

Breaking the Availability Barrier II

*Achieving Century Uptimes with
Active/Active Systems*

Breaking the Availability Barrier II

*Achieving Century Uptimes with
Active/Active Systems*

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

AuthorHouse™
1663 Liberty Drive, Suite 200
Bloomington, IN 47403
www.authorhouse.com
Phone: 1-800-839-8640

© 2007 Dr. Bruce Holenstein, Dr. Bill Highleyman, and Paul J. Holenstein. All rights reserved.

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

First published by AuthorHouse 5/24/2007

ISBN: 978-1-4343-1603-5 (sc)
ISBN: 978-1-4343-1604-2 (hc)
ISBN: 978-14343-1605-9 (e)

Printed in the United States of America
Bloomington, Indiana

This book is printed on acid-free paper.

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,
Denise, Janice, and Karen,
for their enduring patience and support.**

**We also dedicate this book to Jim Gray
for his fundamental contributions to transaction
processing technology on which this book is based.
Jim, an avid sailor, has been missing at sea
since January 28, 2007.**

Breaking the Availability Barrier II

Contents

Forward	i
What is “This Book”?	ii
Achieving Extreme Availabilities	iii
A Roadmap Through “This Book”	vii
Part 1 – Survivable Systems for Enterprise Computing	viii
Part 2 – Building and Managing Active/Active Systems	ix
Part 3 – Infrastructure Case Study	x
Part 4 – Active/Active Systems at Work	xii
Appendices	xiii
Authors’ Notes	xiv
Acknowledgements.....	xiv
About the Authors	xvi
<i>Part 1 – Survivable Systems for Enterprise Computing.....</i>	1
Chapter 1 - Achieving Century Uptimes	3
What is Reliability?	4
MTBF and MTR.....	5
Recovery Time Objective (RTO)	6
Availability	7
Recovery Point Objective (RPO).....	8
System Availability.....	9
The Availability Relationship.....	9
The Reliability Relationship.....	11
The 9s Measure of Availability	12
The Price of Reliability	13
Reliability and Cost	13
<i>Hardware</i>	13
<i>Software</i>	14
<i>People and Sites</i>	14
<i>Environment</i>	14

Breaking the Availability Barrier II

<i>Networking</i>	15
Reliability and Performance	15
<i>Software</i>	15
<i>Transaction Management</i>	16
<i>Redundant Disks</i>	16
<i>Node Synchronization</i>	17
The Why of Century Uptimes	18
The How of Century Uptimes	19
Doubling Your 9s.....	19
The n+1 Solution	21
Dual Redundancy for Double 9s.....	22
Stateless Distributed Systems	23
Stateful Distributed Systems.....	25
Active/Active Systems.....	26
<i>What is an Active/Active System?</i>	26
<i>Dually Redundant Uniprocessors</i>	30
<i>Dually Redundant Multiprocessors</i>	32
The Acceptance of Active/Active Technology	34
What's Next.....	35

Chapter 2 - Reliability of Distributed Computing Systems 37

Active/Active Systems Reviewed	41
The Availability Relationship	43
Availability of Computing Networks	44
The Importance of Repair Time.....	49
Sequential Repair	51
Parallel Repair.....	52
The Impact of Parallel Repair	53
The Importance of Recovery Time and the 4 Rs.....	56
The Impact of Restore Time on MTR.....	56
Impact of Hardware, Software, and Operator Faults	60
What is r' and What is R?	63
Nodes and Systems	64
When Hardware Reliability No Longer Matters.....	66
System Splitting	69

Breaking the Availability Barrier II

Multiprocessor Systems.....	70
Uniprocessor Systems.....	76
Failover Time	77
What is Failover?.....	77
The Impact of Failover Time on Availability.....	77
<i>s+1 Subsystems Fail</i>	78
<i>Failover</i>	78
<i>System Probability of Failure with Failover</i>	79
Some Examples	80
<i>A Cluster</i>	80
<i>Active/Backup</i>	81
Active/Active Systems.....	82
Failover Faults	83
Environmental Faults	88
What's Next?	92
Chapter 3 – An Active/Active Primer	93
A General Solution.....	94
Database Locality	96
Database Synchronization	98
The Data Replication Engine.....	99
Ping-Ponging	102
Data Loss	103
Collisions	103
Replication Latency	105
<i>Queues</i>	105
<i>Polling</i>	106
<i>Communication Buffering</i>	106
Replication Capacity.....	107
Referential Integrity.....	110
<i>Database Consistency</i>	110
<i>Natural Flow</i>	111
<i>Serialization</i>	113
Synchronous Replication	117
Network Transactions.....	118
Coordinated Commits.....	121

Breaking the Availability Barrier II

Replicated Lock Management	123
Comparison	126
Deadlocks.....	127
Failure Mechanisms	128
Node Failure.....	129
Database Failure.....	129
Network Failure	130
Controlling Database Costs.....	131
Full Mirrored Locality	131
Full Locality.....	133
Split Mirrors.....	134
Network Attached Storage.....	134
Redundant Network Attached Storage.....	135
The Availability/Performance/Cost Compromise	135
What's Next?.....	136

Part 2 – Building and Managing Active/Active Systems 139

Chapter 4 - Active/Active Topologies.....141

Architectural Topologies.....	141
Database Partitioning.....	142
<i>User Locality.....</i>	<i>143</i>
<i>Remote Transactions</i>	<i>146</i>
<i>Database Modification</i>	<i>149</i>
<i>Data Content Routing</i>	<i>152</i>
<i>Application Routing.....</i>	<i>154</i>
Software Replication.....	156
<i>Asynchronous Replication</i>	<i>157</i>
<i>Single Entity Instance</i>	<i>160</i>
<i>Multiple Entity Instances</i>	<i>161</i>
<i>Data Collision Detection and Resolution</i>	<i>161</i>
<i>Relative Replication.....</i>	<i>165</i>
<i>Row Content.....</i>	<i>166</i>
<i>Node Precedence</i>	<i>166</i>

Breaking the Availability Barrier II

<i>Combination</i>	167
<i>Business Algorithm</i>	167
<i>Designated Master</i>	167
<i>Master/Slave</i>	168
<i>Self Correcting</i>	169
<i>Manual Correction</i>	169
<i>Data Collision Logging</i>	169
<i>Synchronous Replication</i>	170
Transaction Replication	174
Hardware Replication	176
Hardware Replication	176
The Fairness Doctrine	182
Network Topologies	187
Unidirectional Disaster Recovery	187
Sizzling Hot Takeover	191
Bidirectional Active/Active	193
Route Thru	194
Ring	195
Hierarchical	201
Fully Connected	203
Partially Connected	204
Internet/Intranet	205
What's Next	207
Chapter 5 - Redundant Reliable Networks	209
The Need for Network Redundancy	209
Reliability Is More Than Just Redundancy	210
Reconfiguration Time	210
Capacity	211
Latency	212
Security	212
The Great Protocol Wars of the Twentieth Century	213
Redundancy Configurations	213
Backup Networks	213
Local Networks	214

Breaking the Availability Barrier II

<i>Dual LANs</i>	214
<i>Virtual IP Address</i>	215
Long-Haul Networks	217
Reconfigurable Networks	218
Local IP Networks	219
<i>Alternate Routes</i>	219
<i>Dynamic Routing</i>	220
<i>Fragmentation</i>	221
Long-Haul IP Networks	222
<i>IP Networks</i>	222
Telephone Networks	224
<i>Dialed Backup</i>	224
Backup Connections	225
The Internet	225
Fault Detection	226
Heartbeats	226
Missing Responses	228
TCP Detection	229
Path Monitoring	229
Fault Recovery	230
Alternate Routing	230
Backup Networks	230
<i>Redundant Transmission</i>	230
<i>Load Sharing</i>	231
<i>Primary/Backup</i>	231
Virtual IP	232
Node Failure	234
Automatic or Manual Failover	235
Fault Repair	235
Transaction, Session, and Connection Loss	236
Cost	237
A Case Study	238
What's Next	240
Chapter 6 - Distributed Databases	243
The Need for Distributed Databases	243

Breaking the Availability Barrier II

Database Synchronization	244
Asynchronous Replication.....	245
Synchronous Replication.....	246
Transaction Replication.....	247
Issues with Distributing a Database.....	247
Replication Latency	248
Application Latency.....	249
Data Loss Following a Failure.....	250
Deadlocks	250
Collisions	251
Database of Record.....	252
<i>Synchronous Replication</i>	253
<i>Partitioned Database</i>	253
<i>Master/Slave Topology</i>	254
<i>Global Mutex</i>	254
<i>General Asynchronous Replication</i>	254
<i>Verification and Validation</i>	255
<i>Succession</i>	255
Issues with Remote Access	256
Why Remote Access?	256
Network Latency	258
Data Locality	259
Partitioning	259
Network Redundancy	262
Security	262
Fairness	263
Failure Modes	263
Failover	264
Redundant Configurations.....	264
Database Failure	265
<i>IP Link</i>	265
<i>NonStop Expand</i>	267
<i>Switching Users</i>	269
Node Failure	269
Network Failure.....	269
Shared Links	270

Breaking the Availability Barrier II

Database Recovery.....	270
Shared Links	271
Replication Queue.....	271
Reload	272
What's Next.....	272
Chapter 7 - Node Failures.....	273
Causes of Node Failures.....	273
Detecting Failures	274
Local Network	274
Replication Link.....	275
Database.....	275
Heartbeat Network	275
<i>Heartbeats</i>	276
<i>Heartbeat Period</i>	276
<i>Network Redundancy</i>	277
<i>Failure Detection</i>	278
Node.....	279
<i>Self-Monitoring</i>	279
<i>Arbitration</i>	282
Failover	283
Local Network	283
Replication Link.....	283
Database.....	284
Heartbeat Network	284
Node.....	284
Switching Users	285
A Network Topology	285
<i>Intranode Network Topology</i>	285
<i>Virtual IP</i>	287
<i>Network Topology</i>	287
<i>Rerouting Virtual IP</i>	288
Manual Switchover	288
Arbitration.....	288
Automatic Switchover	288
<i>Address Resolution Protocol (ARP)</i>	289

Breaking the Availability Barrier II

<i>Gratuitous ARP</i>	290
<i>Implementing the Switchover</i>	290
Node Recovery	291
Other Issues	291
Manual Versus Automatic Failover.....	292
Load Shedding.....	292
Split Brain.....	292
Tug of War.....	293
Synchronous Replication.....	293
Capacity Expansion	294
What's Next	294

Chapter 8 – Eliminating Planned Outages with Zero Downtime Migrations..... 295

Introduction	295
Yes! Application Availability Counts.....	296
The ZDM Solution	298
An Overview.....	298
The ZDM Procedure	299
<i>Step 1: Isolate the Node to be Upgraded</i>	300
<i>Step 2: Upgrade the Isolated Node</i>	300
<i>Step 3: Create a Test Database</i>	300
<i>Step 4: Test the Upgraded Node</i>	304
<i>Step 5: Synchronize the Upgraded Database</i>	305
<i>Step 6a: Live Trial or Verification</i>	305
<i>Step 6b: Problems?</i>	306
<i>Step 7: Make Upgraded Node Fully Operational</i>	307
<i>Step 8: Upgrade Old Node</i>	307
Data Collisions in a Partitioned Environment	307
Rolling Upgrades.....	308
Single-System Upgrade	309
Migrating to Active/Active	311
Making an Application Migration-Ready.....	311
<i>Making an Application Active/Active-Ready</i>	311
<i>Decomposing an Application for Migration</i>	314
Putting the Active/Active-Ready System into Service.....	317

Breaking the Availability Barrier II

Migrating to an Active/Active Architecture	318
Uses for ZDM	319
System Upgrades	319
Incremental Migration	319
Capacity Expansion	321
Load Balancing	322
The Online Copy Facility	322
Requirements	322
Fuzzy Copying.....	324
Zero Downtime Migrations with Low Risk.....	325
Planned Outages Eliminated.....	327
What's Next?.....	327

Chapter 9 – Total Cost of Ownership (TCO)329

Choosing the Solution	329
Net Present Value (NPV).....	330
Internal Rate of Return (IRR)	333
Return on Investment (ROI)	339
Total Cost of Ownership (TCO).....	340
Initial System Cost	341
Processing Redundancy	341
<i>Highly-Available Nodes</i>	343
<i>Fault-Tolerant Nodes</i>	345
<i>Comparison Summary</i>	348
Storage Redundancy	350
<i>Disk Configuration</i>	350
<i>Full Locality</i>	355
<i>Partial Locality</i>	356
<i>Network Attached Storage (NAS)</i>	357
<i>Mix and Match</i>	359
Recurring Costs	360
Maintenance.....	360
Downtime.....	360
Software	362
Communications	363
Network Management.....	364

Breaking the Availability Barrier II

Personnel.....	365
Facilities.....	367
Business Risk Insurance	368
A TCO Toolkit	370
Putting It All Together	372
What's Next	373
Appendices	375
Appendix 1 – Rules of Availability	377
Volume 1 Rules	377
Volume 2 Rules	381
Volume 3 Rules	384
References and Suggested Reading	387
Index	395
About the Authors	419

Forward

Given today's technology, [six 9s] is unachievable for all practical purposes, and an unrealistic goal.

- Evan Marcus and Hal Stern, 2000¹

My, how things change in just a few years! Not only are we going to talk about achieving systems with six 9s availability but also with eight 9s availability and beyond. Furthermore, we are not talking just about system availability. We are talking about application service availability. After all, following a failure of some sort, if the users of an application are being serviced in an unacceptable manner (such as experiencing excessively long response times), then the application is essentially *not available*.

If you could configure your current system to:

- provide extreme availability - MTBFs measured in centuries,
- affect only a subset of users upon a failure,
- recover from any failure in subseconds to seconds,
- lose little if any data as the result of a failure,
- eliminate planned downtime,
- achieve disaster tolerance,
- use all available capacity,
- load balance at will,
- be easily expandable,
- require no change to existing applications,
- all at little or no additional cost,

wouldn't you be interested? We think so, and that is what this book is all about. Active/active systems can and do provide these benefits today.

¹ Evan Marcus, Hal Stern, *Blueprints for High Availability: Designing Resilient Distributed Systems*, Wiley; 2000.

Breaking the Availability Barrier – Volume 2

Abe Lincoln said that “it is better to remain silent and be thought a fool than to speak out and remove all doubt.” At the risk of sounding foolish to some, we recognize that there are naysayers who will argue that extreme availabilities cannot be achieved. In this book we are speaking out, confident that the many examples of successful installations of active/active systems will prove us not to be fools, notwithstanding Abe.

What is “This Book”?

We referred to “this book” in the previous section. Actually, when we started to write “this book,” we intended it to be the second in a series on active/active systems. However, when we finished it, it became apparent that it was much too long to be a comfortable single book to read. Therefore, we decided to break it up into two volumes.

We will refer to the (now) three volumes as Volumes 1, 2, and 3. “This book” comprises Volumes 2 and 3. The titles of the active/active trilogy are:

Volume 1: *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, published by AuthorHouse in 2004,

Volume 2: *Breaking the Availability Barrier II: Achieving Century Uptimes with Active/Active Systems*, published by AuthorHouse in 2007 along with Volume 3.

Volume 3: *Breaking the Availability Barrier III: Active/Active Systems in Practice*, published by AuthorHouse in 2007 along with Volume 2.

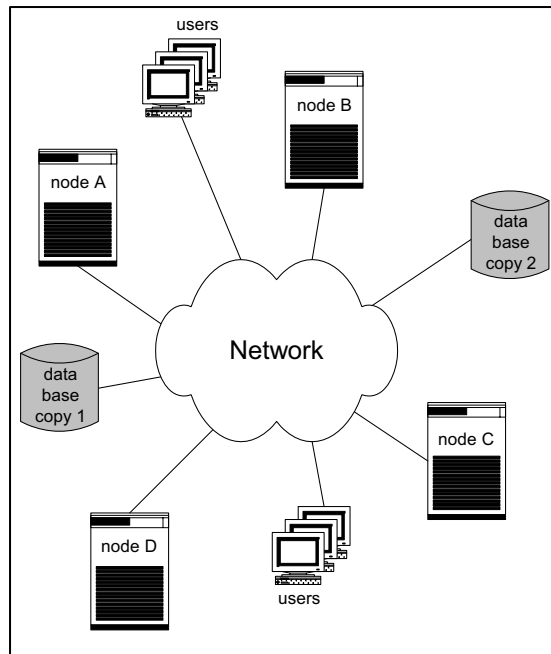
In keeping with Volumes 2 and 3 being essentially one book, this Forward is the same in each volume. However, the content of each volume is markedly different.

Let us now return to the introduction of active/active systems.

Achieving Extreme Availabilities

The secret to the achievement of extreme availabilities is in the configuration. By configuring (or re-configuring) your monolithic system as an active/active architecture, the benefits described in our introduction can all be achieved.

What is an active/active system? We define it as *a network of independent processing nodes, each having access to a common replicated database. All nodes can cooperate in a common application, and users can be serviced by multiple nodes.*



An Active/Active System

Note an important implication of this definition. Active/active architectures are not just about protecting against hardware failures. In most cases, any event that will bring down a monolithic system will only bring down one node in an active/active system. Such

Breaking the Availability Barrier – Volume 2

failure events include not only hardware faults, but also software faults, operator errors, environmental failures (air conditioning, power, etc.), and manmade or natural disasters. Active/active architectures protect users against all of these faults, allowing service to be continued by simply switching users from a failed node to one or more surviving nodes.

Another implication is what active/active is not. Active/active is not a technology; it is a business solution. Active/active is not about distributed database synchronization; it is about achieving century uptimes. More specifically,

- Active/active systems are not co-located clusters. A basic tenet of active/active systems is that they protect against area-wide problems. If the nodes cannot be geographically separated, then they are not part of an active/active system.
- Active/active systems are not independent nodes using a common database. In such an architecture, the database cannot be geographically distributed and represents a single point of failure.
- Active/active systems are not those that use hardware replication for database synchronization. Hardware replication cannot guarantee referential integrity.² As a consequence, applications at synchronized sites cannot use the database copies.
- By the same token, active/active systems are not those that use software replication engines that do not guarantee referential integrity.
- Active/active systems are not clusters. Users on an active/active system can be put back into service in seconds by

² See Chapter 4, Volume 2, [Active/Active and Related Technologies](#).

Forward

switching them to another operating node. Clusters require that another node be brought online, a process that typically takes minutes. This time delay precludes century uptimes.

- Active/active systems are not lock-stepped or voting systems because such designs require each node to process the same requests, thus precluding scalability.
- Active/active systems are not limited to enterprise applications. There are active/active distributed database systems on the market that are loosely coupled and synchronized by replication.
- Active/active systems do not require distributed disk-resident databases. Many active/active systems maintain their databases in memory.

Of course, in some cases, there may be no need for a database in an application (for example, a cluster of Web servers). In such systems, there is no context saved between operations. Implementing clusters of systems such as these is not a difficult task as it is only necessary to route any transaction to any surviving server. However, if an active database is involved such that context is retained from transaction to transaction, then providing a redundant synchronized database is necessary. This brings with it a myriad of issues. These volumes concentrate on applications which depend upon an integrated and updatable distributed database.

In many cases, the nodes in the application network are completely symmetric. Any transaction can be routed to any node, which can read or update any set of data items in the database. Should a node fail, users at the other nodes are unaffected. Furthermore, the users at the failed node can be switched quickly to surviving nodes, with their services restored in seconds or less.

Breaking the Availability Barrier – Volume 2

In seconds is the secret. Common today is the use of cluster technology to provide high availability. Should a node in the cluster fail, users are switched to a backup node. However, the applications on that node must be brought up and database tables and files opened before application services can be offered to the users. This process typically takes several minutes or more. In active/active configurations, all applications are already up and running on each node and are actively processing transactions. All that must be done is to switch over affected users to surviving nodes.

Let us say that an active/active system can recover services in three seconds and that the equivalent cluster can recover in five minutes (300 seconds). The cluster will be down one-hundred times longer than will the active/active system. This lops off two nines from the cluster's availability relative to the equivalent active/active system. A six 9s active/active system would be reduced to an availability of four 9s if it were in a cluster configuration. No wonder in 2007 many pundits still state that six 9s is not possible. But it is, as we will show in these volumes.

This leads to one of our availability rules:

Rule 36: *To achieve extreme reliabilities, let it fail; but fix it fast.*³

Are extreme availabilities important to you? Are the four 9s available with HP NonStop servers or with PC or Unix clusters acceptable? As we will discuss later, surveys have shown that the costs of downtime can range from USD \$100,000 to several million dollars an hour, depending upon the application. Perhaps even worse, downtime can lead to the dreaded “CNN Moment” and massive losses in stock value (see Chapter 9, Total Cost of Ownership (TCO), in Volume 2 for what happened to AOL in 1996 and eBay in 1999). At the extreme, downtime can lead to significant property loss or even loss of life.

³ Rules 1 through 35 are formulated in Volume 1 of this series. The complete set of rules are summarized in Appendix 1 of both Volumes 2 and 3.

Only you can make this judgment. If extreme availabilities are important to your enterprise, “this book” is for you.

A Roadmap Through “This Book”

As we explained earlier, “this book” is in fact Volumes 2 and 3 of our trilogy describing how to achieve extreme availabilities with active/active systems. The first volume in this series, published in 2004 by AuthorHouse (www.authorhouse.com) and entitled *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, referred to herein as Volume 1, lays the groundwork and the theory supporting the concepts of active/active systems. These two current volumes focus more on the practical aspects of implementing these systems.

They are broken into four parts, Parts 1 and 2 being in Volume 2 and Parts 3 and 4 being in Volume 3:

- Part 1, Survivable Systems for Enterprise Computing, summarizes and expands on Volume 1 and provides the background for the further topics discussed in these Volumes 2 and 3. Volume 1 is not needed to understand the content or the conclusions of Volumes 2 and 3.
- Part 2, Building and Managing Active/Active Systems, demonstrates how to build the redundancy required by active/active systems and how to control their cost and performance.
- Part 3, Infrastructure Case Study, describes an example of commercially available infrastructure products known to the authors to be suitable for production active/active systems. It also provides a valuable performance analysis tool for these products.

Breaking the Availability Barrier – Volume 2

- Part 4, Active/Active Systems at Work, summarizes many of the beneficial uses of active/active systems, provides several case studies of active/active systems in use today, and describes various related technologies and issues.

The authors' intended audience for these Volumes 2 and 3 and their predecessor Volume 1 includes IT executives who feel that they must reduce the downtime of their systems, system architects and senior developers who must build these systems or modify existing systems to achieve the required availability, and operations staff who must run these systems and recover from system faults.

Part 1 – Survivable Systems for Enterprise Computing

As the French biologist Louis Pasteur said, “Chance favors the prepared mind.” To prepare ourselves to understand active/active systems, Volume 1 of this series laid the groundwork for active/active systems and supported the concepts with mathematical analyses. As said earlier, Part 1 of this Volume 2 summarizes and expands upon the contents of Volume 1.

In Chapter 1, Achieving Century Uptimes, we talk about what is reliability and how to quantify it. We then extend these concepts to extremely reliable system configurations called *active/active* systems.

Chapter 2, Reliability of Distributed Computing Systems, summarizes the mathematical foundations for active/active systems. For the reader who is mathematically adverse, you will be pleased to know that the rest of this book uses minimal mathematics (except for the data replication engine performance model, which is relegated to Appendix 2). In fact, Chapter 2 can be skipped without missing the main points of the material in the following chapters.

An overview of active/active systems is discussed in Chapter 3, An Active/Active Primer. Here we discuss in some detail the structure and characteristics of the all-important data replication engine. We

Forward

also look briefly at the various failure modes and how to recover from them as well as how to control costs of active/active architectures. These later subjects are analyzed in much greater detail in Part 2 of this volume.

Part 2 – Building and Managing Active/Active Systems

The whole rationale behind active/active systems is active redundancy, which masks failures by recovering from them so rapidly that no one notices. A similar but localized philosophy is used in HP's NonStop servers, in which critical software processes are supported by backup processes in other processors resident in the same node and ready to take over in subsecond time. Also, all databases are redundant so that disk faults are masked.

There are a variety of application network topologies that have the characteristics of active/active systems. In Chapter 4, Active/Active Topologies, examples of many of these configurations are described.

In active/active systems, the inherent redundancy includes networks, databases, and processing nodes. Chapter 5, Redundant Reliable Networks, discusses ways in which to build the reliable networks needed for data replication to provide database synchronization between distributed database copies, for heartbeats to monitor the health of the processing nodes, and for users to be switched between nodes.

Chapter 6, Distributed Databases, describes how data replication engines can be used to keep in synchronism the multiple copies of a database in the application network. It discusses issues with replication such as data collisions and loss of data following a failure. Recovery from a failed database copy and access to a viable database copy following a node or network failure are explored.

The monitoring of a processing node's health is discussed in Chapter 7, Node Failures. A node can be considered to have failed if the processing system comprising that node has failed, if its database

Breaking the Availability Barrier – Volume 2

has failed, or if it has lost connectivity to the rest of the application network due to network faults. Techniques for recovering from a node failure are discussed, including issues such as tug-of-wars and operating in split-brain mode.

A highly beneficial use of controlled failures is shown in Chapter 8, Eliminating Planned Outages with Zero Downtime Migration (ZDM). Planned downtime is one of the major causes of reduced application availability. In many installations, the planned downtime required to upgrade a system or to execute other maintenance functions far exceeds unplanned downtime due to faults. In active/active systems, a node can be taken out of service purposefully with little or no impact on the users. This capability can be used to advantage to upgrade hardware, operating system software, application software, database structures, and so on. This technique also allows the capacity of the application network to be easily expanded by adding new nodes online.

Controlling the cost of an active/active system is as important as it is with any other system. However, active/active systems present an additional level of complexity. There are many ways to configure an active/active system to manage the appropriate compromise between cost, availability, and performance. As we look at different potential configurations, how do we know which contenders are the least costly? What are the factors that enter into the total cost of ownership equation? These topics are discussed in some detail in Chapter 9, Total Cost of Ownership (TCO).

Part 3 – Infrastructure Case Study

In the first two parts of “this book”, we describe why active/active systems can provide such high availability and how to build these systems. A set of tools is described that form a basis for the implementation of active/active systems. In Part 3, we look at a set of commercially available tools that fill the needs of active/active systems, and a performance model that can be used to gauge the

effectiveness of such tools. The set of tools which are described are necessarily tools with which the authors are quite familiar but are otherwise reflective of several such tools in the marketplace.⁴

The above chapters have covered two of the three legs of the active/active triangle – availability and cost. The third leg is performance. At the heart of most active/active systems is the data replication engine, and the performance of an active/active system is directly related to this engine. In Chapter 10, Performance of Active/Active Systems, we create a performance model for a generic data replication engine and show how its various performance measures are affected by a variety of replication engine architectures. The mathematics behind the performance model are left for Appendix 2, Replication Engine Performance Model, in Volume 3.

The primary facility that is required is an appropriate data replication engine. Chapter 11, Shadowbase, describes the Shadowbase data replication engine that has been used in many such implementations. Shadowbase is an example of a data replication engine with a very low replication latency (the time it takes for a change that is made to a source database to be propagated to the target database). Low replication latency is important to minimize data collisions and also to minimize data loss following a failure.

In order to take a node out of service and later return it to service, it is important to have a database copy facility that can copy the contents of an active database to a node about to be put into service (or even after it has been placed into service) while the source database is being updated. Chapter 12, SOLV, describes such a utility. Working with Shadowbase, SOLV can efficiently make a copy of an active database even while that database is being updated. In addition, future versions of SOLV will verify that two online databases are in

⁴ See Appendix 4, Implementing a Data Replication Project, in Volume 1 of this series, *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, AuthorHouse, 2004.

Breaking the Availability Barrier – Volume 2

synchronism and will resynchronize two active databases by repairing rows with differing content.⁵

In Chapter 13, ZDM with Shadowbase, we discuss the use of Shadowbase and SOLV to upgrade nodes in an active/active system without taking down the applications. With Zero Downtime Migrations, planned downtime can be completely eliminated since nodes in an application network can be upgraded without denying service to any user. Upgrades can include the hardware, operating system, applications, database, and networks, among others. In addition, ZDM can be used to add nodes dynamically into an application network to expand its capacity.

Part 4 – Active/Active Systems at Work

After learning how to build an active/active system and having seen an example of a tool set needed to do this, Part 4 looks at some actual uses of this technology in place today. It also describes some related technologies and issues.

We start in Chapter 14, Benefits of Multiple Nodes in Practice, by summarizing the various active/active system benefits that we have discussed in the book. These benefits include achieving extreme availability and very fast response time in the face of unplanned outages and even disasters, the elimination of scheduled downtime, the efficient use of all available processing capacity, the simplification of recovery testing, and application capacity expansion, both symmetric and asymmetric.

In Chapter 15, Case Studies, we look at a variety of actual uses of active/active technology. Our examples come from a wide variety of industries, including financial institutions, telecommunications, travel, web services, brokerages, plant management, and even casinos.

⁵ Check with Gravic for availability of this feature.

Forward

Finally, in Chapter 16, Related Technologies and Drivers, we explore some technologies that are related to availability. They include Grid Computing, the NonStop Server Advanced Architecture, Split Mirrors, the Real-Time Enterprise, Bulletproof Storage, and Virtual Tape. We also discuss the large number of regulatory requirements that may affect your availability decisions.

Appendices

Throughout all three volumes of this trilogy, a variety of rules applicable to highly available systems have been stated. These rules are summarized in Appendix 1, Rules of Availability. These are annotated with volume and chapter so that their context can easily be found and studied.

Appendix 1 is contained in both Volumes 2 and 3. The remaining appendices will be found in Volume 3.

Appendix 2, Replication Engine Performance Model, sets forth the detailed mathematics behind the data replication engine performance model summarized in Chapter 10, Performance of Active/Active Systems. It also structures the resulting model into a set of tables suitable for creating an Excel spreadsheet for convenient performance calculations.

Appendix 3, Regulatory Requirements, summarizes the various regulatory issues that may have a bearing on the availability and operations of processing systems. These regulations are referenced in Chapter 16, Related Topics and Drivers.

Additionally, we asked a noted consultant in the field of highly available systems, Dr. Werner Alexi, President of CS Software, Concepts, and Solutions, GmbH, to provide his comments and critique on active/active systems. His views are presented in Appendix 4, A Consultant's Critique.

Authors' Notes

You may have noted that this is a long book when both volumes are considered. As Winston Churchill said, “the length of this document defends it well against the risk of its being read.” To mitigate this, we would like to point out that most detail is summarized in snippets that can easily be scanned, often as rules. For instance, you might want to just hunt for the rules and read the supporting text. This will give you a good feeling for where we are trying to take you.

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 commodity systems to make them redundant, including HP Superdome, Windows Server clusters, Unix clusters, Linux servers, and IBM Parallel Sysplex systems.

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

All three volumes of *Breaking The Availability Barrier* have benefited from reviews by many people. We gratefully acknowledge the contributions to this volume by Mary Heck for her contributions to Appendix 3 and by Dr. Werner Alexi for his critique, published in Appendix 4. We also thank Burt Liebowitz and John Carson, whose

Forward

book *Multiple Processing Systems for Real-Time Applications* provided background for this work, and Jim Gray, whose many writings fueled the fire. They and others who have influenced this volume include:

Werner Alexi, CS Software
Wendy Bartlett, HP
Victor Berutti, Gravic
Richard Buckle, Insession
Robert Cline, SunGard Securities Processing
Dan Coughlin, First Data Corp.
Michael Crispyn, Fifth Third Bank
Terry Cumaranatunge, Motorola
Dick Davis, Gravic
Giampaolo Gandini, Telecom Italia Mobile
Jeff Glatstein, SunGard Securities Processing
Jim Gray, Microsoft
Jon Healy, SunGard Securities Processing
Mary Heck, Gravic
Tom Hoffmann, Motorola
Bill Holenstein, Gravic
Denise Holenstein, Gravic
Dan Hoppmann, A. G. Edwards
ITUG Connection staff
Clark Jablon, Akin Gump
Gene Jarema, Gravic
Jim Johnson, The Standish Group
Tim Keefauver, HP
Rob Klotz, First Data Corp.
Bill Knapp, Gravic
Bob Kossler, HP
Burt Liebowitz, Consultant
Bob Loftis, HP
Mike Nemerowski, SunGard Securities Processing
Carl Niehaus, HP
Kate Noer, SunGard Securities Processing
Gianfranco Pompado, Telecom Italia Mobile

Breaking the Availability Barrier – Volume 2

Tullio Privitera, Telecom Italia Mobile
Janice Reeder, The Sombers Group
Steve Saltwick, HP
Harry Scott, Carr Scott Software
Scott Sitler, HP
Gary Strickler, Gravic
Bart van Leeuwen, Rabobank
Joanne Welk, Motorola

About the Authors

Paul J. Holenstein is Executive Vice President of Gravic, Inc., the maker of the Shadowbase line of data replication products. Shadowbase is a low latency, high-performance, real-time data replication engine that provides business continuity as well as heterogeneous data integration and synchronization. Mr. Holenstein has more than twenty-five years of experience providing architectural designs, implementations, and turnkey application development solutions on a variety of Unix, Windows, and VMS platforms, with his HP NonStop experience dating back to the NonStop I days. He was previously President of Compucon Services Corporation, a turnkey software consultancy that was acquired by Gravic. 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. He has published extensively on availability topics and is a coauthor of Volume 1 of this series. 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 fields of data replication, data integration, and active/active systems. He can be reached at shadowbase@gravic.com.

Dr. Wilbur H. (Bill) Highleyman brings more than forty years experience in the design and implementation of real-time, mission-

Forward

critical computing systems for companies such as Amtrak, Time, McGraw-Hill, Chemical Bank, Chicago Transit Authority, and Dow Jones Telerate. He is Chairman of The Sombers Group, a turnkey custom software house specializing in the development of real-time, online data processing systems with particular emphasis on fault-tolerant systems and large communications-oriented systems. In addition, he is the managing editor of the *Availability Digest* (www.availabilitydigest.com), a monthly periodical discussing topics in high availability with a focus on active/active systems. He was 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. He is the author of *Performance Analysis of Transaction Processing Systems*, published by Prentice-Hall, and is a coauthor of Volume 1 of this series. He holds several patents, including those in the areas of data replication and active/active systems. He can be reached at billh@sombers.com.

Dr. Bruce D. Holenstein is President and CEO of Gravic, Inc. Gravic's 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 such as Oracle, Sybase, DB2, SQL Server, 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. He is a coauthor of this series. Dr. Holenstein earned his undergraduate degree in Electrical Engineering from Bucknell University and his doctorate in Astrophysics from the University of Pennsylvania. Dr. Holenstein has cofounded and run three successful companies and holds patents in the field of data replication. He can be reached at shadowbase@gravic.com.

Part 1 – Survivable Systems for Enterprise Computing

Active/active systems are a powerful approach for achieving extraordinary application availabilities from system and networking components that have ordinary availabilities. An active/active system is a network of independent processing nodes with access to a common replicated database, all cooperating in a common application. Its primary attribute, made possible because everything is duplicated and running, is its extremely fast (subsecond to seconds) recovery time. *Let it break, but fix it fast.*

In Volume 1 of this series, *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, we explored the theory behind active/active systems and discussed techniques for implementing them with off-the-shelf products.

In this first part, we summarize Volume 1 and set the groundwork for extending the discussion into the practical aspects of active/active systems. In Chapter 1, we explore the needs and aspects of highly reliable systems. In Chapter 2, we summarize the theory behind active/active systems (you can skip this chapter if you are adverse to mathematics – most conclusions in the book don't depend upon this material). Finally, in Chapter 3, we review the various methods for building active/active systems.

Chapter 1 - Achieving Century Uptimes

“An hypothesis is a novel suggestion that no one wants to believe. It is guilty until found effective.”

- Edward Teller

The World Trade Center disaster of Sept. 11, 2001, raised the concept of business continuity to the top of corporate consciousness. The great Northeast Blackout in North America in 2003 reaffirmed how difficult it is to plan and execute a business recovery plan and how important it is to have available effective corporate data processing facilities and up-to-date corporate data at remote facilities. The Blackout went further and changed the highly available system paradigm. No longer is having systems ten miles apart sufficient. Hundreds or thousands of miles are needed to survive some disasters. If the failure of your system carries with it significant penalties in terms of cost, lost customers, unbearable publicity, or even loss of life or property, then achieving extreme reliability is of paramount interest to you.

Reliable systems do not come for free. The price of redundancy is measured both in terms of cost and in terms of performance. In fact, we can have high reliability, low cost, and high performance. Pick any two.

This second and third volumes of our series on availability⁶ are focused on how you can make the best compromise between these three factors – reliability, cost, and performance - while achieving extreme uptimes measured in terms of centuries.

⁶ W. H. Highleyman, Paul J. Holenstein, Bruce D. Holenstein, *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, AuthorHouse; 2004; Volume 1.

What is Reliability?

Reliability is not really the reliability of a system. It must be considered the reliability of services that are provided by the system to its user community. A system may be up and running, but if it is not supporting its users in the way that they require, then for all practical purposes the system is considered *down*.

To say that a service is reliable is a qualitative statement. What might be an acceptable availability for one application may be totally unacceptable for another. We therefore need a measure of reliability⁷ so that the users of a system can specify what reliability they need. This is often written into a Service Level Agreement (SLA) as a requirement for each service to be provided. When we say “users,” we recognize that a user may be a person, an application, or another system.

With respect to reliability, there are two states of a service so far as a user is concerned – it is *up* or it is *down*. If it is *up*, the user is being provided with satisfactory service by the system. Satisfactory service means that the functions required by the user are operational and that the system is responsive.

A service is *down* from a user’s viewpoint if one or more required functions are unavailable or are performing so poorly as not to be useful. For instance, if the user requires subsecond response time, and if the response time degrades to ten seconds, the service may be deemed to be down so far as the user is concerned.

When a service goes down, it is said to have suffered a *failure*. The measures of reliability deal with the characteristics of service

⁷ According to our definitions, availability and reliability are complementary terms. Availability is a measure of uptime, and reliability is a measure of downtime. We use these terms loosely here, but become more formal later.

Chapter 1 – Achieving Century Uptimes

failures. As we shall see, the reliability requirements for different services may deal with different failure characteristics.

MTBF and MTR

There are two primary characteristics of failures:

- (1) How long is a service expected to be up before it fails? This is called its *mean time before failure*, or *MTBF*.⁸
- (2) How long does it take to restore the service? This is called its *mean time to repair* (*MTR*).⁹

In some applications, MTBF is the governing factor. A satellite, for instance, is generally not repairable. When it fails, it remains useless thereafter. In this case, only MTBF counts. MTR is not a consideration. The SLA for a satellite might be specified as an MTBF that exceeds 20 years.

In other applications, MTR is the governing factor. Short outages measured as a few seconds might be tolerable in an emergency call system such as 911 systems in the United States. A ten-second outage might be an aggravation, but a ten-minute outage could mean a life lost or a building destroyed. The SLA for a service such as this might, for example, specify that outages shall occur no more frequently than once per month (its MTBF), and that 99.9% of all outages shall be restored within ten seconds (its MTR). This means that less than 1 out

⁸ There can be a minor confusion here in that MTBF is also used to designate mean time *between* failures, or the average time from one failure to the next. For the reliabilities that we will be discussing, these two meanings are virtually the same.

⁹ We use “mean time to *repair*” in this chapter somewhat loosely, as it is the normal usage. However, we discuss the “4 Rs” in Chapter 2 – repair, recovery, restore, and return. There we redefine MTR to be the mean time to *return* the system to service. It is the sum of the repair, recovery, and restore times. MTR is more precisely the proportion of time that the system is out of service. It is the system *downtime*.

Breaking the Availability Barrier – Volume 2

of 1,000 outages are allowed to last longer than 10 seconds – an average of one per 1,000 months, or approximately eighty years.

Other examples of services that may fail but which must be restored very quickly include nuclear power plant control systems, patient monitoring systems, and electric grid control systems.

Recovery Time Objective (RTO)

A service's MTR requirement is referred to as its *RTO*, or *recovery time objective*.

In commercial computing, the concern is often the cost of downtime. Cost may be measured in dollars, in lost customers, in bad publicity, or in a number of other ways.

For instance, a stock exchange might survive occasional brief outages for a few seconds and thereby suffer lost revenue from the loss of trades. Anything more than this could make the headlines.

Should an online store go down, sales are lost as customers find alternatives. Given enough bad experiences, the store could begin to lose its customer base.

Studies by the HP Online User Group Advocacy program¹⁰ showed that, for over half of the respondents, downtime cost was in excess of \$1,000 USD per hour. For 20% of the respondents, the cost of downtime ranged from \$100,000 USD per hour to \$14 million USD per day to the incalculable.

In a separate study, the July 3rd, 2006, issue of USA Today noted that 40% of 300 companies surveyed reported that their downtime

¹⁰ Survey and instapoll results posted on www.hpuseradvocacy.org. on October 2, 2003, and December 18, 2003, respectively.

Chapter 1 – Achieving Century Uptimes

cost exceeded \$250,000 per hour, and 14% said that their downtime cost exceeded \$1,000,000 per hour (page 1B).

Availability

Thus, for commercial applications, a third reliability measure is appropriate, and this is *availability*. Availability is the percentage of time that a service is up. For instance, an availability of 99.9% means that the service will be up 99.9% of the time. It will be down 0.1% of the time, or an average of about 8.8 hours per year (there are almost 8,800 hours in one year).

This does not mean that the service will have one yearly failure that lasts for 8.8 hours. It may fail ten times a year and be down for an average of .88 hours during each failure, or it may fail once every five years and be down for an average time of 44 hours.

Table 1-1 shows the translation of an availability requirement into hours per year or minutes per month.

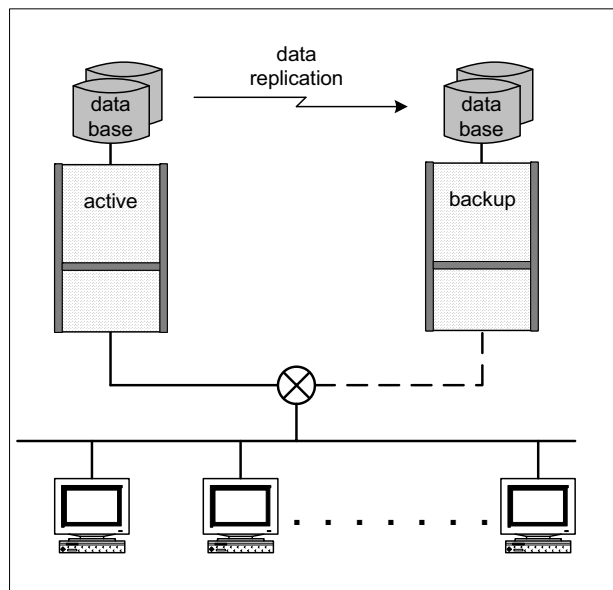
Nines	% Available	Downtime per Year	Downtime per Month
2	99%	87.6 hours	7.3 hours
3	99.9%	8.76 hours	44 minutes
4	99.99%	53 minutes	4.4 minutes
5	99.999%	5.3 minutes	27 seconds
6	99.9999%	32 seconds	2.7 seconds

**Average 24x7 Downtime
Table 1-1**

Recovery Point Objective (RPO)

There is one additional factor that must be considered, and that is the application’s tolerance to data loss that might occur as the result of a failure somewhere in the application network. The amount of data that can be lost as a result of a failure is referred to as the application’s *RPO*, or *recovery point objective*.

Systems requiring extreme levels of availability – MTBFs measured in centuries – must be redundant. This includes having duplicate database copies, which must often be distributed geographically to provide a level of disaster tolerance as shown in Figure 1-1. These databases must be kept in synchronism. When a change is made to one database, that change must be propagated to the other databases in the network.



**A Redundant System
Figure 1-1**

It is probable that when a failure occurs, the failed system is in the midst of sending updates to other copies of the database in order to keep them synchronized. This data may be lost and may or may not be recoverable. In some applications, data lost in this manner is unacceptable.

For example, a stock exchange must never lose a transaction (stock prices would be affected for the rest of the trading day). The RPO for its trading applications must be zero.

In other applications, it may be possible to reconstruct or to ignore lost data. In these applications, seconds or even minutes of data loss may be tolerable; and the RPO may be relaxed. For instance, an ATM often keeps a local paper copy of its transactions; and a temperature monitoring system may not need to keep historical values.

System Availability

The Availability Relationship

It is clear that the availability of a service is dependent both upon its MTBF (uptime) and its MTR (downtime).¹¹ A service is, of course, either up or down. An up/down cycle lasts for a time of (MTBF+MTR). Therefore, its availability A is

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

¹¹ If you are math averse, skip the following equations; and just read the conclusions at the end of this section.

Breaking the Availability Barrier – Volume 2

If MTR is very much smaller than MTBF, as is the case for the services which we are considering, then this can be written as¹²

$$A = \frac{1}{1 + \frac{\text{MTR}}{\text{MTBF}}} \approx 1 - \frac{\text{MTR}}{\text{MTBF}} \quad (1-2)$$

where “ \approx ” means “approximately equal to.”

The probability of a system being up (A) is one minus the probability that it is down (F):

$$A = 1 - F \quad (1-3)$$

Thus, from Equation (1-2),

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

Finally, from Equation (1-4), we can write

$$\text{MTBF} \approx \frac{\text{MTR}}{1 - A} \quad (1-5)$$

For instance, a service with an availability, A , of .9999 and a repair time, MTR, of four hours will have an MTBF of 40,000 hours,

¹² See Volume 1, Chapter 1, The Nines Game. As a side note, if MTBF were to be defined as mean time *between* failures, then

$$A = \frac{\text{MTBF} - \text{MTR}}{\text{MTBF}} = 1 - \frac{\text{MTR}}{\text{MTBF}}$$

Chapter 1 – Achieving Century Uptimes

or about five years. We will use this relationship to compare reliabilities in our discussions to follow.

There is one additional interpretation of availability worth noting. So far, we have characterized availability as a measure of service uptime (or service downtime). For instance, an availability of .99 means that the system is expected to be down 1% of the time, or about 88 hours per year.

Availability can also be interpreted as the probability that a user will find the system to be up when service is requested. For instance, if an ATM machine has an availability of .99, then a customer can expect that 99 times out of a hundred, the ATM will be found to be working. One out of a hundred times, he will find the ATM to be out of service.

The Reliability Relationship

For purposes of our ongoing discussions, we can now define reliability more specifically. We define reliability as the probability F that the system will be down:

$$\text{reliability} = F = 1 - A$$

Thus, a system with an availability of .999 will be down 0.1% of the time and is ten times more *reliable* than a system with an availability of .99, which will be down 1% of the time.¹³

¹³ Many contemporary definitions of reliability are functions of MTBF only. MTR is not considered. Our definition of reliability is an inverse function of MTBF for a fixed MTR. A system with an MTBF of 1,000 hours will be 10 times more reliable than a system with an MTBF of 100 hours for a given MTR.

The 9s Measure of Availability

As we have seen, availability is the probability that the system will be operational and will be providing acceptable service to its users. An availability of .999 means that it will be available 99.9% of the time.

However, for extreme availabilities, it can become quite awkward to talk about an availability of .99999999. We therefore characterize availability as the number of nines. For instance,

- 0.999 is three 9s.
- 0.99999999 is seven 9s.
- 0.998 is a little less than three 9s.
- 0.9992 is a little more than three 9s.

The number of 9s relates directly to the expected downtime for a given period of time, as was set forth in Table 1-1.

Several studies have been made of the availability of various systems in actual practice. These include all sources of failure – hardware, software, people, and environment (air conditioning, power, and so forth). A study by the Gartner Group¹⁴ revealed the following availabilities:

HP NonStop	.9999
Mainframe	.999
OpenVMS	.998
AS400	.998
HP-UX	.996
Tru64	.996

¹⁴ Gartner Group; 2002.

Chapter 1 – Achieving Century Uptimes

Solaris	.995
NT Cluster	.992-.995

Typical System Availabilities
Table 1-2

Thus, according to this study, mainframes are four to five times more reliable than Unix systems; and HP NonStop servers are up to ten times more reliable than mainframes.

The Price of Reliability

Reliability does not come for free. It requires a compromise with cost and performance. As we said earlier, you can optimize reliability, cost, and performance. Pick any two.

Reliability and Cost

Hardware

Reliable, highly available systems require redundancy so that if one component fails, there is another to automatically and rapidly take its place. True, today's nonredundant systems are often characterized as "highly available." These systems typically have availabilities of three 9s. But if downtimes measured in hours per year are unacceptable, then in today's technology, redundancy is a requirement.

This means that a reliable system will have extra processors (at least one), extra disks in a mirrored or RAID configuration, dual LANs, and extra power supplies and fans. In many cases, there may even be an extra system used as a spare.

Software

Software licensing may be a factor in certain cases. If the reliable architecture depends upon distributed nodes cooperating with each other, then there may be additional license fees, depending upon the licensing model of the vendor.

People and Sites

Again, if reliability involves distributing the application over multiple sites (typically done for disaster tolerance), there is the added cost for these additional sites and the staff to operate them.

Environment

A loss of power, a loss of air conditioning, or other environmental problems can take down a site. At the least, there must be backup generators to take over in the event of a power failure and with enough fuel on hand to allow time for fuel replenishment during an extended outage. Furthermore, there must be a UPS (uninterruptible power supply) system with sufficient capacity to power the systems until the backup generators can come online.

In the extreme, the Northeast Blackout demonstrated that distributed nodes should not only be serviced by independent power suppliers but also should be located in areas serviced by different power grids.

Even mundane things like water must be considered. Not only should water be available for staff and perhaps air conditioning, but the lack of water will make toilets inoperable.

Networking

If the system depends upon a network to connect users or to coordinate nodes in a distributed application, then the network should also be redundant. As illustrated by the 9/11 terror attacks, these networks should not only follow different paths but ideally should be provided by different carriers to protect against a carrier outage.

Reliability and Performance

Reliability can also take its toll on the response time and capacity of a system.

Software

Like hardware, software must be redundant because it, too, can fail. If a software module fails, then there must be another module to take its place – either a spare module or a currently active module that can assume the load of the failed module.

In some cases (like HP NonStop process pairs), each critical process¹⁵ has a backup process. The backup's context is kept in synchronization with the primary process via a mechanism known as checkpointing. Checkpointing imposes a nontrivial load affecting both application responsiveness and system capacity.¹⁶ Its virtue is that recovery from a software fault is almost instantaneous and can be transparent to the application.

Another technique is the use of *persistent processes*. Critical processes are monitored by a resilient process (perhaps a check-

¹⁵ A process is a program running in a processor.

¹⁶ W. H. Highleyman, Chapter 9, Fault Tolerance, *Performance Analysis of Transaction Processing Systems*, Prentice-Hall; 1988.

Breaking the Availability Barrier – Volume 2

pointed process pair). Should such a critical process fail, the monitor will re-instantiate it in a surviving processor.

Yet another technique is to provide a group of context-free processes that can handle a particular class of transactions, as do BEA's Tuxedo, IBM's CICS, and HP's NonStop Pathway. If one process fails, it is either restarted, or it is removed from the group with the remaining processes sharing the load. This technique imposes less overhead on the system, but failover can take seconds rather than subseconds – a tradeoff between capacity and response time.

Transaction Management

It is imperative to keep the system's database consistent if it is to be reliable. This task falls to transaction managers such as IBM's IMS and CICS, Tuxedo's distributed transaction manager, and HP's NonStop TMF.

With a transaction manager, a set of database operations representing a logical or business transaction is marked as an atomic set. The transaction manager guarantees either that all operations for the transaction are completed or that none are and ensures that any view of the database is consistent during the processing of the transaction.

A transaction manager may impose a significant performance penalty in terms of capacity, concurrency, and response time.

Redundant Disks

To achieve high reliability, disks must be redundant. This means additional disk write activity as the redundant disks are updated. Mirrored disks must be written sequentially to avoid data loss due to a

Chapter 1 – Achieving Century Uptimes

failure during the write process. This can significantly increase both capacity utilization and response times.

RAID arrays can impose less overhead because writes to the several disks are done in parallel. However, a failure during the write process can cause data contamination.

In general, with redundant disk systems, there is the opportunity to improve read performance by spreading the read activity over the several disks. Many applications are characterized by disk write-once, read-many operations and can significantly benefit from this approach.

Node Synchronization

The achievement of extreme reliability often means that the application is distributed among several independent but cooperating nodes. This requires that nodes be cognizant of activity at all other nodes. More specifically, each node must have access to a common database that holds the current state of the application. The database must be redundant, with redundant copies distributed to survive node outages. The various copies of the database across the network must be kept in synchronism so that anyone accessing the data from any copy will get the same results.

Keeping the databases of these nodes in synchronism will impose additional capacity utilization on all nodes. Depending upon how tightly the synchronization must be maintained, response times can be adversely affected as well.

Synchronization considerations are explored in some detail in Chapter 3, An Active/Active Primer.

The Why of Century Uptimes

Why the quest for uptimes measured in centuries? Even if we thought that we could achieve this, could we ever confirm it by field experience? Is it really meaningful? After all, we will all be gone; and the system will be retired long before a century is up.

The answers to these questions are “yes, we can measure century uptimes” and “yes, century uptimes are meaningful.” Let us look first at measuring these long uptimes.

Some applications now running require a hundred or more systems (online stores and Internet services such as email and search engines are but one example). If such an application requires one hundred systems, and each has a one-century uptime, then we can expect on average one system failure per year.

Furthermore, if in the future a vendor successfully markets extremely reliable systems with, say, ten-century uptimes and has an installed base of 2,000 systems, one would expect this community of systems to experience two outages per year. So it is perfectly reasonable to assume that we can measure this extreme reliability in many cases.

But is this extreme reliability useful? Let us take the case of an extremely reliable system with four 9s availability. This corresponds roughly to the best of today’s commercial offerings. It means that the probability of failure is 0.01%.

Put another way, it means that someone occasionally using the services of this system will find it unavailable on the average of one

Chapter 1 – Achieving Century Uptimes

time out of every 10,000 times that service is sought.¹⁷ If this results in a simple inconvenience, then the level of availability is probably more than sufficient. If it means the possibility of a nuclear meltdown, the loss of life, or the death of a \$100 million satellite, then more 9s are obviously needed. If it means the loss of \$100,000 per hour or bad publicity, then adding more 9s might be strictly a matter of cost. (The cost of reliability is explored further in Chapter 3, An Active/Active Primer, and in Chapter 9, Total Cost of Ownership (TCO).)

Thus, there are many cases in which extreme reliabilities measured in centuries merit strong consideration. In these cases, we must be prepared to cope with any failure, even if we don't know what it will be.

Besides, an uptime of a century means that someday in the next century the system will likely fail. That someday may be tomorrow.

The How of Century Uptimes

Doubling Your 9s

It is shown in Volume 1 that adding a second redundant system to a single system doubles its 9s.¹⁸ If, as in our earlier Figure 1-1, we provide a Unix system that has three 9s of availability with an equivalent standby system that can assume control transparently to the users in the event of a primary system failure, we now have a system with six 9s of availability. This certainly qualifies as an extremely available system, having an expected downtime of only about 30 seconds per year. Assuming a repair time (MTR) of four hours (a generally accepted field service interval), this represents an MTBF of 5 centuries.

¹⁷ Though this statement is true, the reason that we say “occasionally” is that if a service is used frequently (say by another process every second), and if the system is down for four hours, the result will hardly be an inconvenience.

¹⁸ See Volume 1, Chapter 1, The Nines Game.

Breaking the Availability Barrier – Volume 2

To understand this simple rule is straightforward. Using the above example, the probability of random failure of a system with three 9s is .001. If we provide a similar redundant system, then a system failure occurs only if both systems fail. This will occur with a probability of $.001 \times .001 = .000001$. The availability of the redundant system is therefore $1 - .000001 = .999999$, or six 9s.

The active/backup¹⁹ architecture shown in Figure 1-1 is no longer state-of-the-art technology for achieving extreme reliabilities. This approach can provide very high availabilities. However, it has several problems associated with it:

- Only half of the purchased capacity is used at any one time.
- All users are affected by a primary failure and must be switched to the standby node.
- Recovery can take minutes to hours as partial transactions are backed out, as lost transactions are recovered, as applications are brought up, and as users are switched over.
- Transactions in the process of being replicated at the time of failure are generally lost.

Wouldn't it be nice if we could have an architecture that

- provided extreme availability - MTBFs measured in centuries,
- affected only a subset of users upon a failure,
- recovered from any failure in subseconds to seconds,
- lost little if any data as the result of a failure,
- eliminated planned downtime,
- achieved disaster tolerance,
- used all available capacity,
- load-balanced at will,

¹⁹ The active/backup architecture goes by many names including active/standby, active/passive, primary/secondary, primary/backup, and so on.

Chapter 1 – Achieving Century Uptimes

- was easily expandable,
- required no change to existing applications,
- all at little or no additional cost?

The technology is here today to substantially meet all of these goals, and it is the active/active architecture which we will describe in detail in this volume.

The $n+1$ Solution

Providing a redundant backup system is the classic approach to extreme availability in older technology, but it is an expensive approach. One must purchase twice the equipment required to run the application. A more cost-effective approach is to add partial redundancy.

This is the approach taken by HP's NonStop servers. Each of these servers can contain up to sixteen processors. One configures the system with the number of processors needed (n) plus one additional one (thus the term " $n + 1$ ").

Normally, all $n+1$ processors are used to carry the application load. However, if a processor fails, the remaining n processors are sufficient to carry the load.

These systems are designed to survive any single hardware or software failure and many cases of multiple failures. If we assume that each processor (and its associated peripheral gear) is equivalent to our previously mentioned Unix box so far as availability is concerned, then each processor has an availability of three 9s. The probability that a specific pair of processors will fail is equivalent to an availability of six 9s, as described earlier.

However, a dual failure may potentially cause a system outage. We call this a *failure mode*. In a NonStop system, for instance, the

Breaking the Availability Barrier – Volume 2

failure of two specific processors may bring down the system if a critical process pair should happen to be running in that processor pair. In a fully configured NonStop system, there are sixteen processors. Thus, there are 120 ways in which two processors can fail.²⁰ Let us assume that 100 of these combinations will take down the system (that is, there are 100 potential failure modes for this system).

Thus, the failure rate is 100 times worse than one would assume based on an availability of six 9s. This, in effect, reduces our availability by two 9s, leaving a typical NonStop system with four 9s of availability.

Four 9s availability is an order of magnitude better than any other system being marketed as of this writing, and this availability is borne out in industry studies we described earlier. (Of course, things are more complex than this simple analysis because of faults caused by operator errors, environmental faults, and so forth;²¹ but our analysis serves to illustrate the point of this discussion.)

Dual Redundancy for Double 9s

Four 9s is a fairly high availability, but it does not come close to achieving century uptimes. If the system repair time is four hours, this equates to an average uptime of five years – impressive but not our goal.

What are our goals? As a straw man, let us say that our objectives are to provide application uptime of six 9s or better at little or no

²⁰ Specifically, $n(n-1)/2$ ways. One of n processors may fail, followed by one of the remaining $n-1$ processors. But this has counted each processor pair twice – for instance, processor 3 followed by processor 4 and processor 4 followed by processor 3.

²¹ See Volume 1, Chapter 5, The Facts of Life, and in this Volume, Chapter 2, Reliability of Distributed Computing Systems.

Chapter 1 – Achieving Century Uptimes

additional cost. In addition, in order to control costs, we want to be able to use all available capacity of the components that are up. Impressive ambitions, and ones which are achievable today, as we shall see.

As we have seen from the examples above, in order to achieve this level of reliability short of buying an entire standby system, we really need two levels of redundancy. The ability to survive any single failure can in today's technology give us four 9s. To do better than that means that we have to be able to survive at least two failures.

By the way, if we assume a four-hour repair time, six 9s equates to an MTBF of five centuries. If we can achieve this goal, we have achieved century uptimes.

Stateless Distributed Systems

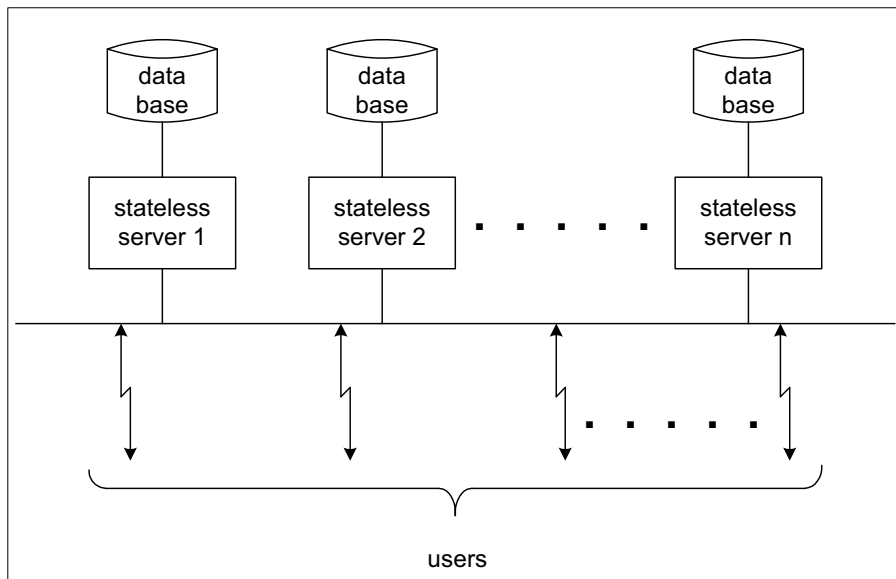
There is a special case for which the goal of six 9s or better is trivial, and that is when the application is stateless, or context-free. In a stateless system, the application responds to user requests independently of any prior activity. The results of previous requests have no bearing on the result of the current request. The system's response to a specific request will always be the same.²²

An excellent example of a stateless system is a web service that delivers fixed pages of information such as catalog pages for an online store. Each request results in a page being sent that is independent of any prior request (though the requested page may have come from a link embedded in a prior page), or the request itself carries any needed context.

²² This, of course, ignores the problem of rolling updates to the servers.

Breaking the Availability Barrier – Volume 2

Rather than having one large web page server, whose failure would bring down the store's catalog, several smaller, independent servers may be used, as shown in Figure 1-2, each capable of delivering any requested page. These servers may be collocated or may be geographically dispersed to provide disaster tolerance. Each user request is routed to an available server, which will respond to the request. A user's subsequent request will most likely be routed to another server.



A Stateless Distributed System
Figure 1-2

Should a server fail, it is simply removed from the pool of available servers. This architecture can survive multiple server failures – up to the point where the surviving servers can no longer handle the load.

Chapter 1 – Achieving Century Uptimes

The availability analysis for stateless distributed systems is similar to that for the NonStop server given earlier, except that it is trivial to provide more than one spare. Two, three, or more spares may be provided as desired with no increase in complexity.

For instance, let us assume that we need ten servers to handle our peak load; and we provide dual redundancy via two spares for a total of twelve servers. In order for the system to fail, we would have to lose three servers (and it would still be operational except for reduced capacity). If each server has an availability of three 9s, then by our previous argument the probability of losing three servers is $(.001)^3$, or nine 9s. However, as pointed out in the NonStop case, there are 220 different ways that three servers out of twelve can fail.²³ This causes a loss of two and a half 9s, resulting in an availability of full capacity in excess of six 9s.

Thus, we have achieved an availability commensurate with providing a standby system but at an expense of 20% excess capacity rather than 100% excess capacity.

Stateful Distributed Systems

Using stateless servers is all well and good for the web-page catalog operation, but an online store is not of much use unless customers can make purchases as well. (Gone are the days when customers browsed the catalog online and then called a toll-free number to place an order.) But ordering is hardly context-free. As the customer shops, the system must maintain her shopping cart, obtain billing and shipping information, validate her credit source, confirm the order to the customer, obtain her acknowledgement, and then ensure that the order is shipped.

²³ Specifically, $n(n-1)(n-2)/6$, where $n = 12$.

Breaking the Availability Barrier – Volume 2

Ordering and order fulfillment are an example of what we call a *stateful* system.

Building reliable, stateful distributed systems is our real focus in this series. *A stateful system is one in which the processing of each request is dependent upon the results of previous requests.* That is, the system must carry the context of a thread of requests and must remember all pertinent information as the transaction progresses. As a consequence, the stateful system must appear as a single system to the user with a single database of record.²⁴ The question, then, is how to build such a system with the additional sparing needed to achieve our reliability goals?

The answer is to take a lesson from stateless distributed systems and build a distributed system of independent cooperating nodes. The operative word here is *cooperating*. The nodes must each cooperate equally in the application and must maintain a common system state across all nodes. When a transaction occurs at one node, the results of that transaction must be communicated to all other nodes to update their state.

Active/Active Systems

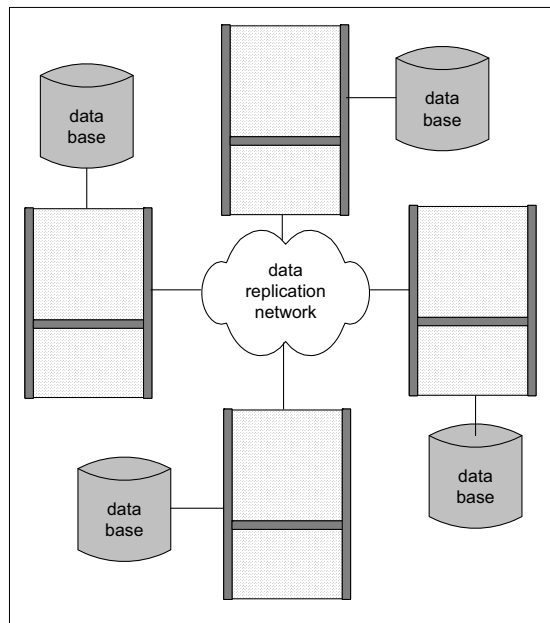
What is an Active/Active System?

Typically, the system state in commercial applications is represented by the contents of the database.²⁵ Therefore, this cooperation means that any node can change any row in the database; and that redundant copies of the application database across the network must receive these changes in order to keep all databases synchronized.

²⁴ See Chapter 6, Distributed Databases, for a further description of the database of record.

²⁵ Even though we talk primarily about disk-resident databases, databases may be memory-resident or stored on other media.

Such systems are called *active/active* systems since all nodes in the network are actively participating in the processing of user requests. A general active/active system is shown in Figure 1-3. The distributed copies of the database are kept in synchronism via data replication across their interconnecting network.



**A Stateless Distributed System
Figure 1-2**

Active/active systems do not protect only against hardware failures. They protect against anything that can take a node down including hardware faults, software faults, operator errors, environmental faults (power, air conditioning, etc.), and manmade or natural disasters.

Typically, the sum total of the capacity of the many nodes in the application network equals the capacity requirements of the

Breaking the Availability Barrier – Volume 2

application plus some additional capacity required to support the data replication overhead necessary to keep the databases in synchronism and to provide the required capacity in the event of a node failure.

Note that in an active/active system, all of the purchased capacity is usable. There is no need for an idle standby system. In addition, disaster tolerance comes for free if the nodes are geographically distributed. This leads to the following rules:

Rule 37: *All of the purchased capacity in an active/active system is usable. There is no need for an idle standby system.*²⁶

Rule 38: *Providing that the nodes in an active/active system are geographically distributed, disaster tolerance comes for free.*

There have been several precursors to active/active systems in the search for extreme availabilities, such as:

- *active/backup systems*, in which a second system is ready and able to take over processing should the primary system fail. These systems suffer from a time delay due to management involvement in the switchover decision followed by an extended time to switchover. Also, many of these systems use replication techniques which do not guarantee referential integrity, leaving the database copies unsuitable for transaction processing.
- *clusters*, which require that another node be brought online, a process that typically takes minutes.
- *multiple nodes sharing a common database*. Nodes generally are not geographically separated for disaster tolerance.

²⁶ Note that Rules 1 through 35 were formulated in Volume 1 of this series. Rule 36 is given in the Forward to this volume.

Chapter 1 – Achieving Century Uptimes

Furthermore, the common database cannot be geographically distributed.

- *lock-stepped systems*, in which two systems process the same data and compare their results on an operation-by-operation basis.
- *redundant systems*, in which data is processed independently and the results compared (this includes voting systems of three or more systems).
- *federated systems*, in which a common but non-redundant view of data distributed about the network (the *federated database*) is provided to any user no matter to which node he is connected.

None of these are active/active systems by our definition since the various systems are not independently processing transactions against a common replicated database in a loosely-coupled redundant environment. In all cases, the capacity of the entire complex is the capacity of a single system or database. Furthermore, just because bidirectional replication may be used does not make the system active/active if the replication engine does not guarantee the referential integrity of the target database, since that database cannot be used for active transaction processing.

Sparing is the number of additional redundant components that are available for taking over the functions of a failed component. As we have seen, in order to achieve extreme availabilities, we need to have at least two levels of sparing in an active/active system. If the system can withstand a single failure (as with NonStop systems), we can achieve availabilities of four 9s. If a system can survive dual failures, we can achieve availabilities in excess of six 9s.

Breaking the Availability Barrier – Volume 2

One can achieve dual sparing in an active/active system by providing two spare nodes, neither of which is fault-tolerant. Another approach is to use fault-tolerant nodes, such as HP's NonStop servers, to implement the first level of sparing and then to provide a single spare node in the network to implement the second level of sparing. Let us look at these two approaches.

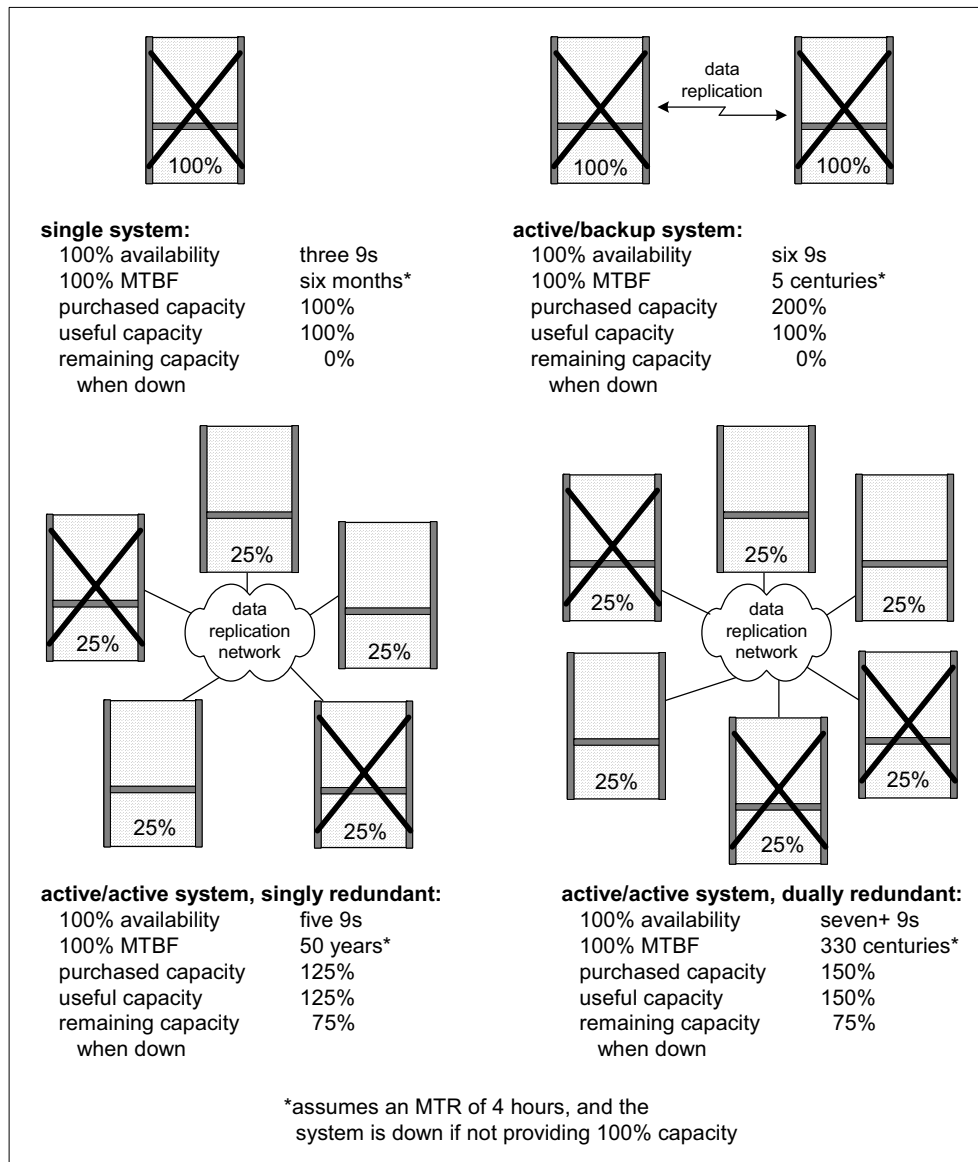
Dually Redundant Uniprocessors

Consider a Unix system that has an inherent availability of three 9s, which we would like to reconfigure for century uptimes. One way to do this, as shown in Figure 1-4, is to provide an equivalent standby system. We have doubled our cost, but we have achieved our availability goal. However, we have had to buy 200% of our required capacity; and only half of it can be used at any one time.

We can rearchitect this system by splitting it up into, say, four smaller cooperating nodes in an active/active configuration, with each node capable of carrying 25% of the load. We then add one spare system for fault tolerance. The system will continue to provide 100% capacity in the presence of any single node failure but will provide reduced capacity should two nodes fail. The result is that we have purchased only 125% of the required capacity and have achieved full-capacity availability of five 9s, or a 50-year MTBF based on a four hour MTR. Impressive, but not our goal.

In order to achieve our objective of century uptimes, we must add a second spare to become dually redundant. Now the system can survive the failure of any two nodes and has achieved a 100% availability of almost eight 9s. Thus, we have had to purchase only 150% of the required capacity and have achieved hundreds of centuries of uptime. Furthermore, all of this excess capacity is available during normal operations.

Chapter 1 – Achieving Century Uptimes



Uniprocessor Stateful Distributed Systems
Figure 1-4

Dually Redundant Multiprocessors

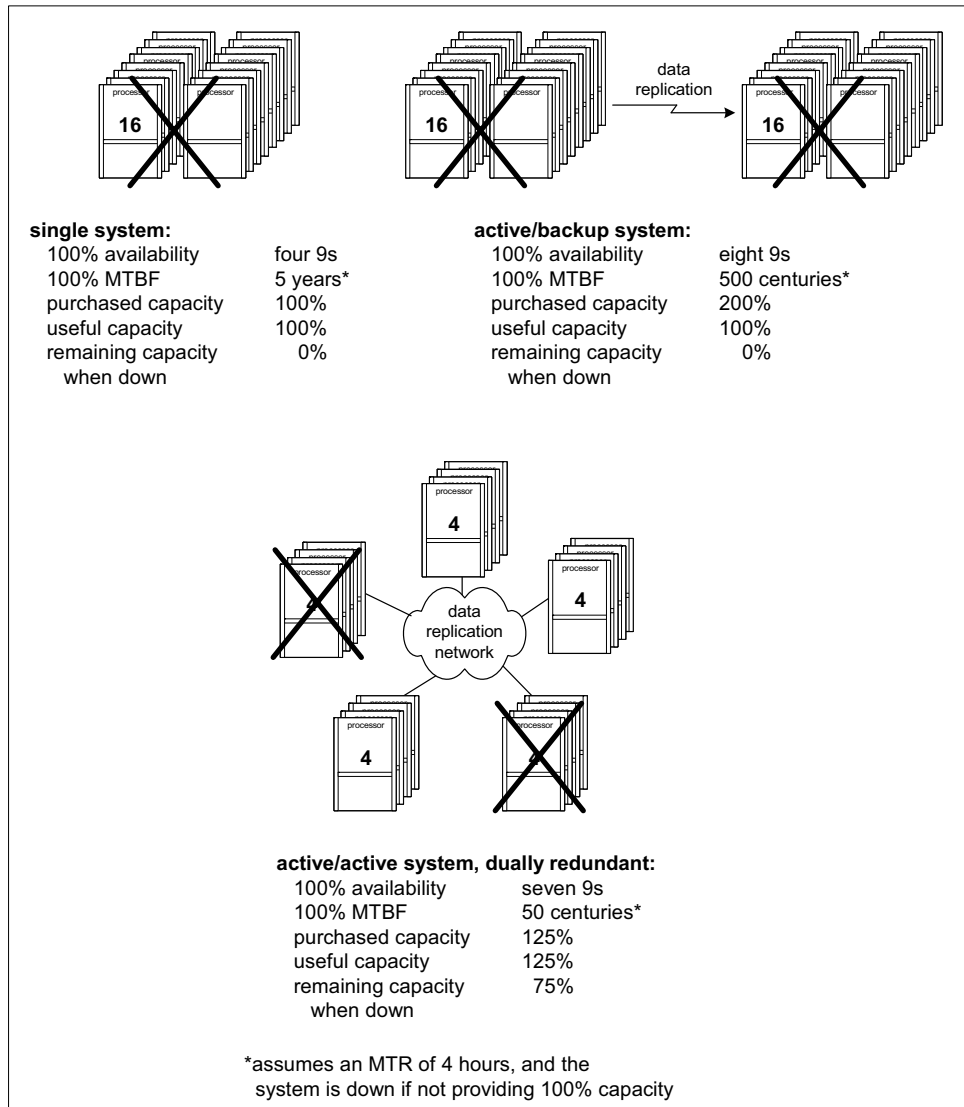
Alternatively, we can achieve the first level of redundancy by using fault-tolerant nodes such as NonStop servers. The second level of redundancy is provided by a single spare node. Let us assume that the application requires sixteen NonStop processors-worth of processing capacity. Figure 1-5 shows the characteristics of this system as a stand-alone system; in an active/backup configuration; and as an active/active system with five fault-tolerant nodes, each with four processors – the required configuration plus 25% excess capacity.

So far as the five-node active/active system is concerned, if each fault-tolerant node has an availability of four 9s, then the availability of any pair of nodes is eight 9s. However, there are ten ways in this system that we can lose two nodes ($5 \times 4 / 2$). Therefore, our system availability is seven 9s, well exceeding our goal of six 9s. Seven 9s is an average of less than four seconds of downtime per year. If the recovery from a dual failure requires four hours, we have achieved uptimes in the order of 50 centuries!

In the above example, we have achieved this reliability with 20 processors instead of the 32 processors which would have been required in an active/backup configuration. In addition, in normal operation, we have available to the application 125% of the capacity of an active/backup system and still have 75% capacity following a two-node failure.

This example also shows why in some cases the extra cost of inherently fault-tolerant systems may, in fact, be cost effective. Comparing Figures 1-4 and 1-5, we see that typically it requires one less node to achieve a particular level of availability if fault-tolerant nodes are used. This leads to the following rule:

Chapter 1 – Achieving Century Uptimes



A Multiprocessor Stateful Distributed System
Figure 1-5

Breaking the Availability Barrier – Volume 2

Rule 39: (NonStop Maxim) - *Century uptimes in an active/active system will generally require fewer nodes if fault-tolerant nodes are used rather than high availability nodes.*

The Acceptance of Active/Active Technology

The distributed active/active systems that we have described can substantially achieve the lofty goals that we set:

- They do achieve century uptimes.
- They do make available for work the purchased capacity of all of the nodes in the system.
- When a node failure occurs, only users on a failed node are affected, and they can be put back into service within seconds by switching them to a surviving node.
- As we shall see later, by using synchronous data replication, we can ensure that no data is lost due to a node or network failure.
- As an added bonus, we do not need to purchase an additional 100% of required capacity. We only need to purchase a fraction of that.

Until recently, there have been significant hurdles to implementing active/active systems. The primary hurdle has been the requirement to synchronize database copies across the network, especially when multiple systems can be updating the same data at the same time.

Other hurdles include the resynchronization of a downed database after its node has been returned to service, the introduction of expansion nodes into the system, the handling of network failures, the switching of users from a downed node to a surviving node, and the need for planned downtime for software and hardware upgrades.

Chapter 1 – Achieving Century Uptimes

As described in later chapters in these Volumes 2 and 3, solutions for all of these problems now exist; and active/active systems are being deployed into service for large-scale, mission-critical applications. These systems are being deployed as leading edge applications. As they prove their worth, active/active systems will become more predominant and will replace lesser technology to become the de facto architecture of choice.

Several case studies of existing active/active systems are presented in later chapters.

High availability gives a competitive advantage to an enterprise:

Rule 40: (Darwin's extension to Murphy's Law) - *Eventually, a disaster will befall every enterprise; and only those that are prepared will survive.*

What's Next

This volume is about building cost-effective, extremely reliable systems, those exceeding six 9s of availability with uptimes measured in centuries. To achieve these goals, such systems may incur only a fraction of the cost of active/backup systems in common use. This volume focuses on the why and how of active/active systems and extends the fundamental concepts set forth in Volume 1 of this series.

The next two chapters summarize and expand the concepts presented in Volume 1. Chapter 2 analyzes why active/active systems can be so reliable. As Abraham Lincoln said, "For those who like this sort of thing, this is the sort of thing they like." If you are one who does not like math, you can skip Chapter 2. Chapter 3 summarizes the technology available today to build these systems.

Breaking the Availability Barrier – Volume 2

Later chapters then delve into advanced topics beyond those presented in Volume 1, and present a variety of case studies describing successful implementations of active/active systems.