

Breaking the Availability Barrier I
Survivable Systems for Enterprise Computing

Volume 1 of 3 Volume Series
Dr. Bruce Holenstein, Dr. Bill Highleyman, and Paul J.
Holenstein

© 2007 Gravic, Inc. All rights reserved.

No part of this book may be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the authors.

ISBN: 978-1-4107-9231-0 (e-book)

ISBN: 978-1-4107-9232-7 (Paperback)

ISBN: 978-1-4107-9233-4 (Dust Jacket)

Library of Congress Control Number: 2003099708

All products mentioned in this book are trademarks of their respective owners. The information in this book is provided on an as-is informational basis. The authors, owners, and publisher disclaim liability for any errors or omissions. The reader accepts all risks associated with the use of the contents of this book.

About the Authors:

The authors of the book, Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein, have a combined experience of over 90 years in the implementation of fault-tolerant, highly available computing systems. This experience ranges from the early days of custom redundant systems to today's fault-tolerant offerings from HP (NonStop) and Stratus.

Click for Book Order Information: [Breaking the Availability Barrier](#)
or visit Amazon.com or Authorhouse.com to purchase.

Breaking the Availability Barrier

*Survivable Systems for Enterprise
Computing*

Breaking the Availability Barrier

*Survivable Systems for Enterprise
Computing*

**Dr. Bill Highleyman
Paul J. Holenstein
Dr. Bruce Holenstein**

© 2004 by Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein.
All rights reserved.

No part of this book may be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the authors.

ISBN: 1-4107-9231-5 (e-book)
ISBN: 1-4107-9232-3 (Paperback)
ISBN: 1-4107-9233-1 (Dust Jacket)

Library of Congress Control Number: 2003099708

This book is printed on acid free paper.

Printed in the United States of America
Bloomington, IN

All products mentioned in this book are trademarks of their respective owners. The information in this book is provided on an as-is informational basis. The authors, owners, and publisher disclaim liability for any errors or omissions. The reader accepts all risks associated with the use of the contents of this book.

Dedication

This book is dedicated to our spouses,
Janice, Karen, and Denise,
for their enduring patience and support.

Breaking the Availability Barrier

Contents

Forward.....	xvii
PART 1 – BREAKING THE FOUR 9S BARRIER	xviii
PART 2 – ADVANCED TOPICS.....	xxii
APPENDICES	xxiii
AUTHORS’ NOTES	xxiv
ACKNOWLEDGEMENTS.....	xxv
ABOUT THE AUTHORS.....	xxv
<i>Part 1 - Breaking the Four 9s Barrier.....</i>	<i>1</i>
Chapter 1 - The 9s Game.....	3
WHAT IS RELIABILITY?	3
SOME CAVEATS.....	6
9S – THE MEASURE OF AVAILABILITY	6
TODAY’S SYSTEMS.....	8
SIMPLE SYSTEMS	8
Non-Redundant System	8
Redundant System	9
DOUBLE YOUR 9S.....	10
THE REAL FAULT-TOLERANT WORLD	10
RANDOMLY DISTRIBUTED PROCESS PAIRS	11
PROCESS/PROCESSOR PAIRING.....	13
AVAILABILITY IN GENERAL	14
MORE SPARING.....	15
HOW MANY FAILURE MODES?	16
THE IMPACT OF REPAIR TIME	17
SOME HELPFUL CHARTS.....	19
ANSWERS	21

Breaking the Availability Barrier

SUMMARY	22
HOW FAR SHOULD WE GO?	23
A CASE STUDY	24
WHAT'S NEXT?.....	25
Chapter 2 - System Splitting	27
THE AVAILABILITY RELATION	27
FULL REPLICATION	29
SYSTEM SPLITTING.....	31
Simple Splitting.....	31
Multiple Splitting	34
Impact on Mean Time Before Failure.....	36
Elimination of Planned Down Time	37
REPLICATION OF DATA	37
MUST WE REPLICATE THE DATABASE?.....	39
SUMMARY	42
SYSTEM SPLITTING AND PROCESS PAIRING	43
WHAT'S NEXT?.....	44
Chapter 3 - Asynchronous Replication	45
USES FOR DATA REPLICATION.....	46
Disaster Tolerance.....	46
Increased Availability	46
Localization.....	46
System Maintenance	47
Enterprise Application Integration (EAI).....	47
Operational Data Store (ODS)	47
Data Warehousing.....	47
Zero Down Time System Migration	48
TYPES OF DATA REPLICATION	48
Directionality	49
Threading	51
Queuing	52
Target Updating	53
Synchronism.....	54
THE BASIC DATA REPLICATION ENGINE.....	56
ADVANTAGES OF ASYNCHRONOUS REPLICATION	59

Breaking the Availability Barrier

No Performance Penalty	60
Non-Invasive.....	60
Heterogeneous.....	60
Data Manipulation	60
Data Integrity	61
Highly Reliable.....	61
Highly Available.....	61
Highly Scalable.....	61
Highly Secure.....	62
ASYNCHRONOUS REPLICATION ISSUES.....	62
General Issues	62
<i>Data Loss</i>	62
<i>Database Corruption</i>	63
Ping-Ponging.....	64
Data Collisions.....	66
COLLISION AVOIDANCE	67
Partitioned Database	67
Synchronous Replication	68
COLLISION DETECTION.....	68
Before-Image Comparison.....	69
Versioning.....	69
COLLISION RESOLUTION	69
Generic Algorithms.....	69
Data Field Specific Algorithms	70
Relative Replication.....	70
Fuzzy Replication	71
Manual Resolution.....	72
FAILURES AND RECOVERY	72
Failover	72
<i>Source Node Failure</i>	72
<i>Target Node Failure</i>	73
<i>Network Failure</i>	73
Restoration	74
<i>Duplicate Transactions</i>	75
<i>Data Collisions</i>	75
WHAT'S NEXT?	76

Breaking the Availability Barrier

Chapter 4 - Synchronous Replication	77
REPLICATING SYSTEMS	77
SPLITTING SYSTEMS	78
DATA COLLISIONS	80
SYNCHRONOUS REPLICATION	81
Dual Writes	82
Coordinated Commits	84
APPLICATION LATENCY.....	86
Dual Writes	87
Coordinated Commits	88
SYNCHRONOUS REPLICATION EFFICIENCY	89
SCALABILITY AND OTHER ISSUES.....	93
Scalability.....	93
<i>Multiple Database Copies.....</i>	<i>93</i>
<i>Communication Channel Efficiency.....</i>	<i>93</i>
Transaction Profile.....	94
Read Locks.....	95
Other Algorithmic Optimizations	95
EXAMPLES	96
Geographically Distributed Systems.....	96
Collocated Systems	98
EFFICIENCY MODEL EXTENSIONS.....	98
Dual Write Single Round Trip Operations.....	98
Dual Write Serial Updates	99
Plural Writes	101
DEADLOCKS	102
FAILURES AND RECOVERY	103
Dual (Plural) Writes.....	103
Coordinated Commits	103
<i>Failover.....</i>	<i>103</i>
<i>Restoration</i>	<i>104</i>
WHAT'S NEXT?.....	105
Chapter 5 - The Facts of Life	107
A REVIEW OF AVAILABILITY.....	108
WHY DO COMPUTERS STOP?.....	109
SOME DEFINITIONS.....	113

Breaking the Availability Barrier

TRIGGERED OUTAGES	114
THE IMPACT OF FAILOVER FAULTS	115
A Better Value for Subsystem Availability	117
Effect of Failover Faults on System Availability	118
Effect of Failover Faults on Effective Subsystem Availability ..	118
Effect of Failover Faults on System Splitting	119
THE GOLDEN RULE – REDUCE RECOVERY TIME	120
THE IMPORTANCE OF RESTORE TIME	122
WHAT’S NEXT?	124

Chapter 6 - RPO and RTO 125

REPLICATING THE SYSTEM.....	125
RTO AND RPO	126
REPLICATING THE APPLICATION DATA	128
No Replication	129
<i>No Replication, Periodic Backup Only.....</i>	<i>130</i>
<i>No Replication, Periodic Backup with Audit Trail.....</i>	<i>130</i>
Unidirectional Replication	133
<i>Unidirectional Replication – Cold Standby</i>	<i>133</i>
<i>Unidirectional Replication – Warm or Hot Standby</i>	<i>133</i>
Bi-Directional Replication	133
Active/Active Replication.....	134
<i>Partitioned Active/Active Replication.....</i>	<i>135</i>
<i>Asynchronous Active/Active Replication</i>	<i>136</i>
<i>Synchronous Active/Active Replication</i>	<i>136</i>
RECOVERY TIME	137
No Replication	137
<i>No Replication, Periodic Backup Only.....</i>	<i>137</i>
<i>No Replication, Periodic Backup with Audit Trail.....</i>	<i>139</i>
Unidirectional Replication	139
<i>Unidirectional Replication – Cold Standby</i>	<i>139</i>
<i>Unidirectional Replication – Warm or Hot Standby</i>	<i>140</i>
Bi-Directional Replication	140
<i>Hot Standby</i>	<i>140</i>
<i>Partitioned Active/Active Replication.....</i>	<i>140</i>
<i>Asynchronous Active/Active Replication</i>	<i>141</i>
<i>Synchronous Active/Active Replication</i>	<i>141</i>

Breaking the Availability Barrier

DATA LOSS.....	141
No Replication	142
Asynchronous Replication	142
Synchronous Replication	142
RECOVERY STRATEGIES	143
Unidirectional Replication	144
<i>Asynchronous Replication</i>	144
<i>Synchronous Replication</i>	144
Active/Active Replication.....	145
<i>System Failure</i>	145
<i>Network Failure</i>	145
COMPARISON SUMMARY.....	146
MULTI-NODE APPLICATIONS.....	150
RECOVERY DECISION TIME.....	150
SUMMARY	151
WHAT'S NEXT	152

Chapter 7 - The Ultimate Architecture 155

AN AVAILABILITY REVIEW.....	156
THE STRAWMAN SYSTEM.....	158
SPLITTING INTO INDEPENDENT SYSTEMS	159
SYSTEM SPLITTING WITH DUAL DATABASES	162
DO WE NEED TO REPLICATE A MIRRORED DATABASE?..	163
Option 1: Split Mirrors.....	165
Option 2: Network Storage	165
THE ULTIMATE ARCHITECTURE.....	167
PERFORMANCE IMPACT OF SYNCHRONOUS REPLICATION	
.....	169
HERE COME LOCAL CLUSTERS	170
DATABASE REPLICATION – ENHANCEMENTS WANTED..	171
CONCLUSION.....	172

Chapter 8 - The Rules of Availability 175

Breaking the Availability Barrier

Part 2 - Advanced Topics.....	179
Chapter 9 - Data Conflict Rates	181
SYNCHRONOUS VERSUS ASYNCHRONOUS REPLICATION	182
DEADLOCKS AND COLLISIONS	184
Deadlocks.....	185
<i>Mutual Waits</i>	<i>185</i>
<i>Lock Latency</i>	<i>186</i>
<i>Intelligent Locking Protocols.....</i>	<i>186</i>
Collisions	186
THE MODEL	188
MODEL SUMMARY	190
Synchronous Replication	191
<i>Mutual Waits</i>	<i>191</i>
<i>Lock Latency</i>	<i>191</i>
Asynchronous Replication.....	192
<i>Transactions Sent Serially After Commit.....</i>	<i>192</i>
<i>Transactions Broadcast After Commit.....</i>	<i>192</i>
<i>Modifications Sent Serially After Application</i>	<i>192</i>
<i>Modifications Broadcast After Application</i>	<i>192</i>
<i>Modifications Broadcast Upon Receipt</i>	<i>192</i>
MUTUAL WAIT DEADLOCKS.....	193
REPLICATION CONFLICTS	197
Collisions Under Asynchronous Replication.....	197
<i>Transactions Sent Serially After Commit.....</i>	<i>199</i>
<i>Transactions Broadcast After Commit.....</i>	<i>200</i>
<i>Modifications Sent Serially After Application</i>	<i>200</i>
<i>Modifications Broadcast After Application</i>	<i>201</i>
<i>Modifications Broadcast Upon Receipt</i>	<i>201</i>
Deadlocks Under Synchronous Replication	201
<i>Mutual Waits</i>	<i>201</i>
<i>Lock Latency</i>	<i>202</i>
<i>Collisions, Waits, and Deadlocks</i>	<i>203</i>
<i>Combined Effects.....</i>	<i>205</i>
<i>Deadlock Resolution</i>	<i>207</i>
PROCESSING NODES AND DATABASE NODES	210

Breaking the Availability Barrier

NOT ALL ACTIONS COLLIDE	212
FILE/TABLE HOT SPOTS	213
EXAMPLES	213
Synchronous Replication	214
<i>Dual Writes</i>	214
<i>Coordinated Commits</i>	214
Asynchronous Replication	215
Chapter 10 - Referential Integrity	217
BACKGROUND	217
System Replication	217
Data Replication	220
<i>Early Systems</i>	221
<i>Asynchronous Replication</i>	222
<i>Synchronous Replication</i>	223
<i>Physical Replication</i>	224
<i>Asynchronous Replication Issues</i>	225
Transactions	226
Simple Data Replication Model	229
Natural Flow	232
Referential Integrity	234
Current Data Replication Architectures	238
<i>Single-Threaded Replication Engine</i>	238
<i>Single-Threaded Replication Engine With DOC</i>	240
<i>Multi-Threaded Replication Engine</i>	241
Summary	242
MULTI-THREADING FOR PERFORMANCE	243
MULTI-THREADED EXTRACTOR	245
RULES-BASED EXTRACTOR ASSIGNMENT	245
Inter-Extractor Coordination	249
MULTI-THREADED COMMUNICATION CHANNEL	252
MULTI-THREADED APPLIER	254
Suspend on Commit	254
Appliers Coordinate Commits	256
Using a DOC	257
EXCEEDING TRANSACTION COUNT LIMITS	258
Multiple Appliers	259

Breaking the Availability Barrier

Partial Transactions.....	260
Adaptive Replication Engine	261
RESOLVING DEADLOCKS	262
Deadlocks With An Application.....	262
Deadlocks With Another Applier	262
Deadlocks With Another Transaction.....	263
SUMMARY.....	265
Chapter 11 - Failover Faults	271
FAILOVER FAULTS	272
REPAIR STRATEGIES	273
NO FAILOVER FAULTS.....	273
FAILOVER FAULT REQUIRING SUBSYSTEM REPAIR.....	275
FAILOVER FAULT REQUIRING SYSTEM RECOVERY	276
FAILURE MODEL SUMMARY.....	277
AN INTERPRETATION.....	278
Effective Subsystem Availability	278
System Availability Degradation.....	281
Splitting Systems	281
The Importance of System Recovery Time	285
MARKOV MODELING	286
Appendices	291
Appendix 1 - Availability Relationships	293
AVAILABILITY.....	294
SYSTEM SPLITTING	295
SYNCHRONOUS REPLICATION	295
FAILOVER FAULTS	297
DATA CONFLICT RATES	298
Appendix 2 - Availability Approximation Analysis	301

Breaking the Availability Barrier

Appendix 3 - Failover Fault Models	305
GENERAL	305
NO FAILOVER FAULT	308
Parallel Repair	308
Sequential Repair	311
Simultaneous Repair	314
FAILOVER FAULT REQUIRING SUBSYSTEM REPAIR	317
Parallel Repair	317
Sequential Repair	320
Simultaneous Repair	323
FAILOVER FAULT REQUIRING SYSTEM RECOVERY	326
Parallel Repair	326
Sequential Repair	330
Simultaneous Repair	333
Appendix 4 - Implementing a Data Replication Project	337
OVERVIEW	337
ANALYSIS OF NEEDS	338
EVALUATION OF OPTIONS	341
Building a custom solution	341
Buying a complete or semi-complete solution	342
COTS Data Replication Products	342
Distinguishing Criteria	343
RESOLUTION OF CONCERNS	344
PROJECT IMPLEMENTATION	345
COMPANY LISTING	346
References and Suggested Reading	349
Index	353

Forward

There is an old saying in business. You can optimize schedule, cost, and quality. Pick any two.

When it comes to configuring your data processing system, there is an equivalent saying. You can optimize performance, cost, and availability. Pick any two.

We typically configure our systems for performance and cost and let availability fall where it may. Even so, we are achieving impressive availabilities by industry standards. Typically, highly available systems such as HP NonStop Servers will fail only about once every five to ten years and will then be down for an average of about four hours. This means that our systems are up 99.99% of the time and are delivering four 9s of availability to our businesses.

When we talk about availability, we mean that all services upon which our businesses depend are available. “Available” means not only just working but also working at a performance level that makes our services useful. When sub-second response is expected, a multi-second response may be no better than no response at all.

Are four 9s good enough for you? You probably have faced what an outage of several hours means to your business. If you are running a Web service, this could mean irate customers, lost sales, and perhaps lost customers. If you are providing banking services, this may result in large fines by governmental authorities. A stock market trading system outage could make international headlines. The failure of a critical emergency service such as a “911” system in the U.S. could contribute to death by a cardiac arrest or a building burned to the ground.

How would you like to improve the availability of your systems so that the loss of any significant capacity is measured in terms of centuries rather than years – *at little or no additional cost*? As an

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

added plus, your systems could tolerate major disasters such as floods, fires, earthquakes, and terrorism as well as environmental failures which may take out your power or air conditioning. Getting more availability for your system expenditures is what this book is all about.

This book is separated into two Parts. Part 1, “*Breaking the Four 9s Barrier*,” discusses the availability characteristics of today’s systems and how to economically reconfigure these systems via distribution in order to dramatically improve availability. Part 2, “*Advanced Topics*,” presents a more detailed analysis of some of the considerations in distributing systems for improved availability.

The authors’ intended audience includes IT executives who feel that they must reduce the down time of their systems, the system architects and senior developers who must build these systems or modify existing systems to achieve the required availability, and the operations staff who must then run these systems and recover from system faults. Many of the concepts presented are supported by mathematical arguments. However, for those who consider themselves mathematically challenged, we have tried to make clear the concepts being formulated so that the mathematics can, in fact, be glossed over.

Part 1 – Breaking the Four 9s Barrier

The availability of a system is directly related to the number of ways in which it may fail – its failure modes. We show in Chapter 1, “*The 9s Game*,” that the number of failure modes can be reduced significantly by paying attention to how we allocate critical processes to processors.

We show in Chapter 2, “*System Splitting*,” that failure modes can be further reduced by splitting a system into several smaller, independent nodes. This strategy not only dramatically improves availability but also, when a node outage does occur, guarantees that only the capacity provided by that node is lost. Furthermore, the

chance of losing the capacity provided by two or more nodes is virtually never.

As an added advantage, splitting a system into several nodes allows you to do upgrades and maintenance a node at a time, virtually eliminating planned down time.¹

However, nothing comes for free. If we split a system into several cooperating nodes, the system database must also be distributed across these nodes. Providing many duplicate copies of the database can be very expensive as often the cost of the database subsystem represents a majority of the system cost. Chapter 2 also shows how we can distribute a system across geographically dispersed nodes without suffering additional database costs.

Inherent in splitting a system into two or more independent nodes with duplicate copies of the database is that these databases must be kept in near synchronization. Chapter 3, “*Asynchronous Replication*,” shows how to do this by replicating data changes from the source node that is making the changes to the other database copies in the application network so that they can be updated with these same changes. We also explore in this chapter many advantages and several issues with asynchronous replication.

A particularly severe problem with asynchronous replication in some applications is data collisions that lead to database contamination. This occurs when two users at different locations simultaneously update the same data item on different database copies. The result is inconsistent data propagated across the network. Data collisions can happen with surprising frequency and often require manual intervention to resolve. The only general solution to this predicament is to avoid data collisions.

¹ It is interesting to note that one of the authors identified a need for a working peer-to-peer replicated system almost a decade ago. See Highleyman, W. H., “*Distributing OLTP Data Via Replication*,” Tandem Connection, Volume 16, Issue 2; April/May, 1995.

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

Chapter 4, “*Synchronous Replication*,” describes how to avoid data collisions by ensuring that distributed copies of a database are kept in exact synchronism by the replication of updates across the network as atomic transactions. However, doing so can increase transaction response time because of communication network delays. The performance impact of different methods for synchronous replication is evaluated, and we show that the method of choice depends not only upon whether the system nodes are collocated or geographically dispersed but also upon the length of the transaction.

Distributed transactions (such as HP NonStop’s network TMF) are generally appropriate for short transactions within a collocated distributed system. For geographically distributed systems or for large transactions, the use of asynchronous data replication with coordinated commits of transactions at each of the nodes is more efficient. In either event, the performance impact is more than overcome by the increased speed of new systems that are being introduced today.

Up until now, we have considered redundant systems whose outages are caused by multiple hardware failures. It turns out that more important factors are software faults and operator errors. Chapter 5, “*The Facts of Life*,” extends the availability concepts to include these sources of outages. Here we stress the importance of fast recovery from an outage and point out that this is not only a technical issue but also more importantly a serious business process issue.

As in every facet of life, there are trade-offs and compromises. When it comes to availability, two important considerations are the time that it will take for a system to recover from a failure and the amount of data that may be lost due to a failure. Every organization must grapple with their own tolerance to recovery time and data loss and set objectives for what can be tolerated. The corporate objective for recovery time is known as the Recovery Time Objective, or RTO. The corporate objective for acceptable data loss is called the Recovery Point Objective, or RPO. There are a variety of technologies today that allow one to replicate a system, and each has

its own RTO and RPO characteristics. These various technologies are reviewed in Chapter 6, “*RTO and RPO.*”

In Chapter 7, “*The Ultimate Architecture,*” we put all we have learned into configuring systems that can meet the following objectives:

- The frequency of losing more than a tolerable amount of capacity is measured in centuries.
- The system can be distributed geographically for disaster tolerance.
- The reconfigured system will incur little if any additional cost.
- Reconfiguring for availability is non-intrusive. It does not require application rewrites.
- By and large, the facilities for achieving these objectives are available today.

These are tough objectives indeed, and certainly there will be some cost and performance impacts. But are these impacts worth the significantly enhanced availability that can be achieved? Only you can answer that question.

Remember – a system that is down has zero performance and perhaps an incalculable cost.

Throughout these chapters in Part 1, a variety of rules that relate to very high availability systems are set forth. These rules are summarized in Chapter 8, “*The Availability Rules.*” Perusing these rules is an excellent review and summary of the concepts presented in Part 1.

Part 2 – Advanced Topics

Several concepts revolving around distributed systems are introduced in Part 1 of this book. But distributed systems pose many problems. Some of these are analyzed in more detail in the advanced topics of Part 2.

One of the problems in distributing an application and its database across multiple systems is that data collisions which contaminate the database may occur. A possible solution to this problem is to permit data collisions, then detect and resolve them as discussed in Chapter 3, “*Asynchronous Replication.*” As shown there, data collisions can in some instances be resolved automatically but in other cases must be resolved manually.

Another solution to this problem is to avoid data collisions by using synchronous replication, as shown in Chapter 4. However, this solution may present a performance challenge. In addition, if the applications in such an environment are not structured properly, deadlocks may occur when two or more applications attempt to lock copies of a data item, or attempt to lock the same set of data items in different orders.

Chapter 9, “*Data Conflict Rates,*” gives an analysis of data collision rates and deadlock rates that may help you decide whether you can tolerate data collisions in your application. It is shown that deadlocks are orders of magnitude less likely than data collisions.

Another potential problem in distributed databases is the violation of referential integrity rules if transactions are not replicated in natural flow order. The satisfaction of natural flow requires that transactions, and indeed the database modifications contained within the transactions, must be applied in the same order at the target databases as they were at the source database. Otherwise, for instance, a child entity might be added to the database before a parent existed; or a later update might be overwritten by an earlier update. Chapter 10,

“*Referential Integrity*,” explains this problem and explores various data replication engine architectures and their ability to maintain referential integrity.

A major impact on the availability improvement that can be obtained by distributing an application is the fact that the automatic recovery from a fault may itself fail. This is called a *failover fault*. Failover faults are discussed in Chapter 5, “*The Facts of Life*,” where a cursory analysis of the impact of such faults on various aspects of system availability is presented. Chapter 11, “*Failover Faults*,” provides a more formal analysis of these impacts.

A fundamental requirement for increasing system availability by splitting a system into multiple independent nodes is a robust data replication engine that can support low-latency bi-directional replication, whether one is using a synchronous or an asynchronous approach. The authors have extensive experience in the field of data replication and can be contacted via the e-mail addresses given in their biographies for further information.

Appendices

As we progress through the book, we derive many useful relationships regarding the availability of distributed systems. These relationships are repeated and conveniently organized in Appendix 1 for ready reference.

Many of the relationships depend upon a fairly simple approximation given in Chapter 1, “*The 9s Game*,” to the dependency of system availability on subsystem availability, subsystem mean time before failure, subsystem mean time to repair, and the level of sparing. Such approximations are used to simplify the analysis and to give a clearer view of the concepts being presented. A formal analysis of these relationships is given in Appendix 2, “*Availability Approximation Analysis*,” where it is shown that the approximations not only are conservative but also are generally within 5% of the actual values over the range of parameters that are of interest to us.

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

The Markov Chain models which support the various failover fault cases analyzed in Chapter 11 are detailed in Appendix 3, “*Failover Fault Models.*”

Finally, Appendix 4 provides a methodology for implementing a data replication project and lists many of the data replication products that are available as of the writing of this book.

Authors’ Notes

In many places throughout this book, reference is made to HP NonStop systems. NonStop systems were originally developed by Tandem Computers to provide very high availability. Tandem Computers was subsequently acquired by Compaq Computers, and Compaq was then acquired by HP. HP has changed the name of the Tandem systems to HP NonStop servers. The authors have considerable experience with these systems. However, concepts and recommendations presented in this book are extendable to all types of redundant systems, including HP Superdome, Windows Server clusters, UNIX clusters, and IBM Sysplex systems.

It should be noted that in the NonStop world, some important characteristics of fault-tolerant systems are summarized by the acronym RAS. RAS stands for reliability, availability, and scalability. In the RAS context, “reliability” is taken to mean “data integrity.” For purposes of this book, the term *reliability* is used in the more general sense of system reliability. System reliability is associated with system availability.

Part 1 of this book is, in part, a compilation of a six-part series of papers published in The Connection, starting with the November/December, 2002, issue. The Connection is the official publication of ITUG, The International HP NonStop Users’ Group. These papers have been modified slightly to fit the format of this book and include certain corrections. The originals may be found in the archives of The Connection at www.itug.org.

Each of the chapters in this book has been written to be self-standing at the risk of some repetition. Therefore, the reader is

encouraged to pick and choose the topics of interest and to read only those chapters that apply. Adequate reference is made to other chapters to suggest further reading.

Acknowledgements

Breaking The Availability Barrier has benefited from reviews by many people. We gratefully acknowledge their guidance and especially thank Carl Niehaus, Wendy Bartlett, Alan Wood, Julie Scherer, and Gary Strickler for their in-depth evaluations and helpful criticism of various sections. We also thank Burt Liebowitz and John Carson whose book Multiple Processing Systems for Real-Time Applications was the inspiration for this work, and Jim Gray whose many writings fueled the fire. They and others who have influenced the book include:

Wendy Bartlett at HP	ITUG <u>Connection</u> staff
Richard Buckle at Insession	Clark Jablon at Akin Gump
Victor Berutti at Gravic	Bill Knapp at Gravic
Ron Byer, Jr., at NetWeave	Ron LaPedis at HP
Gary Chatterton at Sombers	Burt Liebowitz, Consultant
Robert Cline at JPMI	Malcolm Mosher at HP
Dick Davis at Gravic	Mike Nee at NTI
John Dennis at HP	Carl Niehaus at HP
Jim Gray at Microsoft	Janice Reeder at Sombers
John Hoffmann at Sombers	Julie Scherer at Gravic
Bill Holenstein at Gravic	Paul Siegel at Sombers
Denise Holenstein at Gravic	Gary Strickler at Gravic
Dan Hoppmann at A. G. Edwards	Mark Waterstraat at Insession
Ron Hopson at NetWeave	Alan Wood at HP

About the Authors

Dr. Wilbur H. (Bill) Highleyman brings more than forty years experience in the design and implementation of computer systems to his position as Chairman of The Sombers Group, Inc. SGI is a turnkey

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

custom software house specializing in the development of real-time, on-line data processing systems, with particular emphasis on fault-tolerant systems and large communications-oriented systems. He is also Chairman of NetWeave Corporation, which developed the middleware product NetWeave. NetWeave is used to integrate heterogeneous computing systems at both the messaging and the database levels. Dr. Highleyman, a graduate of Rensselaer Polytechnic Institute and MIT, earned his doctorate in electrical engineering from Polytechnic Institute of Brooklyn. He has published extensively on availability, performance, middleware, and testing and is the author of "Performance Analysis of Transaction Processing Systems," published by Prentice-Hall. He holds four patents and can be reached at billh@somers.com.

Paul J. Holenstein is Executive Vice President of Gravic, Inc., the makers of the ITI Shadowbase line of data replication products. Shadowbase is a low latency, high performance real-time data replication engine that provides disaster recovery as well as heterogeneous data transfer. Mr. Holenstein has more than twenty-two years of experience providing architectural designs, implementations, and turnkey application development solutions on a variety of UNIX, Windows, and VMS platforms, with his HP NSK experience dating back to the NonStop I days. He was previously President of Compucon Services Corporation, a turnkey software consultancy. Mr. Holenstein's areas of expertise include high-availability designs, data replication technologies, disaster recovery planning, heterogeneous application and data integration, communications, and performance analysis. Mr. Holenstein, an HP-certified Accredited Systems Engineer (ASE), earned his undergraduate degree in computer engineering from Bucknell University and a master's degree in computer science from Villanova University. He has co-founded two successful companies and holds patents in the field of data replication. He can be reached at shadowbase@iticsc.com.

Dr. Bruce D. Holenstein is President and CEO of Gravic, Inc. Gravic's ITI Shadowbase software supports many of the architectures described in this book and operates on systems such as UNIX, Windows, NonStop and other platforms running databases including

Breaking the Availability Barrier

Oracle, Sybase, DB/2, and SQL/MP. Dr. Holenstein began his career in software development in 1980 on a Tandem NonStop I. His fields of expertise include algorithms, mathematical modeling, availability architectures, data replication, pattern recognition systems, process control, and turnkey software. Dr. Holenstein earned his undergraduate degree in Electrical Engineering from Bucknell University and his doctorate from the University of Pennsylvania. Dr. Holenstein has co-founded and run three successful companies and holds patents in the field of data replication. He can be reached at shadowbase@iticsc.com.

Dr. Bill Highleyman
Paul J. Holenstein
Dr. Bruce Holenstein
Paoli, Pennsylvania
July, 2003

Part 1 - Breaking the Four 9s Barrier

Chapter 1 - The 9s Game

True or False:

- *Adding processors to a multi-processor redundant system increases its reliability.*
- *A 16-processor redundant system has the reliability of a UNIX box.*
- *A fault-tolerant processor board is less reliable than a UNIX processor board.*

The above are indeed provocative questions, and their answers are not straightforward. This chapter develops some simple, though not necessarily intuitive, concepts that help answer these and other questions. More importantly, such concepts lead to some straightforward steps that you can take to improve the reliability of your system – steps as simple as being aware of how you allocate processes to processors.

What is Reliability?

The questions above all refer to *reliability*. But before we go much further, we have to agree on how to measure reliability.

There are actually two components that impact the reliability of a system – how long it will work before it fails and then how long it

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

will take to fix. We call the first component the *mean time before failure* (MTBF)² and the second *the mean time to repair* (MTR).

To a space satellite designer, reliability is MTBF. Once the satellite fails, it is gone forever. It is not repairable. Clearly, a satellite with a ten-year MTBF is ten times more reliable than a satellite with a one-year MTBF.

However, when it comes to life and property protection, reliability is MTR. In a 911 system, an outage of 30 seconds may be simply an aggravating hiatus; but an outage of one hour can mean death by cardiac arrest or a building burned to the ground.

In large transaction processing systems, reliability is often measured as down time. Down time has a cost associated with it – perhaps \$1,000 per hour or \$100,000 per hour. (Of course, MTR plays a role here as well; the longer the down time, the higher the cost in many cases – from customer annoyance to lost sales to lost customers.) Down time alternatively can be measured as the proportion of time that a system is up, a measure that we call the *availability* of the system. Since the system is always either up or down, then

$$\text{availability} = A = \frac{\text{MTBF}}{\text{MTBF} + \text{MTR}} \quad (1-1a)$$

Note that this also can be written as

$$A = \frac{1}{1 + \frac{\text{MTR}}{\text{MTBF}}} \approx 1 - \frac{\text{MTR}}{\text{MTBF}} \quad (1-1b)$$

² MTBF is often called Mean Time Between Failures. In this sense, it is the average time from one failure to the next failure. However, we use MTBF in the context of the average time from the time that a component is returned to service to the time of its next failure. Therefore, we call MTBF the Mean Time *Before* Failure. Thanks to Dr. Alan Wood of HP for pointing out this subtle but important distinction in semantics.

where “ \approx ” means “approximately equal to,” and the approximation is valid so long as MTBF is very much greater than MTR (which certainly is true in the cases that we will be considering).

It is availability that this chapter is all about. When we speak of reliability, we mean availability. A system with .999 availability is more reliable than a system with .99 availability.

More specifically, we will compare the reliability of systems by comparing their probabilities of failure. If a system has an availability of .99, then it has a probability of failure of 1-.99, or .01. That is, it will be down 1% of the time. Using Equation (1-1b),

$$\text{probability of failure} = F = 1 - A \approx \frac{\text{MTR}}{\text{MTBF}} \quad (1-2)$$

Jim Gray has characterized availability and reliability in a very folksy way:³

“Availability is doing the right thing within the specified response time. Reliability is not doing the wrong thing.”

As outlined above, we apply measures to these:

$$\begin{aligned} \text{Availability} &= \text{Doing the right thing} = A \\ \text{Failure} &= \text{Doing the wrong thing} = F = (1-A) \\ \text{Reliability} &= \text{Not doing the wrong thing} = 1/(1-A) \end{aligned}$$

Consider System A, which is up 99% of the time, and System B, which is up 99.9% of the time. System A has an availability of .99, a failure probability of .01, and a reliability measure of 100. System B has an availability of .999, a failure probability of .001, and a reliability measure of 1000. Thus, we say that System B is ten times more reliable than System A.

³ Gray, J.; “*Why Do Computers Stop and What Can We Do About It?*” 5th Symposium on Reliability in Distributed Software and Database Systems; 1986.

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

This is how we will use the term *reliability* throughout this chapter.

Some Caveats

There is some algebra used in this chapter to develop availability concepts. About the worst relationship looks like

$$A \approx 1 - f(1 - a)^{s+1}$$

If you are algebraically challenged, do not despair. Just skip the math and grasp the concept. The concepts are clearly stated and don't depend upon the math for understanding. Besides, charts and tables are provided so that you can use these relationships without ever breaking out a calculator.

Also, we are interested in developing concepts and rules of thumb. This requires that we take a simplistic view of things. You will be tempted to say, "Yes, but my system does this" or "You haven't considered that." True, but we are taking a 50,000 foot view of things in order to develop some general concepts. Moreover, the concepts developed will allow those of you who are motivated to drop down to a 5,000 foot view. A 500 foot view, however, is probably obscured by a lack of good data and too many trees in your way.

The simplistic view presented in this chapter is most applicable to repairable systems. Software failures generally are recovered rather than repaired. They are more complex and are considered in Chapter 5.

9s – The Measure of Availability

When we calculate availability for today's systems, we will get numbers like .99999. Saying this gets cumbersome and can lose the meaning. So we talk about availability in terms of the number of 9s.

“.99999” is “five 9s.” “.998” is “a little less than three 9s.” “.99992” is “a little more than four 9s.”

Though we will speak of 9s, this measure can be converted to average down time over any given period, as shown in Table 1-1.

Nines	% Available	Hours/Year Down Time	Minutes/Month Down Time
2	99%	87.60	438.
3	99.9%	8.76	43.8
4	99.99%	.88	4.38
5	99.999%	.09	.44
6	99.9999%	.01	.04

**Average 24x7 Down Time
Table 1-1**

Of course, an availability of three 9s does not mean that the system will be down 8.76 hours each year. It means that over a sufficiently long period of time, one can expect that the system will be down an average of eight or nine hours per year. This could occur as short 1 minute failures every 17 hours or as a one-day failure every three years.

Specifically, knowing the availability tells us nothing about the MTBF or MTR. But knowing the availability and either the MTBF or the MTR tells us the other. More to the point, from Equation (1-1b) we can deduce that

$$\text{MTBF} \approx \text{MTR} / (1 - A) \quad (1-3)$$

$$\text{MTR} \approx \text{MTBF} (1 - A) \quad (1-4)$$

Thus, if we know that our availability is three 9s, and if we have an MTR of 4 hours, then we have an MTBF of 4,000 hours.

Today's Systems

How do today's systems rate so far as availability is concerned? Results compiled by the Gartner Group⁴ indicate the following availabilities:

HP NonStop™	.9999
Mainframe	.999
Open VMS	.998
AS400	.998
HPUX	.996
Tru64	.996
Solaris	.995
NT Cluster	.992 - .995

Thus, mainframes are four to five times more reliable than UNIX systems, and HP NonStop systems are ten times more reliable than mainframes.

Simple Systems

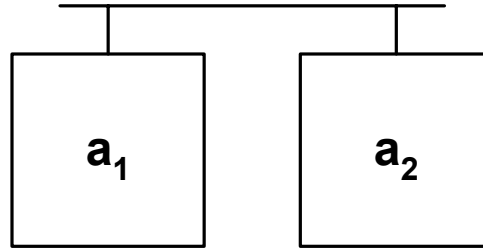
Let us start our conceptual journey by looking at the two simplest systems and by reviewing a little probability theory along the way.

We consider a system made up of subsystems, each subsystem with an availability of a . The availability of the entire system is A .

Non-Redundant System

Figure 1-1 shows a system comprising two non-redundant subsystems. Both must work in order for the system to work.

⁴ Gartner Group; 2002.



Non-Redundant System
Figure 1-1

Let the availability of subsystem 1 be a_1 and of subsystem 2 be a_2 . Remember that each of these availabilities is the probability that the subsystem will be operational. In order that the entire system be operational, both subsystem 1 *and* subsystem 2 must be operational. The probability of this is the product of the component probabilities:

$$A = a_1 a_2 \quad (1-5)$$

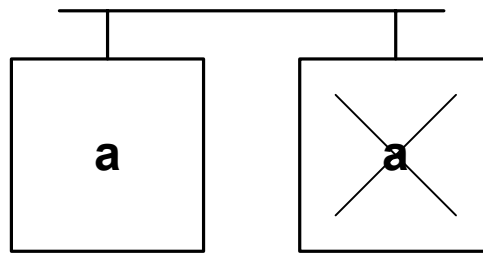
Rule 1: *If all subsystems must be operational, then the availability of the system is the product of the availabilities of the subsystems.*

Redundant System

Figure 1-2 shows a redundant system comprising two identical subsystems, but in this case the system is operational if either subsystem 1 is operational *or* if subsystem 2 is operational. In order for the system to be down, both subsystem 1 *and* subsystem 2 must be down. Since the probability that either subsystem will be down is $(1-a)$, then the probability that both will be down is $(1-a)^2$. The system availability is therefore

$$A = 1 - (1-a)^2 \quad (1-6)$$

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein



Redundant System

Figure 1-2

Double Your 9s

Let us explore Equation (1-6) a little further. If the subsystem availability a is .99, and if we add an additional subsystem so that we have a spare subsystem, then the system availability A of the resulting redundant subsystem is

$$A = 1 - .01 \times .01 = .9999$$

Note that we have doubled the 9s from a subsystem availability of two 9s to a system availability of four 9s. This is the beauty of redundant systems.

Rule 2: *Providing a backup doubles the 9s.*

This is the basis for the high reliability of fault-tolerant systems and for the even higher reliability that can be achieved by replicating a system using data replication techniques (more about that in Chapter 2).

The Real Fault-Tolerant World

Redundant systems are the basis for the high availability (and high scalability as well) of fault-tolerant systems. But these systems are a bit more complex than the simple systems that we have just considered:

- They comprise multiple redundant subsystems – processors, processes, disks, communications.
- Processes critical to system operation are replicated as process pairs.
- Processes are distributed randomly across processors (usually to satisfy load balancing considerations).

Consistent with our 50,000 foot view, we will consider a fault-tolerant system as a single group of like subsystems. Therefore, a subsystem is a processor and its collection of disks and other peripherals. Certainly in a real system, each subsystem will be somewhat different since different processors have associated with them different numbers of devices; but our assumption that all subsystems are similar is warranted by the simplifications that allow us to develop some general concepts.

Furthermore, we will assume a subsystem availability of .995. This is close to the K-series subsystem availability of .996 reported to one of the authors by Tandem in the mid-1990s. Today's systems undoubtedly comprise more reliable components and are manufactured using higher quality techniques, but they are also more complex. It is therefore assumed that a subsystem availability of .995 is still in the ballpark.

Note that this value for availability includes all sources of failure: hardware, software, maintenance, and operations. More about that in Chapter 5.

Randomly Distributed Process Pairs

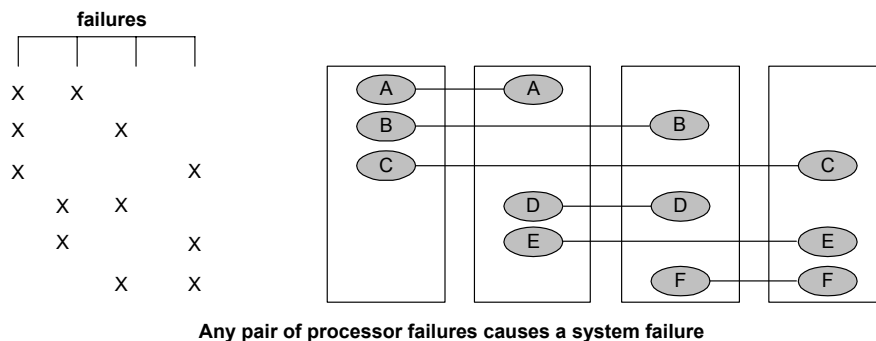
So far as availability is concerned, the heart of a fault-tolerant system is its critical processes. The loss of any one of these processes will cause a system failure, either immediately or after a short period of time due to system degradation. For instance, in HP NonStop

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

servers, critical processes include the disk processes (DP2), PATHMON, and a slew of monitors for communication and other subsystems.

Therefore, these processes are provided as process pairs so that they will survive any single processor failure. Coupled with transaction protection that guarantees that no data will be corrupted as a result of a fault, these features provide the high availability for which fault-tolerant systems are known.

But a dual processor failure may take down a critical process pair and result in a system outage. Let us take a look at a four processor system in which critical processes are randomly distributed across all processors so that any dual processor failure will take down the system (Figure 1-3).



Any pair of processor failures causes a system failure

Randomly Distributed Processes Figure 1-3

Note that there are six possible ways that two out of four processors can fail. We call these *failure modes*. Since any given two processors will fail with a probability of $(1-a)^2$, and since there are six ways that this can happen, then the system will fail with a probability of $6(1-a)^2$. Thus, its availability is⁵

⁵ This relation is an approximation since it does not account for failure modes involving more than two subsystems. Since the probability of three or more failures is extremely small, the relation is quite accurate. Besides, it's a lot simpler than the fully accurate relation.

$$A \approx 1 - 6(1 - a)^2$$

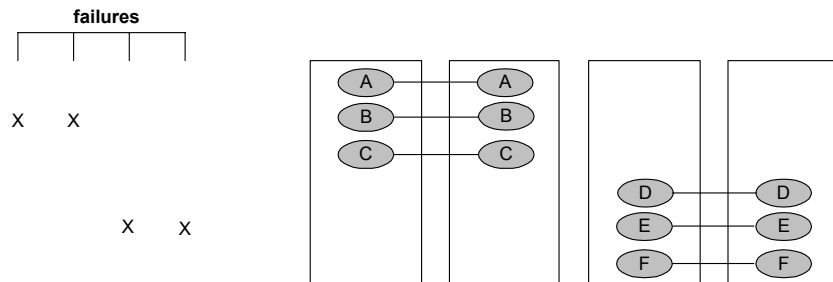
You can probably figure out that for n processors, the number of failure modes is $n(n-1)/2$. For the above example, $n=4$. The number of failure modes is $4 \times 3 / 2 = 6$.

Process/Processor Pairing

Figure 1-4 shows an alternate strategy for distributing process pairs. Processors are organized into pairs, and process pairs are constrained to run only in processor pairs. For a four-processor system, there are only two failure modes. Either the first pair of processors must fail or the second pair must fail in order to cause a system failure. Thus, the availability of this configuration is

$$A \approx 1 - 2(1 - a)^2$$

This configuration has three times the reliability of the randomly distributed configuration. In general, for n subsystems, the number of failure modes is $n/2$ for this strategy.



Only certain pairs of processor failures cause a system failure

**Process Pairing
Figure 1-4**

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

In fact, the advantage of process/processor pairing gets better as the system gets larger. Consider an eight-processor system (where f is the number of failure modes):

	<u>n</u>	<u>a</u>	<u>f</u>	<u>A</u>
random	8	.995	28	.9993
paired	8	.995	4	.9999

To review the terminology in the above table,

n is the number of subsystems in the system,
 a is the subsystem availability,
 f is the number of failure modes, and
 A is the system availability.

For an 8-processor system, a paired configuration is seven times as reliable as a random configuration. For a 16-processor system, reliability is improved by a factor of fifteen!

Rule 3: *System reliability is inversely proportional to the number of failure modes.*

Rule 4: *Organize processors into pairs, and allocate each process pair only to a processor pair.*

Availability in General

We have seen above that failure probability is proportional to the number of failure modes. Thus, letting f be the number of failure modes, we may write

$$A \approx 1 - f(1 - a)^2$$

This relationship assumes that each process is backed up by only one other process (we can consider the backup process as a spare which is put into service if the primary process fails).

But what if we have two spares? Then any given failure occurs only with a probability of $(1-a)^3$. In general, if we have s spares, then the system will fail only if we have $s+1$ subsystem failures. Any particular failure of $s+1$ subsystems will occur with a probability of $(1-a)^{s+1}$, and the system availability becomes

$$A = 1 - F \approx 1 - f(1-a)^{s+1} \quad (1-7)$$

This is our general availability equation.⁶ Note that it reduces to our simple example represented by Equation (1-6) for $f=1$ and $s=1$.⁷ This result can be expressed as

Rule 5: *If a system can withstand the failure of s subsystems, then the probability of failure of the system is the product of the probability of failures of $(s+1)$ systems.*

There is one assumption that is inherent in our discussion so far, and that is that the system is returned to service as soon as a failed subsystem has been repaired. It needs no further recovery. This assumption is explored further in Chapter 5, “*The Facts of Life.*”

More Sparing

Note that from Equation (1-7), reliability increases exponentially with the number of spares. If a is .99, each additional level of sparing adds another two 9s to the system availability. Single sparing gives a system availability of four 9s, double sparing gives an availability of six 9s, and so on.

⁶ As we said earlier, this is an approximation. However, not only is it conservative in that it gives a lower value for A than the actual value, but it is within 5% for the range of values in which we are interested. See Appendix 2 for more on this.

⁷ This result is an extension of an excellent summary of availability found in Chapter 8, “*Reliability Calculations,*” Multiple Processing Systems for Real Time Applications by Burt H. Liebowitz and John H. Carson, Prentice-Hall; 1985.

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

Rule 6: *System availability increases dramatically with increased sparing. Each additional level of sparing adds a subsystem's worth of 9s to the overall system availability.*

How do we increase process sparing in fault-tolerant systems?
There are several methods:

- For process pairs (i.e., a primary process with a backup process waiting to take over in the event of a primary process failure), allow a process to start a new backup in a surviving processor if the process loses its backup due to a processor failure.
- For persistent processes that are restarted in another processor by a monitor, give the monitor the choice of more than two processors in which to restart the process (of course, the monitor must be redundant as well).
- Design the application to allow more than one active process or more than one backup process.

How Many Failure Modes?

The worst case for availability is a random distribution of processes, since in this case any pair of processor failures will take the system down. For n subsystems and s spares, the number of failure modes for this case is the number of ways that $s+1$ subsystems can fail out of n systems.⁸ These maximum values for f are shown in Table 1-2.

We can see from this table that the maximum number of failure modes for a single-spared 16-processor system is 120. However, we know that if we pair processors and processes, we can reduce the

⁸ For Math Nuts: This is $n!/(n-s-1)!(s+1)!$

number of failure modes to eight, a 15:1 reduction as we have earlier noted.

Since reliability is inversely proportional to the number of failure modes (Rule 3), we can lose more than a nine from our achievable availability for a 16-processor system if we are not careful with process allocation. More about this later.

		Processors (n)							
		2	4	6	8	10	12	14	16
Spares (s)									
0	2	4	6	8	10	12	14	16	
1	1	6	15	28	45	66	91	120	
2		4	20	56	120	220	364	560	
3		1	15	70	210	495	1001	1820	
4			6	56	252	792	2002	4368	
5			1	28	210	924	3003	8008	
6				8	120	792	3432	11440	
7				1	45	495	3003	12870	
8					10	220	2002	11440	
9					1	66	1001	8008	
10						12	364	4368	
11						1	91	1820	
12							14	560	
13							1	120	
14								16	
15									1

Maximum Failure Modes (f)
Table 1-2

The Impact of Repair Time

So far, we have talked about availability as the predominant measure of reliability. But as we indicated in the opening to this chapter, the system mean time to repair, MTR, is often an equally important parameter. Let us look at system MTR and its relation to subsystem mean time to repair, mtr.

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

From Equation (1-1a), we can express subsystem availability a in terms of its $mtbf$ and mtr :

$$a = \frac{mtbf}{mtbf + mtr}$$

where

- a is the subsystem availability.
- $mtbf$ is the subsystem mean time before failure.
- mtr is the subsystem mean time to repair.

Note that we are using upper case MTBF and MTR to represent the system and lower case $mtbf$ and mtr to represent the subsystem.

A little algebraic manipulation results in

$$1 - a = 1 - \frac{1}{1 + \frac{mtr}{mtbf}} \approx \frac{mtr}{mtbf} \quad (1-8)$$

The approximation depends upon $mtbf$ being much greater than mtr . This is certainly true by orders of magnitude in the systems that we are considering.

Substituting Equation (1-8) into Equation (1-7), we have

$$A = 1 - F \approx 1 - f \left(\frac{mtr}{mtbf} \right)^{s+1} \quad (1-9)$$

We see that reliability is exponentially affected by subsystem mtr . For one spare ($s = 1$), the system failure probability will be cut by a factor of four if we can cut subsystem mtr in half.

But how does subsystem mtr affect the overall system MTR and its MTBF? It can be shown⁹ that

⁹ If $(s+1)$ subsystems are being repaired independently, and if each requires an average time of mtr to repair, then they are being repaired at a rate of $(s+1)/mtr$.

$$\text{MTR} = \frac{\text{mtr}}{(\text{s} + 1)} \quad (1-10)$$

For one spare, system MTR is half the subsystem mtr. Thus, for one spare, if our subsystem mtr is four hours, then our system MTR is two hours. (This assumes that the repairs of multiple subsystems are independent of each other.)

Rule 7: *For a single spare system, the system MTR is one-half the subsystem mtr.*

Furthermore, if we reduce mtr by a factor of k , we will reduce MTR by a factor of k . Since we have seen that the failure probability will be reduced by k^2 (Equation (1-9)), then from Equation (1-2) we can conclude that we will increase our system MTBF by a factor of k .

Rule 8: *For the case of a single spare, cutting subsystem mtr by a factor of k will reduce system MTR by a factor of k and increase the system MTBF by a factor of k , thus increasing system reliability by a factor of k^2 .*

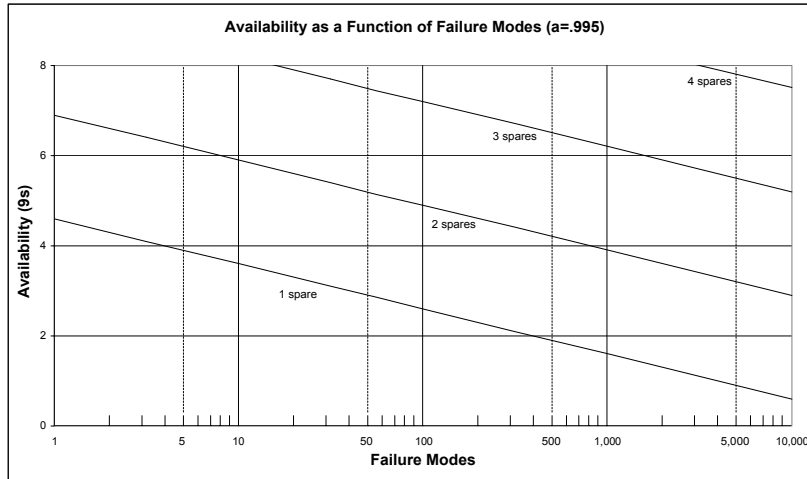
For instance, let us say that our system has an MTBF of five years and an mtr of four hours, leading to an MTR of two hours. If we can cut mtr in half to two hours, our system MTR will be reduced to one hour; and our system MTBF will be increased to ten years. Our reliability has increased by a factor of four, as indicated above.

Some Helpful Charts

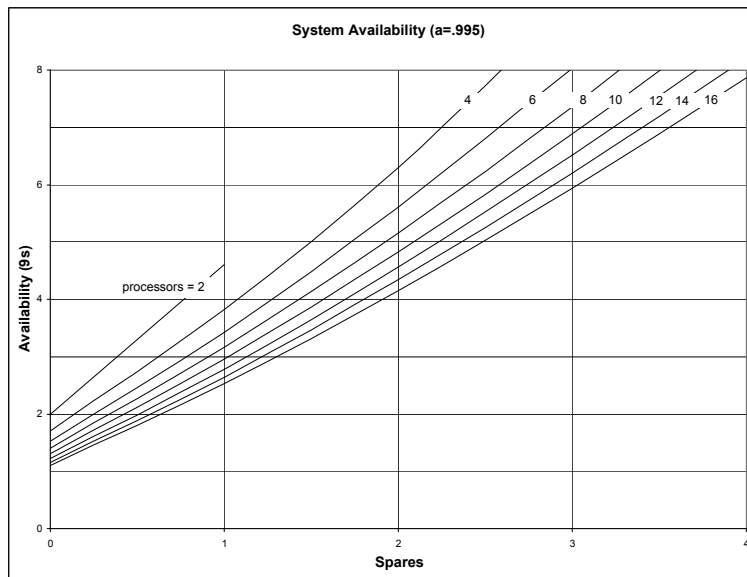
Figure 1-5 shows availability as a function of the number of processors and the number of spares for random distribution of processes. Note that no matter the number of processors, each additional spare adds about two 9s to the system availability.

Thus, the average time to the next repair, which will return the system to service, is $\text{mtr}/(\text{s}+1)$.

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein



Availability as a Function of Processors and Spares
Figure 1-5



Availability as a Function of Spares, Failure Modes
Figure 1-6

Figure 1-6 shows availability as a function of spares and failure modes. Let's look at process pairing and random distribution for 16 processors. For process pairing ($f=8$), system availability is almost four 9s. For random distribution ($f=120$), system availability is about two and a half nines. My! That's about the availability of a UNIX box.

Answers

Let's return to our initial provocative statements.

- *Adding processors to a multi-processor redundant system increases reliability.*

Well, it all depends. If sparing remains the same, then the number of failure modes increases; and reliability decreases. However, if the extra processors are used to increase sparing, then reliability can dramatically increase. However, since we don't generally change the design of the system when we add processors, this statement is typically false. Adding processors reduces reliability.

- *A 16-processor redundant system has the reliability of a UNIX box.*

Again, it all depends. If you are distributing processes randomly, then this is true. However, if you are intelligent in the way you distribute processes, your system reliability will beat that of a UNIX system by an order of magnitude or more. So hopefully, your answer to this is false.

- *A fault-tolerant processor board is less reliable than a UNIX processor board.*

The answer to this one is probably true, but so what? In order to build a truly fault-tolerant system, all components, both hardware and software, must be fail-fast so that corrupted data does not get

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

propagated. To achieve this, fault-tolerant processor boards often use a pair of on-board, lock-step processors which continually compare results. If there is a mismatch, the processor board shuts down immediately. Thus, the processor board can fail if either on-board processor fails, giving two failure modes instead of one for the UNIX processor. Given comparable component and manufacturing quality and comparable component count per processor, we expect the fault-tolerant processor board to fail twice as often as a UNIX processor. But this is a trivial price to pay for the ultimate fault tolerance provided. A fault-tolerant architecture beats non-fault-tolerant architecture by two 9s or so (100 times more reliable).

Summary

We summarize the concepts presented above by considering how we might improve reliability. Our general availability relationships of Equations (1-1), (1-7), (1-9), and (1-10) state that¹⁰

$$A = \frac{MTBF}{MTBF + MTR} \approx 1 - \frac{MTR}{MTBF} \quad (1-1a,b)$$

$$A \approx 1 - f(1-a)^{s+1} \approx 1 - f \left(\frac{mtr}{mtbf} \right)^{s+1} \quad (1-7), (1-9)$$

$$MTR = \frac{mtr}{s+1} \quad (1-10)$$

From the above equations, we can also determine that

$$MTBF \approx \frac{mtbf}{f(s+1)} \left(\frac{mtbf}{mtr} \right)^s \quad (1-11)$$

¹⁰ These equations are restatements of the Einhorn relationships. See Einhorn, S. J.; "Reliability Prediction for Repairable Redundant Systems," Proceedings of the IEEE, February, 1963.

These expressions relate system availability, A , system mean time to repair, MTR, and system mean time before failure, MTBF, to four parameters around which we can get our hands – f , s , $mtbf$, and mtr :

mtbf Subsystem mean time before failure is out of our control. Not much we can do about that.

mtr Reductions in subsystem repair time have an exponential impact on availability. In a one-spare system, cutting subsystem repair time in half provides a four-fold improvement in reliability. Cutting it by a factor of ten provides 100 times more reliability – two 9s on the availability scale. Consider a tighter service level agreement (SLA) or, for larger users, on-site spares and on-site maintenance.

s There's not much that we as users can do to increase sparing of manufacturer-supplied critical processes. The fault-tolerant system vendor must decide that the significant software development effort to do this is worthwhile. Until this step is taken, there is not much sense in critical application processes being written with sparing in excess of one.

f Ah! We can control the number of failure modes. As we have shown, the intelligent distribution of critical processes can reduce failure rates significantly. If we work at it, we actually can pick up one or two 9s.

How Far Should We Go?

From Figure 1-6, we see that we can achieve an availability of four 9s with our fault-tolerant systems if we can keep the failure modes to five or less. Isn't this enough?

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

The Standish Group¹¹ has defined the following application categories and their required availability:

<u>Class</u>	<u>9s</u>
Non-critical	2
Task critical	3
Business critical	4
Mission critical	5
Safety critical	6

**Availability Requirements
(The Standish Group)
Table 1-3**

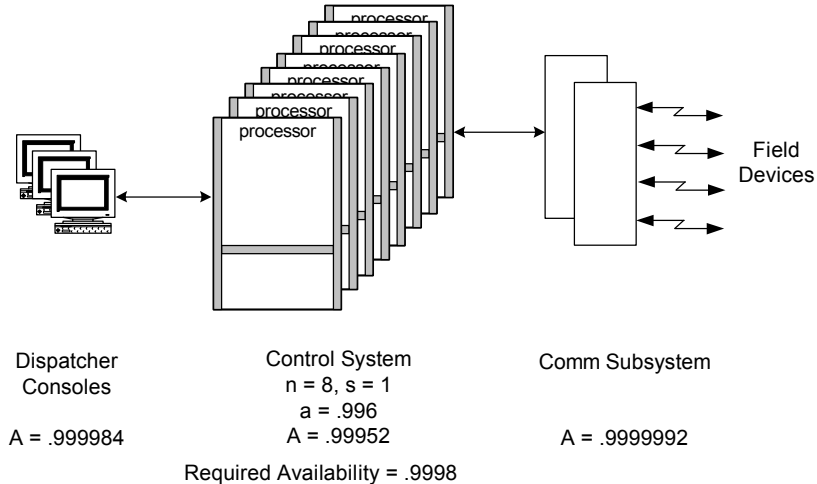
So if you have a need that you might characterize as business critical, mission-critical, or safety-critical, you had better mind your 9s.

A Case Study

Amtrak provided a real-life case study of the above concepts with their real-time train control system for the busy Northeast Corridor. Shown at a very high level in Figure 1-7, Amtrak's system uses a NonStop system to monitor and control trains via a duplexed communication link to track-side devices (signals, switches, occupancy detectors). A redundant console system is used by the train dispatchers to monitor and direct train traffic.

A detailed analysis of the system availability (whose results are shown in Figure 1-7) predicted a system availability of .9995. Not bad! Unfortunately, the specifications for the system required an availability of .9998. The design had missed the reliability mark by a factor of 2.5. To make matters worse, this was a fixed price contract; and the system was not going to be accepted unless the availability requirement was met.

¹¹ Standish Group; 2002.



System Availability = $.999984 \times .99952 \times .9999992 = .99950$
 Failure Rate is $.0005 / .0002 = 2.5$ times worse than required.
 Solution: Reduce failure modes from 28 to 12 through software configuration.

Amtrak Train Control System Figure 1-7

It was clear that the culprit was the eight-processor K-series system that had an availability of .99952 based on the worst case of 28 failure modes. However, a little algebra showed that if the number of failure modes were reduced to 12, the specification would be met. Process allocation guidelines were put in place, and the system was accepted.

This was an inexpensive fix to a potentially very expensive problem and used the concepts discussed in this chapter.

What's Next?

Bear in mind that all we have really talked about so far are general availability concepts as applied to redundant hardware systems. The consideration of faults caused by software and human errors is a much

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein

more complex subject which is considered in Chapter 5, “*The Facts of Life.*”

In Chapter 2, “*Splitting Systems,*” we explore in more depth the availability considerations and advantages of replicating and splitting systems.