
RESPONSIVE COMPUTER SYSTEMS: STEPS TOWARD FAULT-TOLERANT REAL-TIME SYSTEMS

THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE

REAL-TIME SYSTEMS

Consulting Editor
John A. Stankovic

IMPRECISE AND APPROXIMATE COMPUTATION, by Swaminathan Natarajan, ISBN: 0-7923-9579-4

FOUNDATIONS OF DEPENDABLE COMPUTING: *System Implementation*, edited by Gary M. Koob and Clifford G. Lau, ISBN: 0-7923-9486-0

FOUNDATIONS OF DEPENDABLE COMPUTING: *Paradigms for Dependable Applications*, edited by Gary M. Koob and Clifford G. Lau, ISBN: 0-7923-9485-2

FOUNDATIONS OF DEPENDABLE COMPUTING: *Models and Frameworks for Dependable Systems*, edited by Gary M. Koob and Clifford G. Lau, ISBN: 0-7923-9484-4

THE TESTABILITY OF DISTRIBUTED REAL-TIME SYSTEMS, Werner Schütz; ISBN: 0-7923-9386-4

A PRACTITIONER'S HANDBOOK FOR REAL-TIME ANALYSIS: *Guide to Rate Monotonic Analysis for Real-Time Systems*, Carnegie Mellon University (Mark Klein, Thomas Ralya, Bill Pollak, Ray Obenza, Michale González Harbour); ISBN: 0-7923-9361-9

FORMAL TECHNIQUES IN REAL-TIME FAULT-TOLERANT SYSTEMS, J. Vytupil; ISBN: 0-7923-9332-5

SYNCHRONOUS PROGRAMMING OF REACTIVE SYSTEMS, N. Halbwachs; ISBN: 0-7923-9311-2

REAL-TIME SYSTEMS ENGINEERING AND APPLICATIONS, M. Schiebe, S. Pferrer; ISBN: 0-7923-9196-9

SYNCHRONIZATION IN REAL-TIME SYSTEMS: *A Priority Inheritance Approach*, R. Rajkumar; ISBN: 0-7923-9211-6

CONSTRUCTING PREDICTABLE REAL TIME SYSTEMS, W. A. Halang, A. D. Stoyenko; ISBN: 0-7923-9202-7

FOUNDATIONS OF REAL-TIME COMPUTING: *Formal Specifications and Methods*, A. M. van Tilborg, G. M. Koob; ISBN: 0-7923-9167-5

FOUNDATIONS OF REAL-TIME COMPUTING: *Scheduling and Resource Management*, A. M. van Tilborg, G. M. Koob; ISBN: 0-7923-9166-7

REAL-TIME UNIX SYSTEMS: *Design and Application Guide*, B. Furht, D. Grostick, D. Gluch, G. Rabbat, J. Parker, M. McRoberts, ISBN: 0-7923-9099-7

RESPONSIVE COMPUTER SYSTEMS: STEPS TOWARD FAULT-TOLERANT REAL-TIME SYSTEMS

edited by

Donald S. Fussell

*The University of Texas at Austin
Austin, Texas, USA*

Miroslaw Malek

*Humboldt-Universität
Berlin, Germany*

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

ISBN 978-0-7923-9563-8 ISBN 978-1-4615-2271-3 (eBook)
DOI 10.1007/978-1-4615-2271-3

Copyright © 1995 Springer Science+Business Media New York
Originally published by Kluwer Academic Publishers in 1995

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Springer Science+Business Media, LLC .

Printed on acid-free paper.

CONTENTS

LIST OF FIGURES	xi
LIST OF TABLES	xv
PREFACE	xvii
1 A TIGHT LOWER BOUND FOR PROCESSOR COORDINATION	
<i>Soma Chaudhuri, Maurice Herlihy, Nancy A. Lynch and Mark R. Tuttle</i>	1
1 Introduction	2
2 Overview	4
3 The Problem	7
4 The Bermuda Triangle	10
5 The Lower Bound	13
6 Generalizing to the Partially Synchronous Model	14
REFERENCES	16
2 SELF-STABILIZING REAL-TIME DECISION SYSTEMS	
<i>Marco Schneider</i>	19
1 Introduction	19
2 Related Work	21
3 A Model of Computation	22
4 Definitions and Notation	24
5 Terminating Self-Stabilizing Programs	26
6 Boolean (Finite State) Programs	29

7	Non-Terminating Self-Stabilizing Programs	35
8	Self-Stabilization and Informed Decisions	40
9	Acknowledgements	41
	REFERENCES	43
3	MANAGEMENT AND SCHEDULING OF TESTS FOR LOCATING FAILURES DEPENDENT UPON OPERATION-TIME IN RESPONSIVE SYSTEMS	
	<i>Yu Lo Cyrus Chang and Leslie C. Lander</i>	45
1	Introduction	46
2	Definitions and Notation	48
3	Assignment and Scheduling of Multiple Test Sets	52
4	The Analysis of Fault Location	54
5	<i>A priori</i> information analysis	56
6	Conclusion	62
	REFERENCES	62
4	ADDING ROBUSTNESS IN DYNAMIC PREEMPTIVE SCHEDULING	
	<i>Giorgio C. Buttazzo and John A. Stankovic</i>	67
1	Introduction	67
2	Terminology and Assumptions	69
3	Schedulability analysis	71
4	The RED scheduling strategy	75
5	Performance Evaluation	78
6	Related Work	85
7	Conclusions	87
	REFERENCES	87
5	STATIC DEADLOCK ANALYSIS FOR CSP-TYPE COMMUNICATIONS	
	<i>Peter B. Ladkin and Barbara B. Simons</i>	89
1	Introduction	89
2	The Sequence Condition	96
	REFERENCES	101

6 COMPARING HOW WELL ASYNCHRONOUS ATOMIC BROADCAST PROTOCOLS PERFORM

Flaviu Cristian, Richard de Beijer and Shivakant Mishra 103

1	Introduction	103
2	Assumptions	105
3	Overview of Broadcast Protocols	106
4	Simulation Results	108
5	Relative Performance	119
6	Conclusion	121
	REFERENCES	122

7 HARD REAL-TIME RELIABLE MULTICAST IN THE DEDOS SYSTEM

Dick Alstein and Peter van der Stok 123

1	Introduction	123
2	System architecture and failure assumptions	125
3	Protocol description	127
4	Protocol analysis	138
5	Conclusion	140
	REFERENCES	141

8 SPECULATIVE ALGORITHMS FOR CONCURRENCY CONTROL IN RESPONSIVE DATABASES

Azer Bestavros 143

1	Introduction	143
2	RTDBMS Concurrency Control	146
3	A Generic SCC-nS Algorithm	148
4	SCC-NS Family Members	160
5	Conclusion	161
	REFERENCES	162

9 AUTONOMOUS TRANSACTION MANAGERS IN RESPONSIVE COMPUTING

Nandit Soparkar, Henry F. Korth and Avi Silberschatz 167

1	Introduction	168
---	--------------	-----

2	System Structure and Notation	170
3	Synchronization of the Local Schedules	172
4	Effects of Transaction Aborts	177
5	Synchronization of CSR Schedules	178
6	A Pragmatic Restriction	179
7	Further Observations	181
8	Conclusions	183
9	Appendix	184
	REFERENCES	185
 10 ADAPTABLE FAULT TOLERANCE FOR REAL-TIME SYSTEMS		
	<i>A. Bondavalli, J. Stankovic and L. Strigini</i>	187
1	Introduction	187
2	The Three-level Framework	189
3	FERT Specification Language	192
4	Support for Scheduling	200
5	Conclusions and Discussion	204
	REFERENCES	207
 11 FAULT-TOLERANT AUTOMATIC CONTROL		
	<i>Marc Bodson, John Lehoczky, Ragunathan Rajkumar, Lui Sha and Jennifer Stephan</i>	209
1	Introduction to the Software Fault-Tolerance Problem	209
2	The Simplex Architecture	211
3	Laboratory Experiments	215
4	Conclusion	221
	REFERENCES	223
 12 DESIGN AND PERFORMANCE EVALUATION OF A FAULT-TOLERANT, HARD-REAL-TIME, PARALLEL PROCESSOR		
	<i>Bob Clasen, Rick Harper and Edward W. Czeck</i>	225
1	Introduction	226
2	FTPP Overview	226
3	Overview of FTPP Scheduling	228

4	OS Performance Measurements	235
5	NE Contention Model	241
6	Conclusions	246
	REFERENCES	249
	INDEX	251

LIST OF FIGURES

Chapter 1

- | | | |
|---|---|---|
| 1 | The Bermuda Triangle for 5 processors and a 1-round protocol for 2-set agreement. | 5 |
| 2 | A three-round communication graph. | 9 |

Chapter 2

Chapter 3

- | | | |
|---|---|----|
| 1 | Self-diagnosable Responsive System | 47 |
| 2 | Complete-graph Test Assignment | 50 |
| 3 | 1 Failed Unit, Correct Comparisons | 51 |
| 4 | 2 Failed Units, Erroneous Comparisons | 52 |
| 5 | Graphs of the Probability of single <i>vs</i> multiple faults as time t increases (time unit = 1 MTBF), taking $\lambda = 1, \alpha = 1, n = 6$ | 61 |
| 6 | Graphs of the Probability of single <i>vs</i> multiple faults as time t increases (time unit = 1 MTBF), taking $\lambda = 1, \alpha = 2, n = 6$ | 61 |

Chapter 4

- | | | |
|---|-------------------------------|----|
| 1 | RED Guarantee Algorithm. | 76 |
| 2 | RED Scheduling Block Diagram. | 78 |

Chapter 5

- | | | |
|---|-----------------------|----|
| 1 | Processes A, B, and C | 90 |
|---|-----------------------|----|

Chapter 6

- | | | |
|---|-----------------------------|-----|
| 1 | Communication delay density | 106 |
| 2 | Delivery time (PA) | 109 |

3	Delivery time (PA)	109
4	# of messages per broadcast (PA)	110
5	# of messages per broadcast; 1 msg loss per broadcast (PA)	110
6	Delivery time; 1 msg loss per broadcast (PA)	111
7	Delivery time; 1 msg loss per broadcast (PA)	111
8	Delivery time (Amoeba)	111
9	Delivery time (Amoeba)	111
10	Delivery time; 1 msg loss per broadcast (Amoeba)	112
11	Delivery time; 1 msg loss per broadcast (Amoeba)	112
12	# of messages per broadcast (Amoeba)	113
13	# of messages per broadcast; 1 msg loss per broadcast (Amoeba)	113
14	Delivery time (Train)	114
15	Delivery time; 1 msg loss per broadcast (Train)	114
16	Delivery time (Train)	115
17	Delivery time; 1 msg loss per broadcast (Train)	115
18	# of messages per broadcast (Train)	116
19	# of messages per broadcast; 1 msg loss per broadcast (Train)	116
20	Delivery time (Isis)	117
21	Delivery time (Isis)	117
22	Delivery time; 1 msg loss per broadcast (Isis)	117
23	Delivery time; 1 msg loss per broadcast (Isis)	117
24	# of messages per broadcast (Isis)	118
25	# of messages per broadcast; 1 msg loss per broadcast (Isis)	118

Chapter 7

1	Timing diagram of a typical send operation with timestamp τ .	130
2	Timing diagram of a typical receive operation starting at time t .	131

Chapter 8

1	Legend for symbols used in illustrations throughout this article.	146
2	Example of transaction management under a basic OCC algorithm.	147
3	Example of transaction management under the OCC-BC algorithm.	147
4	An undeveloped potential conflict.	148
5	A developed conflict.	148

6	T_1 detects conflict (T_3, X) after T_3 writes X .	151
7	T_1^3 is forked off the <i>BestShadow</i> (T_1, X) , T_1^1 .	153
8	T_1^j , which accounts for the (T_2, Z) conflict, is aborted and replaced by T_1^k when an <i>earlier</i> conflict, (T_2, X) , with T_2 is detected.	154
9	Detecting conflict (T_2, X) causes the abortion of <i>LastShadow</i> (T_1) (T_1^2), and its replacement by T_1^3 .	155
10	T_1^2 , accounting for the developed conflict (T_2, X) , is promoted to replace the optimistic shadow of T_1 . T_1^3 is aborted, while T_1^1 remains unaffected.	155
11	When the unaccounted-for conflict (T_2, Z) materializes, a new optimistic shadow for T_1 is forked off the <i>LastShadow</i> (T_1) , T_1^2 .	156
12	The SCC-nS Algorithm.	157
13	The LastShadow and BestShadow Functions used in SCC-nS.	158
14	Average number of missed deadlines for OCC-BC vs. SCC-2S	159
15	Average tardiness for OCC-BC vs. SCC-2S	159

Chapter 9

1	MDBS Structure	171
2	Overlapped synchronization intervals	174
3	Impossible cycle with the synchronization protocol	176

Chapter 10

1	Structure of a FERT.	193
---	----------------------	-----

Chapter 11

1	Software Architecture	213
2	Ball and Beam: Unrecoverable Region	219
3	Ball and Beam - Recovery at High Speed	220
4	Ball and Beam - Recovery from Fuzzy Logic Controller	221

Chapter 12

1	FTPP Physical Architecture	231
2	FTPP Virtual Configuration	232
3	FTPP Abstract Structure	232

4	Architecture of Rate Group (RG) Frames on a Virtual Group (VG)	233
5	Rate Group Frame - Programming Model	233
6	Overview of Minor Frame	234
7	Performance Measurement Setup	236
8	Message Packet Processing	244
9	Phasing Among PEs	245
10	Effect of Varying Number of Packets and Phasing on Time to Send Message Packets	247
11	Effect of Varying Number of PEs and Phasing on Time to Send Message Packets	247
12	Effect of Varying Process Packet Time and Transfer Packet Time on Time to Send Message Packets	248
13	Effect of Varying Process SERP Time and Transfer Packet Time on Time to Send Message Packets	248
14	Effect of Reducing Each Default Parameter by 50% on Time to Send Message Packets	249

LIST OF TABLES

Chapter 1

Chapter 2

Chapter 3

- | | | |
|---|--|----|
| 1 | Case of an n -unit system with $\alpha = 1$ | 60 |
| 2 | Case of an n -unit system with $\lambda = 1$ | 66 |

Chapter 4

Chapter 5

Chapter 6

Chapter 7

- | | | |
|---|---|-----|
| 1 | Performance characteristics of the protocols. | 138 |
|---|---|-----|

Chapter 8

Chapter 9

Chapter 10

Chapter 11

Chapter 12

1	Execution Time for RG Dispatcher - First Part	240
2	Execution Time for RG Dispatcher - Second Section	240
3	Execution Time of the Local FDIR Task	240
4	Example Operating System Overheads	240

PREFACE

Modern computer systems are rapidly evolving into a highly integrated global network which serves as the repository of ever larger amounts of information critical to the social infrastructure. At the same time, a increasing fraction of the general public is becoming familiar with, and dependent upon, computers in the home. As this evolution continues, there is a growing need for computing environments that can be relied upon to provide timely responses to demands for information and service.

For some time now, a great deal of attention has been paid to the design of fault-tolerant computers for special, mission-critical applications. In parallel, much research has been devoted to methods for designing real-time systems, which again are usually intended for specialized applications. While some of these applications demand highly reliable real-time service guarantees, for the most part research on the design of real-time computer systems and fault-tolerant computers has progressed independently. However, with increasing societal dependence on computers for everyday operation, the integration of fault-tolerance and real-time capabilities into new generations of computer systems has become a significantly more important issue in systems design. We believe that the time has come for the development of a discipline of *responsive systems* design, by which we mean the design of systems optimized to meet demands for reliable and timely service.

While many approaches to the design of reliable systems have been developed, all of them share one characteristic in common — they rely on replication of resources in some way to achieve high reliability. Thus, the advent of widespread parallel and distributed computing has been greeted as the basis of a new generation of computing systems which use redundancy to achieve high reliability. To date, however, the reality in most parallel and distributed systems has been quite the reverse of this hope. The typical user of a distributed client-server environment actually experiences less reliable service than was provided by a time-shared mainframe or minicomputer in previous years. This is because each user of such a system typically relies on a reasonably large number of individual computers for his overall service needs. In many current environments, the fail-

ure of any one of these machines brings down the user's environment. While each machine may be more robust than its counterparts of previous generations, with overall system designs which ignore fault-tolerance, the probability that the user sees a functional system declines exponentially with the number of machines on which he is dependent.

A similar situation exists with respect to the use of parallelism in the design of high-performance machines to meet severe real-time constraints. A parallel system may in principle be able to provide a performance level necessary to the achievement of a real-time constraint beyond the performance limits of any sequential system. Since parallel systems are significantly more complex and harder to program than sequential systems, however, it is harder to realize their performance potential in practice. Moreover, there is a significantly greater opportunity for asynchronous, non-deterministic, or other unpredictable behavior in parallel or distributed computing systems, which makes it much harder to guarantee that a real-time constraint will always be met.

Thus we have the situation at present that, while the technology of parallel and distributed computing contains within it precisely the raw material needed for the design of responsive systems, it is most often the case in current system designs that responsiveness is actually harder to achieve than in sequential systems in spite of this potential. It is a significant challenge to convert the potential of modern, parallel and distributed computing environments into realized responsive computing systems. Developing a discipline of responsive systems design will require integrating the theory and practice of real-time systems, fault-tolerant computing, and parallel and distributed processing into a whole which is more than the sum of its parts.

In order to take the first steps in this direction, we must begin with a clearer understanding of the goal and the costs of achieving it. Much of the reason that current distributed systems for non-critical applications are so unreliable is that they have been designed to maximize raw benchmark speeds while minimizing the cost of storage and processing power. Thus, disks are not used to hold replicated copies of data; processors are not employed to back up the functions of others. The cost to the user of downtime is not usually considered in the design of these systems; hence efforts to minimize costs can adversely affect reliability. Any use of redundancy that might entail performance costs is considered harmful using this metric. Thus, if we are to undertake a discipline of responsive systems design, we must begin by recognizing the value of system availability and finding a way to measure it against the values of other system capabilities such as performance level and storage capacity.

We must in the same vein modify our notion of how to evaluate performance. Raw benchmark speed must be deemphasized and replaced with a valuation based on the expectation of the timeliness of the responses that a system will typically provide to its users. System resources beyond those required to provide timely responses should preferentially be used for greater system reliability or availability then for increasing raw performance as is typically the case today. Overall, then, we need a measure of system value in terms of responsiveness that lets us make design decisions regarding how system resources are to be used to provide the highest level of responsiveness at a given cost level, or alternatively the lowest cost for a given level of responsiveness. We then need to develop methodologies for designing systems and algorithms that will allow us to satisfy reliability and timeliness requirements in a flexible and efficient way.

Given the complexity of modern computing systems and the ambitious goals of responsive systems design, a wide variety of issues will need to be resolved before these goals can be achieved. For example, how can we meet high level specifications with respect to fault tolerance and timeliness, or, in fact, how do we even specify these requirements? How do we express and estimate bounds on computation time? What models should be used to describe the system? What methods should be used to prove that a design or an implementation meet a responsiveness specification? What language, operating system and architecture paradigms would be most appropriate? How do we design algorithms and programs that behave responsively?

In the present volume, which contains twelve papers selected from the Second International Workshop on Responsive Computer Systems that took place in Lincoln, New Hampshire on September 28-30, 1993, you will find some initial steps toward answering these and related questions.

Donald S. Fussell
Miroslaw Malek