage, causing the SMW to be unresponsive and triggering an SMW replica swap. The high network traffic generated by the file system failover and by the variety of anomalous conditions detected causes HSN congestion that reacts by first throttling and then by forcing a reroute. However, the HSN reroute can only be completed with at least one available SMW (not yet replaced by the replica), hence the system-wide outage. In the end, the system in the example was rebooted, hence all running applications were terminated. In summary, the scenario described shows how a local blade/cabinet failure could not be contained and led to a file system failover that propagated across the entire system.

4.2.2 SWO Measurements

Figure 12 shows the PDF and CDF of the distribution of the times between SWOs and the SWO repair times. It is interesting to note that the statistical fitting of the time between SWO can be modeled as indicated in Table 8. The exponential, Weibull, and

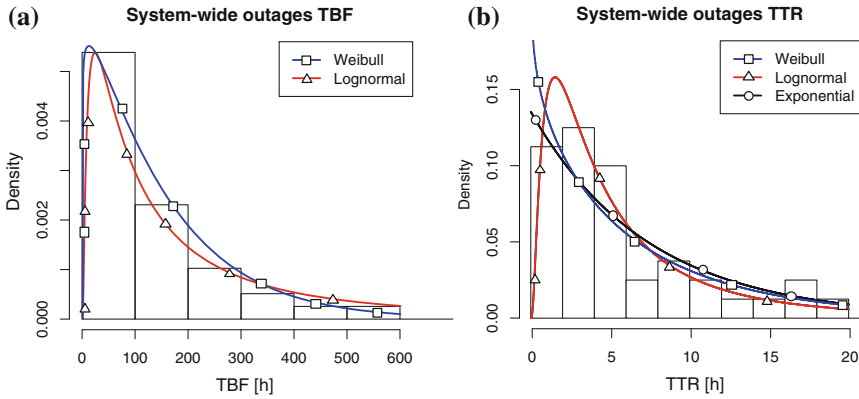


Fig. 12 Distribution fitting for the time between SWOs. **a** PDF, **b** CDF

Table 8 Parameters estimated for the distributions of the time between SWO in Fig. 12a

Distribution	G.O.F. (Kolmogorov)	Parameters
Lognormal	0.072	$\mu = 4.58, \sigma = 1.08$
Weibull	0.073	$\alpha = 1.07, \beta = 152$
Exponential	0.08	$\lambda = 0.006$

Table 9 AFR, MTBF and FIT for processor, memory DIMM, GPU, and storage devices

	Total	AFR (%)	MTBF	FIT/Device	FIT/GB
Processor	49,258	0.23	3,771,492	265.15	NA
DIMMs	197,032	0.112	7,821,488	127.84	15.98
GPU card [6 GB]	3072	1.732	506,394	1974.11	329.02
Disks [2 TB]	20196	0.312	2,807,692	356.16	0.174
SSD [2 TB]	392	0.717	1,230,795	812.48	0.397

lognormal distributions obtain an acceptable goodness of fit (p-value less than 0.1), as reported in Table 9. Note that the good fit for the lognormal distribution indicates a hazard rate that first increases and then decreases ($\alpha > 1$), modeling the case in which a system-wide outage might depend on the preceding one. Interestingly, the reports document only one case: a system-wide outage was attributed to the same cause as the preceding one, which had occurred 3 h 51 min earlier (both outages were associated with Lustre). The other cases were clearly unrelated. In addition, after each system-wide repair, the system is rebooted and cleared of remaining problems. These factors contribute to the good fit of the exponential distribution in Fig. 12.

Figure 13a shows the breakdown of the system-wide outages across the three categories of causes: hardware, software, and environment. About 74.4% (29 out of 39) of the system-wide outages (SWOs) had software causes. Hardware contributed

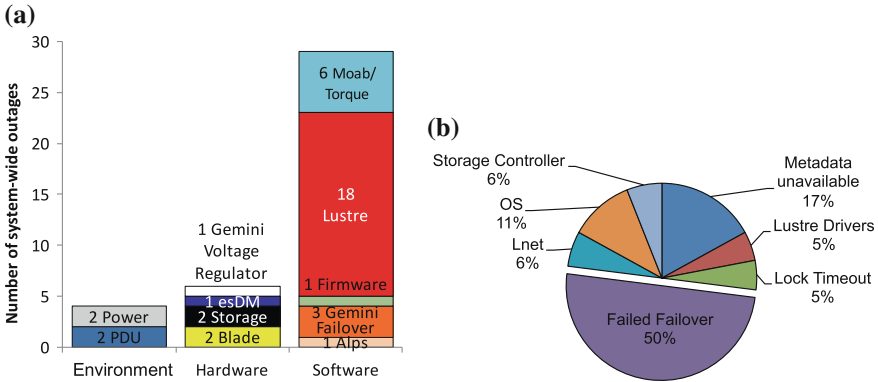


Fig. 13 Breakdown of the SWO root causes and repair times (a), Lustre failures cascading in SWO (b)

to 15.4% of system-wide outages (6 out of 39), and environmental issues (power failures in our case) caused 10% of them (4 out of 39). Failures with network root causes brought down the system in two cases or 0.6% of the recorded network failures. However, as we will discuss in the next section, those failures occurred when the network became partitioned and was hence unable to route around failed nodes due to a lack of redundant paths.

Impact of Lustre Failures. 46% of SWOs were caused by Lustre failures (18 out of 39, about 62% of all software related SWOs). Figure 13b shows the types of Lustre failures that can escalate to system-wide outages. The key observations are (i) about 20% of Lustre failures (18 out of 104) cascade and manifest as system-wide outages, and (ii) for about half (47%) of them, a failure of the failover can be identified as a direct reason for the SWO.

Impact of single node failures. About 0.7% (6 out of 831 cases) of single-node failures led to SWOs because of an inability to reroute network traffic around the failed blades. Data show that node failures due to hardware problems become critical when they affect the Gemini routing or access to the Lustre file system. The full 3D torus topology is effective in providing a number of redundant paths, and the link degradation mechanisms and distributed routing tables ensure protection against transient errors, e.g., corruption of the routing table. The Gemini network uses dimension-ordered routing. In an error-free network, a packet first travels along the X dimension until it reaches the destination node’s X coordinate. The packet then travels in the Y dimension to the destination node’s Y coordinate. Finally, the packet moves in the Z dimension until it reaches the final destination. If the standard X-Y-Z ordering does not work, Gemini tries an alternative route until it finds a workable path.

Certain failure patterns, such as multiple errors in the same loop along the Z dimension, can leave Gemini unable to find a valid route. While such failures are rare, they require the longest time to recover from, with an MTTR of 8.1 h. The

failure reports contain three cases of system-wide outages due to routing problems: (i) a mishap of the failover procedure during the warm swap of a failed service causing a system partition; (ii) a partial failure of the Gemini network that did not trigger the link recovery procedure, causing all the jobs to abort because of lack of communication; and (iii) a crash of the Gemini routing algorithm that caused the routes to hang. In the last case, while the Gemini crash's cause is unknown, system logs revealed a high rate of misrouted packets and routing data corruption within a 1 h time window before the incident.

4.3 *Hardware-Related Failures*

The procedure enforced at NCSA for Blue Waters hardware replacement is to replace (i) the processor when uncorrectable or parity errors are observed, and (ii) memory when the rate of corrected ECC errors over a single address is above a programmed threshold. A similar policy is replacement of storage devices and GPU accelerators when uncorrectable errors are detected, e.g., when a raid controller detects uncorrectable disk errors or when an NVidia GPU accelerator manifests a double-bit error. In fact, GPU accelerator memory is protected only by ECC and therefore is vulnerable to multiple-bit errors.

Table 9 reports the annualized failure rate (AFR, which is the percentage of failed units in a population, scaled to a per-year estimation), MTBF per device, and FIT rate for processor, memory DIMM, GPU, and storage devices (including disks and SSDs). Table 9 also gives the estimated failure rate expressed in FITs/device. The MTBF for a single component is computed as the total number of working hours divided by the AFR. Interestingly, the DDR3 DIMMs show the highest value of MTBF, with 7,821,488 h. The processors and the disks show a figure for the MTBF of about half the DDR3 DIMM MTBF, specifically 3,771,492 h for processor and 2,807,692 for the disks. The GPU accelerators show an MTBF of 506,394 h, i.e., 15 times smaller than the DDR3 DIMM MTBF (about 200 times smaller if comparing the FIT/GB). In fact, the disks provided a high level of reliability. During the measured 261 days, only 45 disks were replaced from the pool of 20,196 devices. The computed AFR for disks is lower than the observed values of 2–6 % given in other studies of disk failures [12], although the population of disks in Blue Waters is smaller than that considered in other studies. Our numbers, however, confirm the MTBF values provided by the manufacturer and show no tangible evidence of defective disk units; we measured a SSD MTBF lower than the manufacturer's declared value of 2,000,000 h.

5 **Blue Waters Hardware Errors**

Blue Waters maintenance specialists diagnose processor and memory related problems by looking at the machine check exceptions contained in the system logs. In

looking at the system logs produced by Blue Waters nodes, we counted 1,544,398 machine check events in the measurement period (i.e., on average, a rate of 250 errors/h), of which only 28 consisted of uncorrectable errors, i.e., errors that cannot be corrected by either ECC or Chipkill and may cause loss of data, corruption of processor state, or both. This indicates an unusual degree of containment of hardware problems that we further investigate in this section. Table 10 shows the breakdown of the machine check errors over the different node types.

In total, 12,721 nodes (46 % of the total Blue Waters nodes) experienced at least 1 memory error; 82.3 % of the nodes that manifested machine checks were compute nodes; 19.4 % were GPU nodes, and 7.34 % were service nodes. 6.6 % of all the memory DIMMs generated at least 1 correctable error during the observation window. In our data, 55 % of the nodes generated only 1 machine check, while 92 % of the machine checks were generated by only 19 % of the nodes.

5.1 Processor and Memory Errors

Table 11 shows a breakdown of the memory errors in Blue Waters. The table shows that about 70.01 % of the memory errors involved a single bit and that 29.98 % involved 2–8 consecutive bits, similar to the results in [13] for a smaller scale Cray supercomputer. The Chipkill can correct errors affecting up to 2 symbols ($\times 8$ Chipkill on compute and GPU nodes), and without it, 30 % of the memory errors analyzed would be uncorrectable (e.g., if only ECC were used). Hence, a key finding is that ECC/Chipkill techniques were effective in correcting 99.997 % of the memory errors that occurred, i.e., we observed only 28 uncorrectable errors out of 1,544,398 errors. The data also show that about 8.2 % of the DIMMs manifest correctable errors, matching with the data in earlier large-scale studies [14]. However, in our study, we found that fewer than 0.1 % of the machines and 0.014 % of the total DIMMs generated uncorrectable errors, i.e., 1 order of magnitude lower than the incidences of 1.3–4 % for the machines and 3.5–5.7 times lower than the number of DDR2 DIMMs with uncorrectable errors (0.05–0.08 %) reported in [14]. In particular, the number of uncorrectable errors over the total number of errors handled is more than 3 orders of magnitude lower than that for DDR2 memory systems reported in other large-scale studies [14], even though Blue Waters generates 2 orders of magnitude more machine checks than previous generations of HPC systems [11, 14]. This shows a substantial improvement of resiliency to multiple-bit errors of DDR3 over DDR2. A key implication is that the joint use of ECC and Chipkill techniques in Blue Waters was able to fix 99.998 % of the errors generated in 1.476 PB of DDR3 RAM and 1.5 TB of L1, L2, and L3 caches across all Blue Waters processors. Only 0.002 % of errors were uncorrectable, compared to the 1.29 % reported for the previous generation of HPC systems [14] employing only ECC and/or $\times 4$ Chipkill (single symbol correction, dual symbol detection). The expectation among hardware designers has been that both transient and permanent hardware failures may rise uncontrollably as device sizes shrink, especially in large-scale machines like Blue

Table 10 Breakdown of Blue Waters machine check errors

	Compute		GPU		Service		All	
	Count	Error/MB	Count	Error/MB	Count	Error/MB	Total	Error/MB
L1	27,522	1.62	594	0.26	632	3.48	28,748	1.48
L2	18,590	0.05	1,098	0.02	1,566	0.17	21,254	0.05
L3	292,920	0.81	2,282	0.01	23,030	1.98	318,232	0.75
Memory	840,322	5.66E-4	97,974	9.73E-4	93,590	3.93E-3	1,031,886	6.42E-4
Other	101,388	-	1,102	-	41,788	-	144,278	
% CPU	34.39		4.93		41.73		33.19	
% RAM	65.61		95.07		58.27		54.41	
Total	1,280,742		103,050		160,606		1,544,398	

Waters. However, results indicate that we are far from that situation. Because of the high numbers of nodes and errors, we claim that the results provided in this section have a strong statistical significance for the characterization of processor and memory error resiliency features.

5.2 Rate of Uncorrectable Errors for GPU Cards

Table 12 reports the rates of uncorrected memory errors in Blue Waters nodes and NVidia GPU accelerators. We note the following: (i) for uncorrectable memory errors, the DDR5 memory on the GPU accelerator shows an MTBF 1 order of magnitude smaller than that for the DDR3 constituting the GPU node RAM and 2 orders of magnitude smaller than that of compute nodes (a similar comparison holds for the FIT/GB reported in Table 12); and (ii) the disparity is even higher if we look at the number of uncorrectable errors per GB reported in Table 12. In particular, we note that the rate of uncorrectable errors per GB on GPU node DDR3 memory is 2 orders of magnitude smaller than that on the NVidia accelerator DDR5 on-board memory and that the FIT rate per GB of memory of GPU accelerators is 10 times higher than that for the DDR3 RAM of the nodes. An implication is that enforcement of Chipkill coding for compute, and service and GPU node DDR3 RAM can decrease the rate of uncorrectable errors per GB by a factor of 100, compared to the rate for the ECC-protected memories (e.g., the DDR5 memory on the GPU card). The rate of uncorrectable ECC errors on supercomputer GPU accelerators has not been successfully quantified by any former study for enterprise-level GPU accelerators. The impact of memory errors in GPU accelerator memory could represent a serious threat to creation of future large-scale hybrid systems. For instance, Titan [15] at ORNL adopted about 5 times the number of GPU nodes and cards as Blue Waters, making a substantial step toward fully GPU-based supercomputers.

Table 11 Breakdown of the count of memory errors

Type	Count	%
Total memory errors	1,031,886	66.81
ECC/chipkill single bit	722,526	70.01
Chipkill (more than 2 bit)	309,359	29.98
Uncorrectable ECC/Chipkill	28	2.71E-05

Table 12 Breakdown of the uncorrectable memory errors (UE)

	RAM [GB]	Errors/node	UE	UE/GB	MTBF (UE)
Compute	1,448,960	37.1	14	1.08E-05	1617 h
GPU	127,104	31.1	4	3.88E-05	768 h
Service	23,232	48.1	10	6.22E-05	193 h
GPU card	18,432	9.76E-3	38	2.06E-03	80 h

6 Errors Affecting Application-Level Resiliency

We conducted the analysis of data produced by Blue Waters during the 365 production days (August 1, 2013 to August 1, 2014) to create an initial error categorization. Our dataset includes 2,359,665 user application runs of more than 1,500 code bases, 769,321 jobs, and 296,114,457 error events stored in about 4 TB of syslog. During the measured period, we measured an MTBF of 8.8h, and an overall availability of 0.968, computed after excluding scheduled downtimes, system upgrades, and programmed maintenance actions.

6.1 Application Exit Status

Table 13 shows the breakdown of the job and applications in our dataset. 64.53 % of the total user runs are XE applications (i.e., 1,522,694), and 35.46 % are XK applications (i.e., 836,971) using CPU and GPU accelerators. To compare the composition of XE and XK applications with respect to application scale, we subdivided the applications into 6 classes following the rules in Table 14. Blue Waters data include only a limited number of applications that can effectively use full-scale executions. Even when running at full scale, many applications do not execute for a long time,

Table 13 Blue Waters Workload across different scales

Scale	XE	XE	% XE apps	% XK apps
Single	<=4 (1 blade)		53.89	84.86
Nano	<= 96 (1 cabinet)		39.24	14.13
Low	<= 512 (1 row)		5.33	0.88
Med	≤5896 (25 % sys)	≤1056 (25 % sys)	1.35	0.09
High	≥11792 (50 % sys)	≥2122 (50 % sys)	0.16	0.03
Full	> 11792	>2122	0.04	0.01
	Total application runs		3,365,617	1,724,126

e.g., 75 % of the full-scale XE applications in the measured data ran for less than 5 h, with a median of 1.2 h.

6.1.1 Dissecting the Application Exit Status

There are more than 256 possible application exit codes, many of which are ambiguous or application dependent. An example of an ambiguous exit code is exit code 143 (application terminated by issuing a TERM signal), which can be issued when the application is killed either by system errors or by the user. *LogDiver* is able to disambiguate and categorize an application exit reason by matching error data with application exit code data. Exit reasons are classified into the following categories: (i) Success, for applications completing successfully; (ii) Wall time, for applications not completing within the allocated wall-clock time; (iii) User, for abnormal terminations caused by user-related problems, including compiler/linking/job script and command errors, missing module/file/directory or wrong permissions, and user-initiated actions such as a control-C signal or termination/kill commands; (iv) System, when an application is terminated due to system-related issues caused by any of the considered system errors; and (v) User/System, when an application is terminated for causes that can be related to both user and system events, such as errors detected by the applications (e.g., through assertions) and handled by means of legitimate exit.

Figure 14 gives the breakdown of the application exit statuses. 61.2 % of XE applications (Fig. 14a) and 76.4 % of XK applications (Fig. 14b) exited successfully. The remaining applications failed due to several reasons, including the following: (i) application execution time exceeded the time limit (3.4 % for XE and 7.1 % for

Table 14 Failure statistics of Blue Waters components estimated from the failure reports with respect to production hours

Components	Failure statistics (MTBI (h) and MNBF (h))	
XE/XK node	● XE _{sys} = 8.6 (MTBI)	● XE _{node} = 128, 832 (MNBF)
	● XK _{sys} = 25.1 (MTBI)	● XK _{node} = 63, 598 (MNBF)
Interconnect (Gemini HSN and Lustre Network)	● Gemini = 857.7 (MTBI)	● LNet _{sys} = 359.8 (MTBI)
		● LNet _{home} = 10,294 (MTBI)
		● LNet _{project} = 10,294 (MTBI)
		● LNet _{scratch} = 571.88 (MTBI)
File system/Storage	● Home = 343.2 (MTBI)	● Home (storage node) = 12,348 (MNBF)
	● Project = 263.1 (MTBI)	● Project (storage node) = 9,468 (MNBF)
	● Scratch = 35.4 (MTBI)	● Scratch (storage node) = 10,292 (MNBF)

XK, category ‘wall time’); (ii) user-related problems (22.2% for XE and 12.2% for XK, category User); (iii) system-related problems (1.4% for XE and 1.83% for XK) caused by hardware, software, configuration, or network issues at the system or node levels and occurred with a MTBI (production hours/total application interrupts) of 15 min; and (iv) a combination of user- and system-related causes, e.g., exceptions raised because of issues with Gemini rerouting (12% for XE and 2% for XK applications, category ‘user/systems’). In further analysis conducted in this study, we remove the contribution of the categories User, User/System, and Wall time and focus on characterization of application failures due to system-related problems (the category System).

Despite the fact that only 1.53% of applications failed due to system problems, these problems contribute account for about 9% (see Fig. 15) of total production node hours. Those applications ran for 17,952,261 node hours (i.e., the equivalent of about 28 days of continuous 100% use of the system) and eventually have to be relaunched or executed multiple times. Considering an average power consumption of 2 KW/blade [1], reexecuting the applications that failed because of system errors (when a checkpoint is not available) would theoretically add as much as \$421,878 to the Blue Waters’ energy bill. (This is based on a cost of 0.047 c/KW and does not consider other costs for cooling and infrastructures.) Therefore, *the impact of system errors on applications and costs of ownership is substantial and destined to grow for larger machines.*

Figure 16 illustrates the breakdown of the application exit statuses for all those applications that experienced at least 1 error during their execution. In particular, Fig. 16 shows the joint distribution of how applications terminate when operating under error conditions. XK applications show little resiliency to error compared to XE applications. In particular, the success rate of XK application goes down from 76.6% to 49% when the applications operate under error. At the same time, the percentage of applications failing because of system problems grows from 1.83% to 40.55%. The same phenomenon has a more modest yet substantial manifestation for XE nodes, where the success rate goes down from 61.27 to 56.54%. Another interesting observation is the percentage of applications exiting with unknown status:

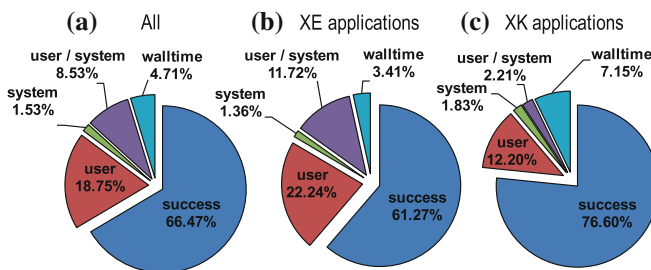
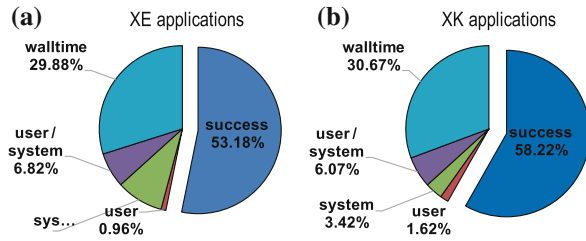


Fig. 14 Breakdown of the application exit reasons generated by (a) all applications, (b) XE applications, (c) XK applications decoded by *LogDiver*

Fig. 15 Breakdown of the application node hours corresponding to the exit reasons generated by *LogDiver* for XE and XK applications



only 0.002 % of XE applications that suffered error are in this category compared to 0.475 % for XK applications. We speculate that this is because of the poor error detection mechanism currently used on the K20X GPUs of XK7 nodes. As will be detailed later, XK applications are more sensitive to system errors for a variety of reasons. Recovering from GPU errors without appropriate support from error detection mechanisms is a hard task, sometimes impossible.

7 Application Resiliency at Different Scales

In this section, we use the output produced by *LogDiver* to measure the resiliency of XE and XK applications. Measurements are produced by *LogDiver* with respect to different application scales, from single node applications up to full scale.

To provide a comparison between platform failures and application failures, we analyze the data extracted from the failure reports analyzed in Sect. 4. Failure events considered include node interrupts (e.g., uncorrectable hardware exceptions), file system and network failures, and system-wide outages (i.e., the entire system is not usable and needs to be repaired). Results are shown in Table 14.

Table 15 shows MTBI (Mean Time Between Application Interruptions) number obtained for different scales and error categories. MTBI is computed as the total number of system hours spent computing at scale x divided by total number of failures occurring because of category c during that time period.

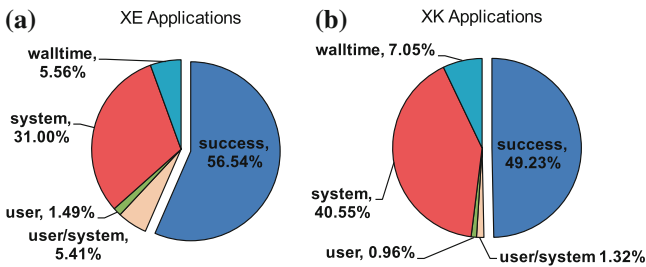


Fig. 16 Distribution of exit status for application experiencing ≥ 1 errors

Table 15 MTBI (Mean Time Between Application Interruption) figures for different scales and error categories

XE applications MTBF (Node MTBF = 128,832 h)									
MTBI XE applications (hours)—system level MTBI XE nodes = 8.6 h									
	Gemini	Health check	LNet	Lustre	Node/Blade	OS			Overall (h)
Full	4	5	20	12	2	12			8.8
High	7	8	9	7	3	25			12.8
Medium	15	10	3	5	25	38			52.2
Small	6	9	2	1	5	48			842.1
Nano	132	224	42	30	175	1,322			1,336
Single	1,872	2,466	452	206	10,078	4,677			5,209
MTBI XK applications (hours)—system level MTBI GPU = 25.1 h									
	Gemini	Health check	LNet	Lustre	Node/Blade	OS		GPU Drivers	Overall (h)
Full	23	21	10	8	10—	—			15.1
High	415	108	131	96	100	142		498	113.1
Medium	287	157	105	88	135	515		2,999	—
Small	10	10	3	2	5	382		179	148.3
Nano	474	371	151	103	1,496	6,043		4,872	205
Single	146	144	38	22	61	23,509		3,155	1,761

Comparing the MTBI figures computed for system components and applications,¹ shown in Table 15, we observe that, when running at full-scale, XE applications are able to perform with an MTBI which is about the same as that of the underlying computing platform, i.e., 8.8 h (see Table 7). This observation implies that resiliency mechanisms of XE applications do an excellent job of protecting large-scale applications from various errors. For full-scale XK applications, the MTBI achieved is 15.1 h. Unlike XE applications, XK applications at full scale are not able to match the theoretical MTBI (of the underlying platform) of 25.1 h, thus, leaving room for improvement. In particular, concerning GPU nodes, our measurements show that the MTBI of XK applications is about 60% lower than the MTBI of the underlying GPU platform (25.1h). An in-depth analysis of GPU/Hybrid applications shows that they are more difficult to handle during error recovery procedures (e.g., checkpoint/restart). Many GPU-related modules are not restartable or replaceable without impacting other software parts. In addition, the lack of communication between different software stacks on XK nodes (e.g., GPU drivers with OpenMP or Charm++) makes it difficult for the resiliency mechanisms (e.g., heartbeats in Charm++ or connection fan-out in MPI) to detect errors related to the GPU stack. For instance, a node may reply to heartbeat messages while manifesting errors in the GPU stack that can kill the application without starting any recovery. As a consequence, the resiliency mechanisms adopted are only partially effective. These results also show that, when running at full-scale, the checkpoint/restart mechanisms are severely stressed. This not only means that a computation makes little progress because of the overhead introduced by the checkpoint/restart necessitated by frequent failures. This also means that fault-handling protocols may need to handle multiple errors, e.g., the failure of two nodes sharing the same copy of the checkpoint file in Charm++ applications. In this case, even for the most advanced resiliency mechanism employed by the system, the failure may be catastrophic and the application may fail inexorably.²

7.1 General Relationship Between Scale and Resiliency

In this section, we estimate the MTBI and the probability of application failure caused by system errors as a function of the number of nodes and node hours. This analysis is performed by grouping the entries in the data set produced by *LogDiver* into several buckets of fixed size representing the scale of the application in terms of both node hours used and nodes used. The metrics considered are then computed with respect to each bucket. For instance, the MTBI for applications executing on 1 to 96 nodes (i.e., 1 cabinet, see Table 13) is computed as the ratio between the sum of the number of hours used by all the applications in the bucket and the number of application failures in that bucket caused by system errors.

¹A detailed description of the comparative analysis between Tables 14 and 15 is reported in [5, 9].

²An example of an efficient checkpoint/restart at full scale is that of the *rhmd* application, which shows an MTBI of 34 h when running on 20,000 nodes.

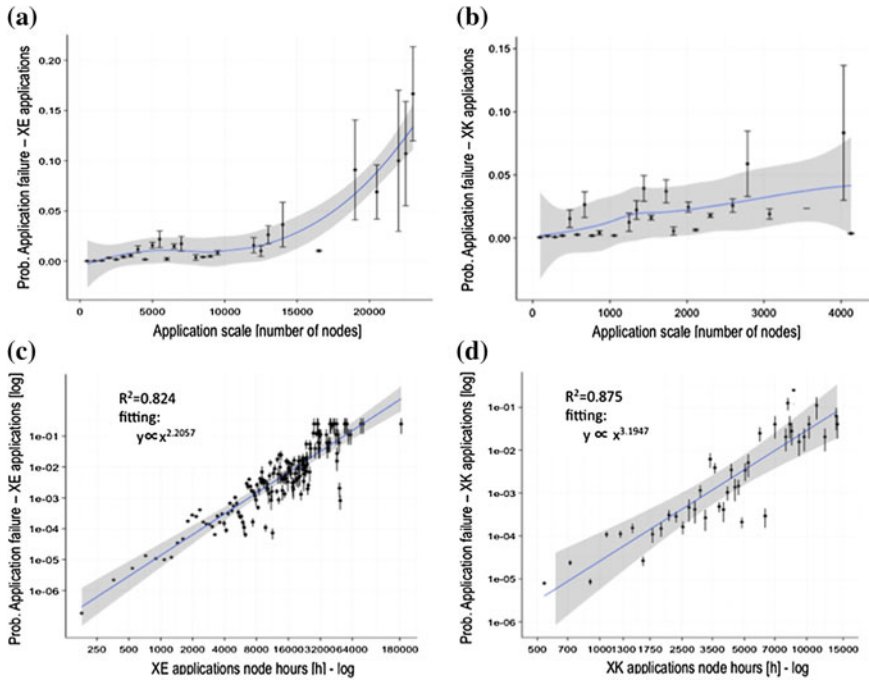


Fig. 17 Probability of application failure plotted against: node hours on XE (a) and XK (b); application scale (number of nodes) on XE (c) and XK (d). The shaded area represents the 95% confidence interval of the smoothing polynomial function

7.1.1 Failure Probability Across Different Scales

Figures 17a, b show the probability of an application terminating because of a system problem as a function of the application scale, for XE and XK nodes, respectively. Data in these plots include applications implemented with and without checkpoint/restart. The error bars show the 95% confidence intervals. Larger bars for large-scale applications are due to the limited number of applications that can effectively use full-scale executions (about 0.04% of the XE applications and 0.01% of XK applications use more than 75% of the system). We observe two different trends of increasing failure probability for XK and XE applications when they scale up. In particular, for medium-scale XE applications executing on fewer than 10,000 nodes, we observe only a small increment in the failure probability, which stays below 1%. When scaling up further, we see a substantial change in the slope of the plot. We measured a failure probability of 0.162 for applications running on more than 22,000 nodes against a failure probability of 0.008 for applications running on fewer than 10,000 nodes. We analyze the causes of the application failures occurring before and after the change of the slope (approximately 10,000 nodes) and observe that the percentage of applications that fail because of interconnect-related issues

increases following a similar trend. For example, we measure that, for large-scale runs ($\geq 10,000$ nodes), 66% of the failures of *namd* when running on XE nodes were because of Gemini- and LNet-related problems, but 23% for small–medium scale runs ($< 10,000$ nodes). This finding shows that interconnect resiliency starts to become a major concern when scaling up to more than 50% of the system size.

7.1.2 Failure Probability Across Processed Node Hours

Figure 17c, d show the effect of the number of node hours on the probability of application failures for XE and XK applications using log–log plots. To formalize our observation above, we fit the data in the plots with a linear regression model (in the log–log space). The probability of application failures for both XE and XK applications can be described with good approximation (as measured by the R^2 score shown in Fig. 17c, d) by a monomial relationship³ of the form $y = \alpha x^k$, which appears as a straight line in a log–log graph, with the power and constant term corresponding, respectively, to the slope and intercept of the line. Examining the fitting, we observe that the probability of an application failure follows a power-of-3 function of the node hours for XK applications and a power-of-2 function of the node hours for XE nodes. In other words, the number of processed node hours has a more pronounced effect on the resiliency of XK applications than on the resiliency of XE applications. This finding emphasizes the need for (i) dedicated resiliency techniques (e.g., memory protection using the chipkill technique) to be deployed for preventing error propagation from the hardware to the application-level and (ii) effective testing of extreme-scale hybrid applications to harness hybrid computing cores in future machines.

8 Related Work

Several studies have attempted to evaluate large-scale systems [11, 12, 16–21]. They have addressed one or more of the following issues: basic error characteristics [16, 17, 19, 22–25], modeling and assessment [20, 26–28], and failure prediction and proactive checkpointing [29–31]. In this chapter, we provide the first study characterizing the failures, errors, and related root causes of a sustained petascale system. Following a process similar to [11], we base our study on the manual failure reports produced by NCSA technical staff managing Blue Waters. In addition to the manual reports, we use system logs to conduct deeper analysis and characterize the resiliency of the hardware protection mechanisms.

³Given a monomial equation $y = \alpha x^k$, taking the logarithm of the equation (with any base) yields $(\log y) = k (\log x) + \log \alpha$. Setting $X = \log x$ and $Y = \log y$ (i.e., moving to log–log graph), yields the equation $Y = mX + b$.

A number of researchers have classified failures in relation to their root cause, e.g., hardware or software [11, 32]. Hardware failures are identified in [32] as contributing 6% of the total failures, whereas 42% were due to software failures. Hardware is identified in [11] as the single largest cause of node failures, with percentages ranging from 30% to more than 60% with respect to the systems analyzed; software percentages range from 5 to 24%. The authors focus only on single node downtime and failure causes, and they do not explicitly characterize system-wide outages. In our work, by comparing the earlier generation with current HPC systems, we demonstrate that there has been a strong technological improvement in hardware resiliency features. Data show that the majority of system outages in Blue Waters are due to software errors (74.4%), while hardware contributes only to single node/blade downtime node hours in 99.4% of documented failures. In addition, because of their crucial role in large-scale systems, we characterized effectiveness of failover and protection mechanisms at different scales, i.e., network, file system, processor, memory, and high-performance GPU accelerators.

Improved fault tolerance comes from detecting and autocorrecting a greater percentage of high-impact errors. However, most studies do not consider the impact of errors and failures in their analysis, making reactive methods less effective. While much of the mentioned work provides novel filtering approaches, little of it includes in the analysis those errors that really impact the production workload. As a result, the available error/failure characterization studies and techniques for extreme-scale machines do not provide enough fidelity of understanding to enable researchers and system architects to learn how applications behave when exposed to errors or to assess requirements for future architectures. Some errors do not pose a real threat to either system or application operations. In this chapter, we quantify the impact of system errors on production workload. We show that many applications are able to complete during system-wide outages thanks to the containment of system failures to a limited portion of the system, e.g., to specific file systems or cabinets. As we demonstrate in this chapter, it is advantageous to analyze system-level error events as a function of the application. Blue Waters logs, however, contain more than 150 different types of errors that can impact user applications [9]. The analysis and the *LogDiver* tool presented in this chapter are the first to correlate errors with user application failures in extreme-scale environments.

9 Conclusions

This chapter presents an in-depth study of the resiliency of Blue Waters. The overall failure characterization indicates that failures due to software are the major contributors to total repair time (53%), although they represent only 20% of the total number of failures. More importantly, software is also the primary cause (74.4%) of SWOs (system-wide outages). The analysis points out that the real system bottleneck is the inadequacy of mechanisms behind complex failover operations (e.g., timeout and distributed locks managers), such as those employed in the Lustre file system. Blue

Waters is a configuration corner case on a scale well beyond that at which hardware and software have been methodically tested, so our findings have significant value. To our surprise, hardware, regardless of its complexity, is highly resilient. In particular, the use of error-correcting codes (including Chipkill) provides extremely high coverage (99.997%) for memory and processor errors, and hence high overall system resiliency. This benefit is seen even in the presence of high error rates (250 errors/h in our case). In the future, we plan to collect data from similar systems and conduct a detailed analysis of Lustre error resiliency at different scales.

The lessons learned from the analysis of application-level resiliency include:

- *In terms of efficiently handling the large volume of heterogeneous textual data*, we developed *LogDiver*, a tool to automate the data preprocessing and metric computation, e.g., the mean node hours between failures and the probability of an application failure. Through *LogDiver*, it is possible to correlate system-related errors and failures with application features. The results of the analysis allow the design of novel data collection mechanisms to support application-aware fault classification and to derive new metrics to predict the resiliency of the next generation of extreme-scale systems.
- *In terms of application susceptibility to system problems*, our analysis shows that 1.53% of applications fail due to system problems. One may argue that this is a small percentage of the overall application runs (more than 5,000,000) considered in this study. However, the failed applications contribute to about 9% of production node hours. As a result, the system consumes computing resources and encounters significant energy cost from the work that must be redone.
- *In terms of resiliency technique*, our data show that application-level checkpoint/restart plays an essential role in improving application resiliency to system problems. The MNBF ((Mean Node Hours Between Failures) of applications protected with checkpoint/restart techniques is at least 2 (or more) times greater than the MNBF of applications that do not have checkpoint/restart support.
- *In terms of the design of future extreme-scale (or exascale) systems*, our study indicates that one must be cautious when using a massive number of hybrid computing nodes. For example, in the analyzed system, the application failure probability increases following the cubic function of the number of node hours when executing on GPU nodes. Consequently, dedicated resiliency techniques (e.g., memory protection using the chipkill technique) must be deployed to prevent errors from propagating to the application level.

References

1. www.cray.com/Assets/PDF/products/xe/CrayXE6Brochure.pdf
2. <http://www.cray.com/Products/Storage/Sonexion/Specifications.aspx>
3. Karo M, Lagerstrom R, Kohnke M, Albing C (2008) The application level placement scheduler. In: Cray User Group—CUG

4. <http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-suite-enterprise-edition>
5. Di Martino C, Kramer W, Kalbarczyk Z, Iyer R (2015) Measuring and understanding extreme-scale application resilience: a field study of 5,000,000 hpc application runs. In: 45th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 25–36, June 2015
6. Inc. A, Bios and kernel developers guide, for amd family 16th
7. TDIM Division (1997) A white paper on the benefits of chipkill-correct ecc for pc server main memory
8. Johnsen P, Straka M, Shapiro M, Norton A, Galarneau T (2013) Petascale wrf simulation of hurricane sandy deployment of ncsa's cray xe6 blue waters. In: Proceedings of the international conference on high performance computing, networking, storage and analysis, SC'13, ACM, New York, NY, USA, pp 63:1–63:7
9. Martino CD, Jha S, Kramer W, Kalbarczyk Z, Iyer RK (2015) Logdiver: a tool for measuring resilience of extreme-scale systems and applications. In: Proceedings of the 5th workshop on fault tolerance for HPC at eXtreme scale, ACM, FTXS '15, New York, NY, USA, pp 11–18
10. Goljadina N, Nekrutkin V, Zhigljavsky A Analysis of time series structure: SSA and related techniques
11. Schroeder B, Gibson G (2010) A large-scale study of failures in high-performance computing systems. *IEEE Trans Depend Secure Comput* 7(4):337–350
12. Schroeder B, Gibson GA (2007) Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you?. In: Proceedings of the 5th USENIX conference on file and storage technologies, FAST '07, Berkeley, CA, USA, USENIX Association
13. Sridharan V, Stearley J, DeBardleben N, Blanchard S, Gurumurthi S (2013) Feng shui of supercomputer memory: positional effects in dram and sram faults. In: Proceedings of SC13: international conference for high performance computing, networking, storage and analysis, SC'13, New York, NY, USA, pp 22:1–22:11, ACM
14. Schroeder B, Pinheiro E, Weber W (2009) Dram errors in the wild: a large-scale field study. *SIGMETRICS Perform Eval Rev* 37:193–204
15. <http://www.olcf.ornl.gov/titan/>, number 2 on top500.org
16. Sahoo RK, Sivasubramaniam A, Squillante MS, Zhang Y (2004) Failure data analysis of a large-scale heterogeneous server environment. In: International conference on DSN'04: proceedings of the 2004 dependable systems and networks, pp 772–781
17. Liang Y, Sivasubramaniam A, Moreira J, Zhang Y, Sahoo R, Jette M (2005) Filtering failure logs for a bluegene/l prototype. In: DSN'05: Proceedings of the 2005 international conference on dependable systems and networks, pp 476–485
18. Liang Y, Zhang Y, Jette M, Sivasubramaniam A, Sahoo R (2006) Bluegene/l failure analysis and prediction models. In: DSN 2006 international conference on dependable systems and networks, 2006, pp 425–434
19. Oliner A, Stearley J (2007) What supercomputers say: a study of five system logs. In: DSN '07, 37th annual IEEE/IFIP international conference on dependable systems and networks, pp 575–584, June 2007
20. Di Martino C, Cinque M, Cotroneo D (2012) Assessing time coalescence techniques for the analysis of supercomputer logs. In: Proceedings of 42nd annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 1–12
21. Pecchia A, Cotroneo D, Kalbarczyk Z, Iyer RK (2011) Improving log-based field failure data analysis of multi-node computing systems. In: Proceedings of the 2011 IEEE/IFIP 41st international conference on dependable systems & networks, DSN '11, Washington, DC, USA, pp 97–108, IEEE Computer Society
22. Di Martino C (2013) One size does not fit all: clustering supercomputer failures using a multiple time window approach. In: Kunkel J, Ludwig T, Meuer H (eds) International supercomputing conference—supercomputing, vol 7905 of Lecture notes in computer science, pp 302–316, Springer, Berlin Heidelberg

23. Di Martino C, Baccanico F, Fullop J, Kramer W, Kalbarczyk Z, Iyer R (2014) Lessons learned from the analysis of system failures at petascale: the case of blue waters. In: Proceedings of 44th annual IEEE/IFIP international conference on dependable systems and networks (DSN)
24. Di Martino C, Chen D, Goel G, Ganesan R, Kalbarczyk Z, Iyer R (2014) Analysis and diagnosis of sla violations in a production saas cloud. In: IEEE 25th international symposium on software reliability engineering (ISSRE), pp 178–188, Nov 2014
25. Tiwari D, Gupta S, Gallarno G, Rogers J, Maxwell D (2015) Reliability lessons learned from gpu experience with the titan supercomputer at oak ridge leadership computing facility. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, p 38, ACM
26. Heien E, Kondo D, Gainaru A, LaPine A, Kramer W, Cappello F (2011) Modeling and tolerating heterogeneous failures in large parallel systems. In: Proceedings of 2011 international conference for high performance computing, networking, storage and analysis, SC'11, New York, NY, USA, pp 45:1–45:11, ACM
27. Bruneo D, Longo F, Ghosh R, Scarpa M, Puliafito A, Trivedi K (2015) Analytical modeling of reactive autonomic management techniques in iaas clouds. In: IEEE 8th international conference on cloud computing (CLOUD), 2015, pp 797–804, June 2015
28. Di Martino C, Kalbarczyk Z, Iyer RK, Goel G, Sarkar S, Ganesan R (2014) Characterization of operational failures from a business data processing saas platform. In: Companion proceedings of the 36th international conference on software engineering, ICSE companion 2014, New York, NY, USA, pp 195–204, ACM
29. Chen X, Lu C, Pattabiraman K (2013) Predicting job completion times using system logs in supercomputing clusters. In: 43rd annual IEEE/IFIP conference on dependable systems and networks workshop (DSN-W), 2013, pp 1–8, June 2013
30. Gainaru A, Cappello F, Kramer W (2012) Taming of the shrew: modeling the normal and faulty behaviour of large-scale hpc systems. In: IEEE 26th international parallel distributed processing symposium (IPDPS), pp 1168–1179
31. Gainaru A, Cappello F, Snir M, Kramer W (2012) Fault prediction under the microscope: a closer look into hpc systems. In: International conference for high performance computing, networking, storage and analysis (SC), pp 1–11
32. Oppenheimer D, Patterson DA (2002) Studying and using failure data from large-scale internet services. In: Proceedings of the 10th workshop on ACM SIGOPS European workshop, EW 10, New York, NY, USA, pp 255–258, ACM