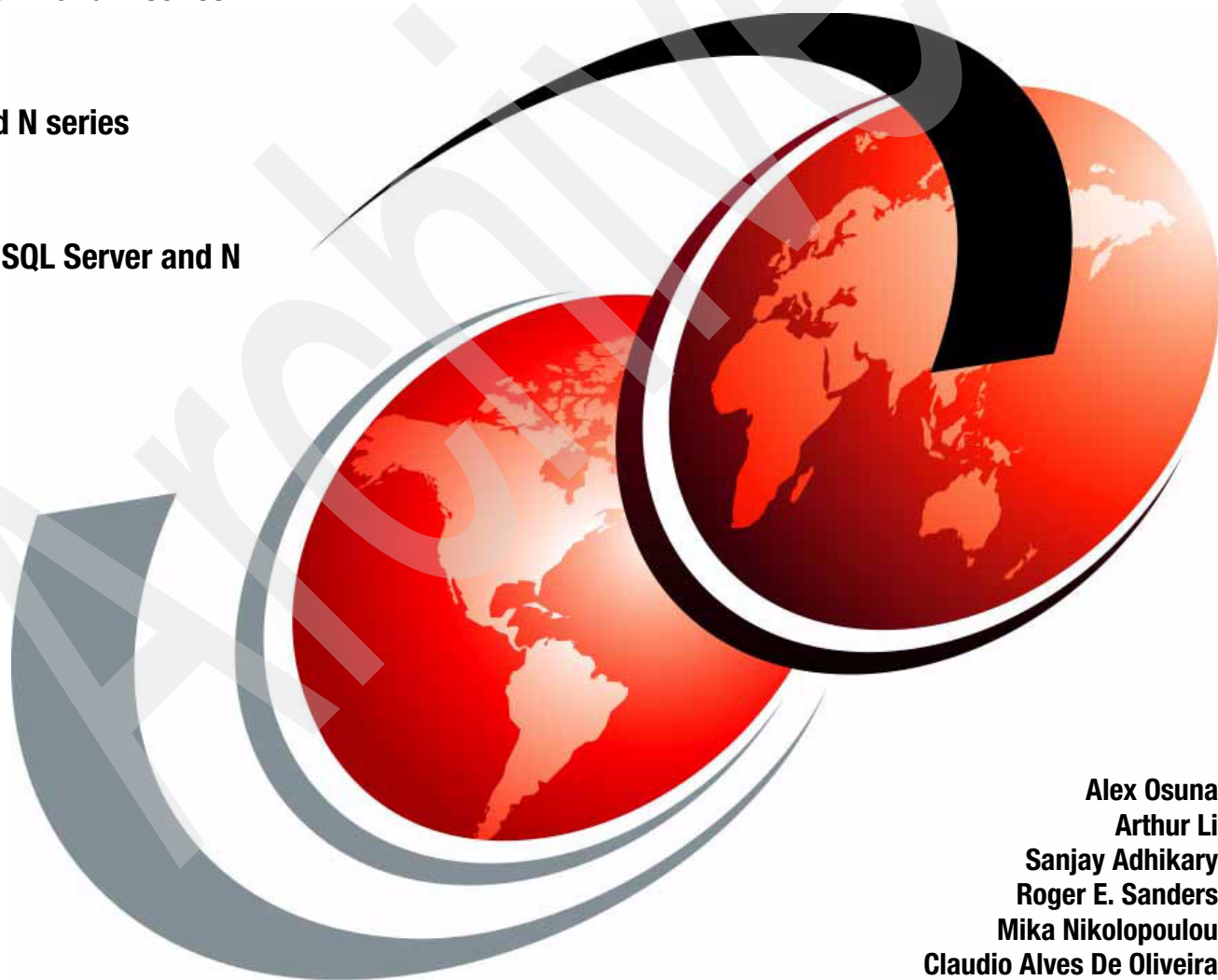


Using the IBM System Storage N Series with Databases

IBM DB2 UDB and N series

Oracle and N series

Microsoft SQL Server and N series



Alex Osuna
Arthur Li
Sanjay Adhikary
Roger E. Sanders
Mika Nikolopoulou
Claudio Alves De Oliveira

Redbooks



International Technical Support Organization

Using the IBM System Storage N Series with Databases

August 2007

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (August 2007)

This edition applies to the DATA ONTAP 7.1 and above.

© Copyright International Business Machines Corporation 2007. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this IBM Redbooks publication	xi
Become a published author	xiii
Comments welcome	xiii
Part 1. Introduction to the IBM System Storage N series	1
Chapter 1. Using IBM System Storage N series and databases	3
1.1 The IBM System Storage N series models	4
1.1.1 Comparison of the IBM System Storage N series Gateway with the IBM System Storage N series	5
1.1.2 IBM System Storage N series A models hardware quick reference	6
1.1.3 IBM System Storage N series G Models hardware quick reference	7
1.1.4 N series product A & G models hardware quick reference	8
1.2 Standard software features	8
1.3 Optional software	10
1.4 Software quick reference	11
1.5 IBM System Storage N series A models	13
1.5.1 IBM System Storage N3700 A10 and A20 introduction	14
1.5.2 N5200, N5300, N5500 & 5600 Models A10 and A20 storage systems	16
1.5.3 IBM System Storage N series 7000 models A10 and A20	18
1.6 IBM System Storage N series gateways	21
1.6.1 IBM N5200, N5300, N5500, and N5600 Gateway models	25
1.6.2 IBM Gateway models N7600 and N7800	27
1.7 LUN sizing for Gateways	27
1.8 Interoperability between Gateway G and A models	27
Chapter 2. Introduction to Data ONTAP	31
2.1 What is Data ONTAP	31
2.1.1 The network interface driver	32
2.1.2 The RAID manager	32
2.1.3 The Write Anywhere File Layout (WAFL) file system	39
2.1.4 File system consistency and Non-Volatile RAM	43
2.2 Snapshots and SnapRestore	45
2.2.1 IBM System Storage N series Snapshots	45
2.2.2 Other storage system's Snapshots	48
2.2.3 SnapRestore	50
2.3 Continuous Availability and disaster recovery	52
2.3.1 SnapMirror	52
2.3.2 SnapVault	53
2.4 Managing storage	54
2.4.1 Aggregates and RAID Groups	55
2.4.2 Introducing FlexVols	56
2.4.3 Cloning FlexVols with FlexClone	58

Part 2. Introduction to enterprise databases	63
Chapter 3. Introduction to IBM DB2 Universal Database	65
3.1 DB2 Universal Database editions	66
3.2 The universal database	66
3.2.1 Universal access	66
3.2.2 Universal application	67
3.2.3 Universal extensibility	67
3.2.4 Universal scalability	67
3.2.5 Universal reliability	68
3.2.6 Universal management	68
3.3 The DB2 query optimizer	69
3.4 DB2 utilities	69
3.4.1 Data movement utilities	69
3.4.2 Data maintenance utilities	71
3.4.3 Other utilities	72
3.5 DB2 terminology and concepts	73
3.5.1 Installing DB2	73
3.5.2 Instances	73
3.5.3 Databases	74
3.5.4 Buffer pools	75
3.5.5 Table spaces	76
3.5.6 Containers	76
3.5.7 Characteristics that affect tablespace performance	77
3.5.8 Registry and performance-related environment variables	79
3.5.9 Backup and recovery using N series	79
3.5.10 Recovery logs	80
3.5.11 The recovery history file	82
Chapter 4. Introduction to the Oracle Database	85
4.1 Oracle Database architecture	85
4.1.1 Memory structures	86
4.1.2 Logical storage structures	89
4.1.3 Physical storage structures	91
4.1.4 Processes	92
4.1.5 Administration tools	95
Chapter 5. Introduction to Microsoft SQL Server	101
5.1 Database Architecture	101
5.1.1 Logical database components	102
5.1.2 Physical database architecture	104
Part 3. Using N series and MSSQL	107
Chapter 6. Preparing IBM System Storage N series for MS Windows & MSSQL	109
6.1 Joining Active Directory	109
6.1.1 Preparing for CIFS shares	110
6.1.2 Select a user account	112
6.2 Active Directory	116
6.2.1 Should Active Directory be in mixed or native Mode?	116
6.2.2 IBM System Storage N series	117
6.2.3 Precreating a computer object	117
6.2.4 Creating the computer object	117
6.2.5 Running CIFS setup with the N series product	122

6.3 Why is it necessary to join an IBM System Storage N series to Active Directory	125
6.4 Troubleshooting the domain-joining process	126
6.4.1 DNS	126
6.4.2 Time synchronization	126
6.4.3 Active Directory replication	126
6.5 Device Discovery	126
6.6 SnapDrive set up.	127
6.6.1 Creating disk with iSCSI management	127
6.7 Conclusion	132
Chapter 7. Managing Microsoft SQL Server with IBM System Storage N series . . .	133
7.1 Managing MSSQL with the N series product	133
7.2 SnapManager	133
7.3 SnapDrive	134
7.4 SMSQL, SnapDrive, and SQL Server 2000	135
7.5 Accelerated test and development with rapid cloning	136
Chapter 8. High Availability with MSSQL and N series	137
8.1 Backup and restore databases	137
8.1.1 Backup databases	138
8.1.2 Restore databases	139
8.1.3 Backup and restore best practices	146
8.2 Mirroring	147
8.2.1 SnapMirror software operation	150
8.2.2 Disaster recovery breaking the mirror.	151
8.2.3 Resyncing the Mirror.	154
8.2.4 Conclusions	154
8.3 About Snapshots and SQL Server	154
Chapter 9. Microsoft SQL Server diagnostics and monitoring with IBM System Storage N series	157
9.1 Determining whether you have a problem	157
9.2 Checklist of services that must be active	158
9.2.1 MSSQLServer error log.	159
9.2.2 SnapManager	160
9.2.3 Event Viewer.	160
9.3 MSSQLServer Monitoring Server Performance and Activity.	161
9.3.1 Choosing a Monitoring Tool	162
9.4 IBM System Storage N series Management Tools	163
Part 4. Using the N series with IBM DB2.	167
Chapter 10. Design considerations and set up with DB2 and IBM System Storage N series	169
10.1 DB2 and the N series product design considerations	170
10.1.1 Designing a database with recovery in mind.	170
10.1.2 Tablespace management	170
10.2 Creating a sample database on the N series product in an NAS environment.	172
10.2.1 Configuring the IBM System Storage N series	173
10.2.2 Configuration and Management	175
10.2.3 Accessing the volumes on the database server	178
10.2.4 Creating the sample database on the mounts	179
10.3 Creating a sample database on N series in a SAN environment	182
10.3.1 Configuring AIX Host for FCP	182

10.3.2	Setting up LUNs on N series.	183
10.3.3	Accessing the LUNs on the database server	187
10.3.4	Creating the sample database on the LUNs	191
Chapter 11.	DB2 Continuous Availability with IBM System Storage N series.	193
11.1	Disk redundancy	193
11.2	Backup and recovery	195
11.2.1	How Snapshots work with DB2.	195
11.3	Setting up Snapshots with DB2.	198
11.3.1	System names and assumptions	198
11.3.2	Separate data from transaction logs files on N series.	198
11.3.3	Overview of DB2 UDB components for backup and recovery	198
11.3.4	Backing up a DB2 UDB database using Snapshot technology.	201
11.3.5	Accessing N series from the database server.	201
11.4	Conclusion	231
Chapter 12.	Cloning a DB2 UDB database in the IBM System Storage N series environment	233
12.1	Selecting a database server to access the cloned database	234
12.1.1	Selecting the production database server and database	234
12.1.2	Select a database server with a different DB2 UDB version.	235
12.1.3	Select a non production server without DB2 UDB installed	235
12.2	Clone an offline database on the same storage system	235
12.2.1	Bring the source database offline	235
12.2.2	Create Snapshot copies of the database FlexVol volumes.	236
12.2.3	Start the source database.	237
12.2.4	Clone the FlexVol volumes	237
12.2.5	Create an export entry for the clone volume.	238
12.2.6	Mount the clone volumes	239
12.2.7	Configuring the cloned database	241
12.2.8	Catalog the source database if necessary	243
12.2.9	Verify the database.	243
12.3	Clone an online database on the same storage system	244
12.3.1	Bring the database into a consistent state—suspend writes.	244
12.3.2	Create Snapshot copies of the database FlexVol volumes.	244
12.3.3	Resume normal database operations—resume writes	245
12.3.4	Clone the FlexVol volumes	245
12.3.5	Create NFS export entries for the cloned volumes	246
12.3.6	Mount the cloned volumes	246
12.3.7	Configuring the cloned database	248
12.3.8	Verify the database.	250
12.4	Cloning an offline database to a remote storage system	250
12.4.1	Configuring SnapMirror.	251
12.4.2	Initializing SnapMirror	254
12.4.3	Create clones of the FlexVol volumes.	256
12.4.4	Creating NFS export entries for the cloned volumes.	259
12.4.5	Mounting the clone volumes	259
12.4.6	Configuring the cloned database	260
12.4.7	Cataloging the source database	263
12.4.8	Verifying the cloned database.	263
12.5	Cloning an online database to a remote storage system	264
12.5.1	Configuring and initialize SnapMirror	264
12.5.2	Bring the source database into a consistent state—suspend writes.	264
12.5.3	Creating Snapshot copies of the FlexVol volumes	265

12.5.4 Resuming normal database operations—resume writes	265
12.5.5 Updating the SnapMirror destination.	265
12.5.6 Creating clone volumes using Snapshot copies	266
12.5.7 Creating NFS export entries for the cloned volumes	267
12.5.8 Mounting the cloned volumes	267
12.5.9 Configuring the cloned database	269
12.5.10 Verifying the cloned database.	271
Chapter 13. DB2 and N series performance considerations.	273
13.1 General performance guidelines	274
13.2 IBM System Storage N series and DB2 tuning	275
13.3 db2_parallel_io	277
13.4 A word about db2empfa	278
13.5 Quick-start tips for general DB2 performance tuning	278
Chapter 14. DB2 and AIX diagnostics and performance monitoring	281
14.1 The DB2 Database System Monitor	282
14.1.1 The snapshot monitor	282
14.1.2 Event monitors	289
14.2 Operating system monitoring tools	291
14.2.1 topas	291
14.2.2 vmstat	294
14.2.3 iostat	295
14.3 IBM System Storage N series monitoring tools.	296
14.4 Performance factors	297
14.5 The Performance toolchest.	297
14.5.1 Performance and monitoring.	297
Part 5. Using IBM System Storage N series with Oracle.	305
Chapter 15. Backup, restore, and disaster recovery of Oracle using N series	307
15.1 How to back up data from N series	308
15.1.1 Creating online backups using Snapshot copies	309
15.1.2 Recovering individual files from a Snapshot copy.	309
15.1.3 Recovering data using SnapRestore	309
15.1.4 Backup and recovery best practices	310
15.1.5 SnapManager for Oracle—Backup and recovery best practices	315
15.1.6 SnapManager for Oracle—Automatic Storage Management-based backup and restore.	316
15.1.7 SnapManager for Oracle—RMAN based backup and restore	317
15.1.8 SnapManager for Oracle—cloning	318
Chapter 16. NFS performance tuning for N series and Oracle Database.	319
16.1 Introduction	320
16.2 Focus on OLTP Workloads.	320
16.3 Deploying databases with IBM System Storage N series and NFS	321
16.3.1 Network Attached Storage versus Storage Area Networks	321
16.3.2 NAS Protocols.	322
16.3.3 Why NFS.	322
16.3.4 Database I/O Patterns	323
16.4 Tuning a Linux, N series product, Oracle environment.	324
16.4.1 Oracle tuning.	324
16.5 Linux NFS client performance.	327
16.5.1 Identifying Kernel Releases	327

16.5.2	Choosing a Network Transport Protocol	327
16.5.3	Linux—Kernel Patches	327
16.5.4	Enlarging a client's Transport Socket Buffers	327
16.5.5	Other TCP enhancements	328
16.5.6	Linux networking—Gigabit Ethernet Network Adapters	328
16.5.7	Linux networking—jumbo frames with GbE	329
16.5.8	Special mount options.	329
16.5.9	IBM System Storage N series tuning	330
16.5.10	Volume layout	330
16.5.11	Placement of Oracle home directory.	331
16.5.12	Placement of Oracle data and log Files	332
16.5.13	NFS settings	332
16.5.14	Volume options	333
Chapter 17.	IBM Storage System N series best practice guidelines for Oracle	335
17.1	IBM System Storage N series system configuration	336
17.1.1	IBM System Storage N series settings	336
17.1.2	Ethernet—Gigabit Ethernet, auto negotiation, and Full Duplex	336
17.2	Volume, aggregate setup and options	337
17.2.1	Databases	337
17.2.2	Aggregates and FlexVol volumes or traditional volumes	337
17.2.3	Volume Size	338
17.2.4	Recommended volumes for Oracle Database files and logfiles	338
17.2.5	Oracle Optimal Flexible Architecture (OFA) on the N series product	338
17.2.6	Oracle home location	339
17.2.7	Best Practices for control and log files on N series.	341
17.3	RAID Group Size.	343
17.3.1	RAID-DP	343
17.4	Snapshot and SnapRestore	343
17.5	Snap Reserve	345
17.6	The N series storage system options	347
17.6.1	The minra option.	347
17.6.2	File access time update	347
17.6.3	NFS settings	348
17.7	Operating systems	348
17.7.1	Linux recommended versions	348
17.7.2	Linux OS settings	350
17.8	Oracle Database settings	352
17.8.1	DISK_ASYNCH_IO.	352
17.8.2	DB_FILE_MULTIBLOCK_READ_COUNT	353
17.8.3	DB_BLOCK_SIZE.	354
17.8.4	DBWR_IO_SLAVES and DB_WRITER_PROCESSES	354
17.8.5	DB_BLOCK_LRU_LATCHES.	354
	Related publications	355
	IBM Redbooks publications	355
	Other publications	355
	Online resources	355
	How to get IBM Redbooks publications	356
	Help from IBM	356
	Index	357

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
developerWorks®
DB2 Connect™
DB2 Extenders™
DB2 Universal Database™
DB2®

Informix®
Intelligent Miner™
IBM®
OS/2®
OS/400®
Redbooks®

Redbooks (logo) ®
System Storage™
TotalStorage®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Snapshot, SecureAdmin, RAID-DP, FlexVol, Network Appliance, WAFL, SyncMirror, SnapVault, SnapValidator, SnapRestore, SnapMover, SnapMirror, SnapManager, SnapDrive, NearStore, FilerView, Data ONTAP, NetApp, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

Flex, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Java, Nearline, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, Microsoft, SQL Server, Visual Basic, Windows NT, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

"Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation."

Xeon, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication discusses how to optimize the IBM System Storage™ N series products with some of the major commercially available databases available to customers today. Topics include installation, performance, monitoring, and management when using the IBM System Storage N series with IBM DB2®, Microsoft® SQL, and Oracle®. We also cover best practices and tips for using the IBM System Storage N series with these major database applications.

The team that wrote this IBM Redbooks publication

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Alex Osuna is a Project Leader with the International Technical Support Tucson Center. He writes extensively and teaches IBM classes worldwide on all areas of storage. Alex has been in the I/T field for over 27 years and specialized the majority of those years in storage hardware and software in areas of field engineering, field support, development, early ship programs, performance analysis, and education. He holds over 10 certifications from IBM, Microsoft, and Red Hat.

Arthur Li is an IBM certified DB2 UDB Solutions Expert with the DB2 UDB Advanced Support team at the IBM Toronto Lab, Canada. He has been with the IBM Database Technologies organization since 2000 and specializes in the areas of connectivity, connection pooling/concentrator, security, federated database access, and Query Patroller. Arthur holds a Bachelor of Science degree in Software Engineering from the University of Toronto.

Sanjay Adhikary is a Senior System Analyst handling cross platform databases for the IBM STC project. He is an Oracle Certified DBA, IBM certified DB2 UDB DBA and a Microsoft Certified Solutions Developer. He has 11 years of total IT experience out of which 7 years as a core Database administrator. He is a post graduate in Business Management from NMIMS Mumbai.

Claudio Alves De Oliveira is a Senior Database Administrator (DBA) handling cross platform databases for the GCM-CRM project at ABM Amro Bank. He is certified as a Microsoft Certified professional DBA and IBM certified DB2 UDB DBA for UNIX®, Linux®, and Windows®. He is post graduate in Information Systems from Fundação Santo André with 15 years of total experience in the IT industry and six years focused on Database Administration.

Roger E. Sanders is a Senior Manager - IBM Alliance Engineering with Network Appliance™, Inc. He has been designing and developing databases and database applications for more than 20 years and has worked with DB2 Universal Database™ and its predecessors since it was first introduced on the IBM PC (as part of OS/2® 1.3 Extended Edition). He has written articles for IDUG Solutions Journal and Certification Magazine, authored DB2 tutorials for the IBM developerWorks® Web site, presented at several International DB2 User's Group (IDUG) and regional DB2 User's Group (RUG) conferences, taught classes on DB2 Fundamentals and Database Administration (DB2 8.1 for Linux, UNIX, and Windows), and is the author of nine books on DB2 and one book on ODBC. Roger is also a member of the DB2 Certification Exam development team and the author of a regular column (Distributed DBA) in DB2 Magazine.

Mika Nikolopoulou is an Information Management Technical Sales Representative for New York/New Jersey Metropolitan area out of New York. She holds the IBM Senior IT Specialist Certification. Mika has been in the database technical field for over 12 years and specializes in Very Large Databases (VLDB), Performance and Tuning, High Availability, Federated databases, and Business Intelligence architecture. She is IBM Certified in DB2 UDB Advanced Administration and Business Intelligence Solutions and holds seven IBM DB2 UDB Product Certifications. She holds a Bachelor and a Masters Degree with Honors in Computer Engineering from University of Patras, Greece and a Masters Degree in Business Administration from the Stern Business School, New York University.



Figure 0-1 The Team Arthur Li, Alex Osuna, Roger Sanders, Mika Nikolopoulou, Claudio Alves De Oliveira, Sanjay Adhikary

Thanks to the following Network Appliance personnel for their contributions to this project:

Andrew Firebaugh

Eric Barrett, Bikash R. Choudhury

Bruce Clarke

Blaine McFadden

Tushar Patel

Ed Hsu

Christopher Slater

Michael Tatum
Glenn Colaco
Darrell Suggs
Chuck Lever
John Phillips
Gerson Finlev
Dwayne Jones
Garth Zoller
Gerson Finlev
Maryam Basiji
Gerson Finlev
Maryam Basiji
IBM
Prashanta Kumar Goswami

Become a published author

Join us for a two-to-six week residency program! Help write an IBM Redbooks publication dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will team with IBM technical professionals, Business Partners, and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at the following Web site:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbooks publications form found at:

ibm.com/redbooks

- Send your comments in an email to:

redbook@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Part 1

Introduction to the IBM System Storage N series

In this part we introduce the basics of the IBM System Storage N series product line and its basic features and capabilities as well as its connectivity capabilities.

Archived

Using IBM System Storage N series and databases

In this chapter we introduce the IBM System Storage IBM N series and the benefits and features in the database environment.

One of the primary benefits of using an IBM System Storage N series in your database environment is that the IBM System Storage N series is a Multiprotocol storage system providing network access protocols such as NFS, CIFS, HTTP, and iSCSI as well as Storage Area technologies such as Fibre Channel (see Figure 1-1).

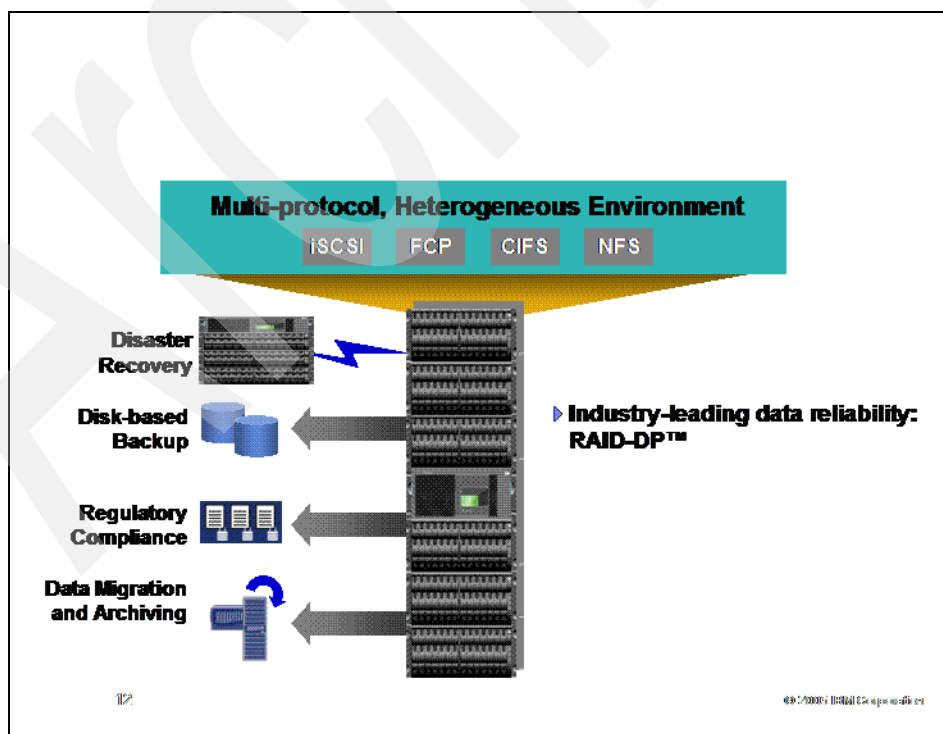


Figure 1-1 Multiprotocol

Most databases used today consist of critical and irreplaceable data. The N series product uses built-in RAID technologies as well as protection with options to add additional protection through mirroring, replication, snapshots, and backup.

With shrinking staff and additional duties that the Database administrator (DBA) has today, the administration of storage is made easier with the N series product administrative interfaces of FilerView®, Operations Manager, and its command line interface. These simple management interfaces make installing, administering, and troubleshooting uncomplicated and straightforward.

Though the IBM System Storage N series is suited for many environments, including the database environment, the advantages of using it with your database applications are as follows:

- ▶ The ability to tune the storage environment to your database application while maintaining flexibility to increase, decrease, or change access methods with a minimum of disruption.
- ▶ The capability to react easily and quickly to changing storage requirements. If additional storage is required, being able to expand it quickly and non-disruptively is needed. When existing storage exists but is deployed incorrectly, the capability to reallocate available storage from one application to another quickly and simply cannot be done.
- ▶ The ability to maintain availability and productivity during upgrades. Keeping required outages to the shortest time possible.
- ▶ Create effortless backup/recovery solutions that operate commonly across all data access methods.
- ▶ File and block level services in a single system, helping to simplify your infrastructure.

1.1 The IBM System Storage N series models

- ▶ IBM System Storage N3700 storage system
- ▶ IBM System Storage N5000 storage system
- ▶ IBM System Storage N7000 storage system
- ▶ IBM System Storage N5000 Gateway
- ▶ IBM System Storage N7000 Gateway








Use	Model	Capacity
Entry Level	N3700 A10&A20 	16TB
Midrange	N5200 A10 &A20 	84TB
Midrange	N5300 A10 &A20 	126TB
Midrange	N5500 A10&A20 	168TB
Midrange	N5600 A10&A20 	210TB A10 252TB A20
Enterprise class	N7600 A10&A20 	420TB
Enterprise class	N7800 A10&A20 	504TB

Figure 1-2 IBM System Storage N series

Use	Model	Capacity
Midrange	N5200 G10 &G20 	50TB
Midrange	N5300 G10 & G20 	126TB
Midrange	N5500 G10&G20 	80TB
Midrange	N5600 G10&G20 	252TB
Enterprise class	N7600 G10&G20 	420TB
Enterprise class	N7800 G10&G20 	504TB

Figure 1-3 IBM System Storage N series gateways

1.1.1 Comparison of the IBM System Storage N series Gateway with the IBM System Storage N series

- ▶ Identical core NAS feature/functionality*
- ▶ Identical iSCSI feature/functionality
- ▶ Identical FCP feature/functionality
- ▶ Filer SAN host support matrix applies

- ▶ Identical behavior for Write Anywhere File Layout (WAFL®) file system
- ▶ Identical data availability characteristics*
- ▶ Identical data integrity characteristics*
- ▶ Identical data management characteristics*
- ▶ Identical serviceability characteristics*
- ▶ Supports same version of Data ONTAP®
- ▶ Differences exist in system initialization and storage expansion
- ▶ A N5000 and N 7000 series Gateway physical attributes are the similar to the N5000 and N7000 models A10 and A20 storage systems.
- ▶ The N series Gateway does not use the following features of Data ONTAP.
 - SnapLock Compliance
 - Disk sanitization and disk scrubbing
 - SnapLock
 - Compliance
 - LockVault Compliance
 - Nearline™ Feature
 - RAID-DP™
 - RAID4
- ▶ N5000 and N7000 storage systems use disk storage provided by IBM only. The Gateway models support heterogeneous storage.
- ▶ Data ONTAP is enhanced to enable the Gateway Series solution.
- ▶ A RAID array provides LUNs to the Gateway:
 - Each LUN is equivalent to an IBM disk
 - LUNs are assembled into aggregates/volumes and then formatted with WAFL file system, just like the IBM System Storage N series.

1.1.2 IBM System Storage N series A models hardware quick reference

Table 1-1 A models hardware quick reference

Function	N3700	N5200	N5300	N5500	N5600	N7600	N7800
Maximum Raw Capacity in TB A10 models	16	84	126	168	210	336	336
Maximum Raw Capacity in TB A20 models	16	84	126	168	252	420	504
Fibre Channel Disk drives	144 GB 10K RPM, 144 GB 15K RPM, 300 GB 10K RPM - EXN2000 144GB 15K, 300GN 10K - EXN4000						

Function	N3700	N5200	N5300	N5500	N5600	N7600	N7800
SATA Disk Drives	250 GB 7.2 K RPM., 320 GB 7.2 K RPM., 500 GB 7.2 K RPM						
Maximum number of disks	56	168	252	336	420(A10) 504(A20)	672(A10) 840(A20)	672 (A10) 1008 (A20)
Maximum Raw Capacity in TB based on SATA Drive type	Capacity limited N/A	42 .00@ 250GB 53.76 @ 320GB 84 .00@ 500GB	63@250 GB 126@500 GB	84.00@ 250GB 107.52@ 320GB 168.00@ 500GB	105@250GB(A10) 126@250GB(A20) 134@320GB(A10) 161@320GB(A20) 210@500GB(A10) 252@500GB(A20)	168@250GB(A10) 210@250GB(A20) 215@320GB(A10) 268@320GB(A20) 336@500GB(A10) 420@500GB(A20)	168@250GB(A10) 252@250GB(A20) 215@320GB(A10) 322@320GB(A20) 336@500GB(A10) 504@500GB(A20)
Expansion units supported	EXN1000 (SATA), EXN2000 (FC), EXN4000 (FC)						

1.1.3 IBM System Storage N series G Models hardware quick reference

Table 1-2 IBM System Storage N series G models quick reference

Function	N5200	N5300	N5500	N5600	N7600	N7800
Maximum Raw Capacity in TB G10 models	50	126	84	252	336	336
Maximum Raw Capacity in TB G20 models	50	126	84	252	420	504
Max. number of – Logical Units (LUNs) on back-end disk storage array	168	252	336	420 for A10 and 504 for A20	840	1008
Max LUN size in GB	500	500	500	500	500	500
Maximum Volume size in TB	16	16	16	16	16	16

Note: A stand-alone gateway must own at least one LUN. A cluster configuration must own at least two LUNs.

1.1.4 N series product A& G models hardware quick reference

Table 1-3 Storage System Reference

Function	N3700	N5200	N5300	N5500	N5600	N7600	N7800
Network Protocol support	NFS V2/V3/V4 over UDP or TCP, PCNFSD V1/V2 for (PC) NFS client authentication, Microsoft CIFS, iSCSI, FCP, VLD, HTTP 1.0, HTTP1.1 Virtual Host						
Other Protocol Support	SNMP, NDMP, LDAP, NIS, DNS						
Onboard I/O ports per node	2 X GbE 2 X Optical FC	4 X GbE 4 X FC 1 X LVD SCSI	4 X GbE 4 X FC 1 X LVD SCSI	4 X GbE 4 X FC 1 X LVD SCSI	4 X GbE 4 X FC (4Gbps)	6 X GbE 8 X FC	6 X GbE 8 X FC
PCI expansion slots per node	N/A	3 X PCI-X	6xPCIe	3 X PCI-X	3 X PCI-E	5 X PCI-E, 3 X PCI-X	5 X PCI-E, 3 X PCI-X
NVRAM in MB per node	128	512	512	512	512	512	2048
Memory in GB per node	1	2	4	4	8	16	32
Redundancy/ High Availability	CompactFlash, dual-redundant hot-plug integrated cooling fans, hot-swappable autoranging power supplies, clustered storage controllers, hot-swappable disk bays**						
Required rack space	3U	3U per node	3U per node	3U per node	3U per node	6U per node	6U per node
Processors (A10)	Two Broadcom MIPS-based	one 2.8 GHz Xeon®	two 1.8 GHz AMD	two 2.8 GHz Xeon	2 AMD 1.8GHz	two 2.6 GHz AMD Opteron	four 2.6 GHz AMD Opteron
Processors (A20)	Four Broadcom MIPS-based	two 2.8 GHz Xeon	four 1.8 GHz AMD	four 2.8 GHz Xeon	4 AMD 1.8GHz	four 2.6 GHz AMD Opteron	eight 2.6 GHz AMD Opteron

1.2 Standard software features

The following features come standard with the N series product. The ones highlighted in bold represent the features that compliment Database applications.

Table 1-4 Standard software features

Data ONTAP	Operating system software that optimizes data serving and allows multiple protocol data access.
FTP	File Transfer Protocol (FTP), a standard Internet protocol is a simple way to exchange files between computers on the Internet.
Telnet	The TELNET Protocol provides a general, bi-directional, eight-bit byte oriented communications facility. It provides user-oriented command line login sessions between hosts.

Snapshot™	Enables online backups, providing near instantaneous access to previous versions of data without requiring complete, separate copies.
FlexVol™	FlexVol creates multiple flexible volumes on a large pool of disks. Dynamic, nondisruptive (thin) storage provisioning, space, and time-efficiency. These flexible volumes can span multiple physical volumes with no regard to size.
FlexShare	FlexShare gives administrators the ability to leverage existing infrastructure and increase processing utilization without sacrificing the performance of critical business needs. With the use of FlexShare, administrators can confidently consolidate different applications and data sets on a single storage system. FlexShare gives administrators the control to prioritize applications based on how critical they are to the business.
FlexCache	FlexCache has the ability to distribute files to remote locations without the need for continuous hands-on management. Storage systems deployed in remote offices automatically replicate, store, and serve the files or file portions that are requested by remote users without the need for any replication software or scripts.
Disk Sanitization	Disk sanitization is the process of physically obliterating data by overwriting disks with specified byte patterns or random data in a manner that prevents recovery of current data by any known recovery methods. This feature enables you to carry out disk sanitization by using three successive byte overwrite patterns per cycle. By default six cycles are performed.
FilerView	A Web-based administration tool that allows IT administrators to fully manage N3700 systems from remote locations. Simple and intuitive Web-based single-appliance administration.
SnapMover®	Migrates data among N3700 clusters with no impact on data availability and no disruption to users.
AutoSupport	AutoSupport is a sophisticated, event-driven logging agent featured in the Data ONTAP operating software and inside each IBM System Storage N series, which continuously monitors the health of your system and issues alerts if a problem is detected. These alerts can also be in the form of E-mail.
SecureAdmin™	SecureAdmin is a Data ONTAP module that enables authenticated, command-based administrative sessions between an administrative user and Data ONTAP over an intranet or the Internet.
DNS	The N series product supports using a host naming file or a specified DNS server and domain.
Cluster Failover	Ensures high data availability for business-critical requirements by eliminating a single point of failure. -Must be ordered for A20 clustered configurations or upgrades from A10 to A20. -Active-active pairing delivers even more “nines to right of the decimal point”.
NIS	The N series product does provide NIS client support and can participate in NIS domain authentication.
Integrated automatic RAID manager	The IBM System Storage N series and Data ONTAP provide integrated RAID management with RAID-Double Parity (default) and RAID 4.
iSCSI Host Attach Kit for AIX®, Windows, Linux	A Host Support Kit includes support software and documentation for connecting a supported host to an iSCSI network. The support software includes programs that display information about storage and programs to collect information needed by customer support to diagnose problems.

1.3 Optional software

Table 1-5 Optional Software

CIFS	Provides File System access for Microsoft Windows environments.
NFS	Provides File System access for Unix and Linux environments.
HTTP	Hypertext Transfer Protocol allows a user to transfer displayable Web pages and related files.
FlexClone	Designed to provide instant replication of data volumes/sets without requiring additional storage space at the time of creation.
Multistore	Permits an enterprise to consolidate a large number of Windows, Linux, or UNIX file servers onto a single storage system. Many “virtual filers” on one physical appliance ease migration and multi-domain failover scenarios.
SnapLock	Provides non-erasable and non-rewritable data protection that helps enable compliance with government and industry records retention regulations.
LockVault	Designed to provide non-erasable and non-rewritable copies of Snapshot data to help meet regulatory compliance needs for maintaining backup copies of unstructured data.
SnapMirror®	Remote mirroring software that provides automatic block-level incremental file system replication between sites. Available in synchronous, asynchronous, and semi-synchronous modes of operation.
SnapRestore®	Allows rapid restoration of the file system to an earlier point-in-time, typically in only a few seconds.
SnapVault®	Provide disk-based backup for N3700 systems by periodically backing up a Snapshot copy to another system.
SnapDrive®	SnapDrive enables Windows and Unix applications to access storage resources on IBM System Storage N series, which are presented to the Windows 2000 or later operation system as locally attached disks. For Unix it allows you to create storage on a storage system in the form of LUNs, file systems, logical volumes, or disk groups.
SnapManager®	Host software for managing Exchange and SQL Server™ backup and restore. SnapManager software simplifies Exchange data protection by automating processes to provide hands-off, worry-free data management.
SnapValidator®	For Oracle deployments, SnapValidator provides an additional layer of integrity, checking between the application and IBM System Storage N series. SnapValidator allows Oracle to create checksums on data transmitted to IBM System Storage N series for writes to disk and include the checksum as part of the transmission.
SyncMirror®	SyncMirror is a synchronous mirror of a volume. It maintains a strict physical separation between the two copies of your mirrored data. In case of an error in one copy, the data is still accessible without any manual intervention.

Single Mailbox Recovery for Exchange	SMBR is a software option from SnapManager that is designed to take near-instantaneous online backups of Exchange databases, verify that the backups are consistent, and rapidly recover Exchange within levels: storage group, database, folder, single mailbox, or single message. The potential results are improved service to internal clients, reduced infrastructure expenses, and significant time savings for Exchange administrators.
Operations Manager	Operations Manager provides remote, centralized management of the N series product's data storage infrastructure, including global enterprise, storage network, and so forth.
MetroCluster	MetroCluster software provides an enterprise solution for high availability over wide area networks.
NearStore® option	A disk-based, secondary storage device for enterprise applications.
Advanced Single Instance Storage	Designed to significantly improve physical storage efficiency and network efficiency by enabling the sharing of duplicate data blocks.

1.4 Software quick reference

This quick reference will indicate what software features are available with respect to N series models.

Table 1-6 Software quick reference

Product/feature/function	Included/optional	N3700 A10 & A20	N5X00 A10 & A20	N5x00 G10 & G20	N7x00 A10 & A20	N7x00 G10 & G20
Data ONTAP	Included	X	X	X	X	X
iSCSI protocol	Included	X	X	X	X	X
FTP protocol	Included	X	X	X	X	X
NDMP protocol	Included	X	X	X	X	X
FlexVol	Included	X	X	X	X	X
Snapshot	Included	X	X	X	X	X
SecureAdmin	Included	X	X	X	X	X
iSCSI Host Attach Kit for AIX, Windows, Linux	Included	X	X	X	X	X
FlexShare	Included				X	X
SnapMover	Included	X	X	X	X	X
CIFS protocol	Optional	X	X	X	X	X
NFS protocol	Optional	X	X	X	X	X
HTTP protocol	Optional	X	X	X	X	X

Product/feature/ function	Included/ optional	N3700 A10 & A20	N5X00 A10 & A20	N5x00 G10 & G20	N7x00 A10 & A20	N7x00 G10 & G20
FCP protocol	Optional	X	X	X	X	X
FlexClone	Optional	X	X	X	X	X
Clustered Failover	Optional	X(A20)	X	X	X	X
Multistore	Optional	X	X	X	X	X
SnapMirror	Optional	X			X requires special HBA card	X requires special HBA card
SnapRestore	Optional	X	X	X	X	X
Open Systems SnapVault (OSSV)	Optional	X	X	X	X	X
SnapVault	Optional	X	X	X	X	X
SnapDrive for Windows & UNIX: AIX, Solaris™, HP-UX, Linux	Optional	X	X	X	X	X
SnapValidator	Optional	X	X	X	X	X
SyncMirror	Optional		X	X	X	X
SnapManager for SQL Server	Optional	X	X	X	X	X
SnapManager for Exchange	Optional	X	X	X	X	X
Single Mailbox Recovery for Exchange (SMBR)	Optional	X	X	X	X	X
Operations Manager Core, BC & SRM License	Optional	X	X	X	X	X
SnapLock Enterprise	Optional	X	X	X	X	X
LockVault Enterprise	Optional	X	X	X	X	X
MetroCluster A20 Models only	Optional		X	X	X	X
Disk Sanitization	Included	X	X		X	
SnapLock Compliance	Optional	X	X		X	

Product/feature/function	Included/optional	N3700 A10 & A20	N5X00 A10 & A20	N5x00 G10 & G20	N7x00 A10 & A20	N7x00 G10 & G20
LockVault Compliance	Optional	X	X		X	
NearStore Bundle	Optional	X	X		X	
RAID4, RAID-DP	Included	X	X		X	
Advanced Single Instance Storage	Optional		X	X	X	X

1.5 IBM System Storage N series A models

The A models of the IBM System Storage N series offer multiprotocol connectivity using internal storage or storage provided by expansion units (see Figure 1-4 on page 14). The IBM System Storage N series systems are designed to provide integrated block and file-level data access, allowing concurrent operation in IP SAN (iSCSI), FC SAN, NFS, and CIFS environments. Other storage vendors may require the operation of multiple systems to provide this functionality. IBM System Storage N series are designed to avoid costly downtime, both planned and unplanned, and to improve your access to mission-critical data, thereby helping you gain a competitive advantage.

- ▶ FC drive options
 - Used for high performance, mission critical production data
- ▶ SATA options can be used for the following:
 - Disk backup options (Nearline storage)
 - Archive storage WORM protection data

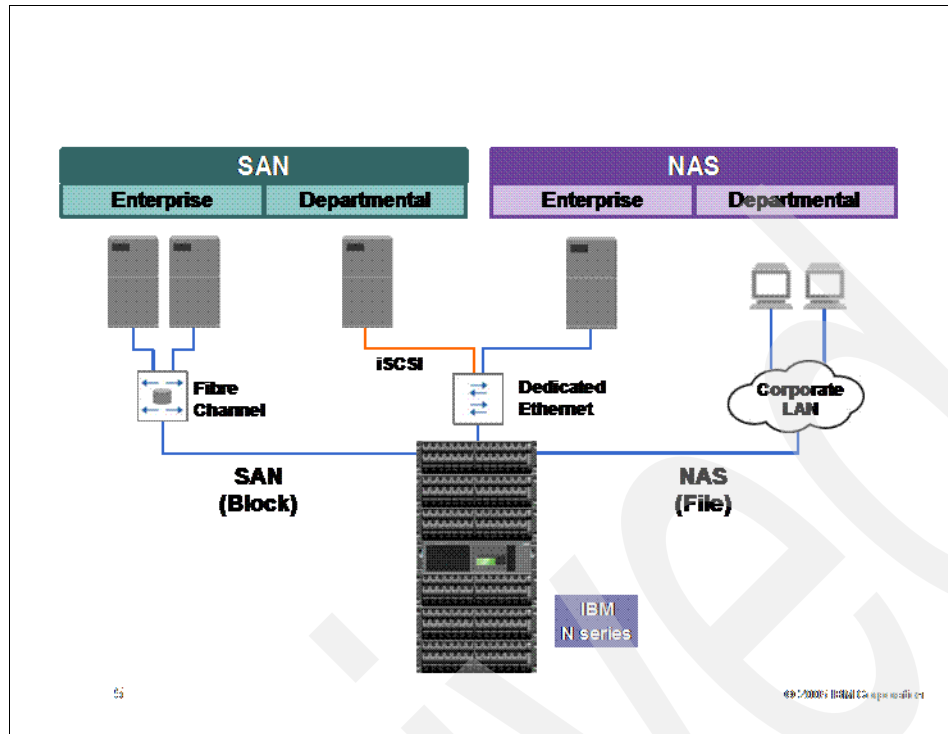


Figure 1-4 IBM System Storage N series A models

1.5.1 IBM System Storage N3700 A10 and A20 introduction

The N3700 storage system is a 3U solution designed to provide NAS and iSCSI functionality for entry to mid-range environments. The basic N3700 offering is a single-node model A10, which is upgradeable to the dual-node model A20 and requires no additional rack space. The dual-node, clustered A20, is designed to support fail over and failback functions to maximize reliability. The N3700 filer can support 14 internal hot-plug disk drives with scalability being provided through attachment to up to three expansion units, each with a maximum of 14 drives. The N3700 can also connect to a Fibre Channel switch for or tape for backup.



Figure 1-5 N3700

The single node A10 uses a single control unit with dual node clustered A20 using two control units (Figure 1-6).



Figure 1-6 N3700 A20

The N3700 is based around a MIPS dual core processor. It has 1GB of system memory of which 128MB is defined as non-volatile due to having a battery back up.

Note: The NVRAM is a battery backed up portion of the main system memory.

N3700 Hardware Features

- ▶ 3U integrated storage system
- ▶ 3U optional storage expansion shelf - up to three
- ▶ Redundant hot plug power supplies
- ▶ Redundant Cooling
- ▶ Integrated 10/100/1000 full duplex Ethernet
- ▶ 2 Integrated Fibre Channel adapters
- ▶ Compact Flash
- ▶ Diagnostic LEDs/OPS

Though ESH2 modules in the EXN2000 support up to 84 drives per loop or one N3700 and five expansion shelves, only 56 drives and three expansion shelves and one N3700 are supported. This is a firmware limitation for backward compatibility by the manufacturer for the previous shelf module, the Loop Redundant Circuit (LRC). This module is not available in the N series product.

Optional Hardware

- Second CPU tray supporting Cluster Failover

Table 1-7 Additional N3700 specifications

Storage System Specifications	N3700 A10	N3700 A20
Clustered Failover-Capable	No (Requires upgrade to A20)	Yes
Max Number of Expansion Units EXN1000/SATA disk drives or EXN2000/FC disk drives	3	3

1.5.2 N5200, N5300, N5500 & 5600 Models A10 and A20 storage systems

The N5200, N5300, N5500, and N5600 are suitable for environments that demand data in high availability, high capacity, and highly secure data storage solutions. The IBM System Storage N5000 series offers additional choice to organizations for enterprise data management. The IBM System Storage N5000 series is designed to deliver high-end enterprise storage and data management value with midrange affordability. Built-in enterprise serviceability and manageability features help support your efforts to increase reliability, simplify and unify storage infrastructure and maintenance, and deliver exceptional economy.

- The IBM N5000 A series comes in two Models A10 and A20:
 - N5200
 - 2864-A10 Single Filer
 - 2864-A20 Clustered
 - N5300
 - 2869-A10 Single Filer
 - 2869-A20 Clustered
 - N5500
 - 2865-A10 Single Filer
 - 2865-A20 Clustered
 - N5600
 - 2868 - A10
 - 2868 -A20
- FC or SATA (both may be used behind a single controller but not in the same drawer)

The N5000 A10 models come in a compact 3U rack mountable units that can coexist in the same rack as an EXN1000, EXN2000, and EXN4000 storage expansion unit (see Figure 1-7 on page 17 and Figure 1-8 on page 17). The A20 models require 6U of space.

There are no visible external differences between the N5200 and the N5500. The differences are in maximum storage capacity and CPU processing power as illustrated in Table 1-2 on page 7. From the front the N5600 and the N5300 also look very similar to the N5200 and the N5500. Some of the differences are seen from the rear (Figure 1-9 on page 18), especially the absence of a LVD iSCSI connector. The N5600 and N5300 also use a BIOS prompt upon boot rather than a Common Firmware Environment (CFE) prompt.



Figure 1-7 N5000 A model

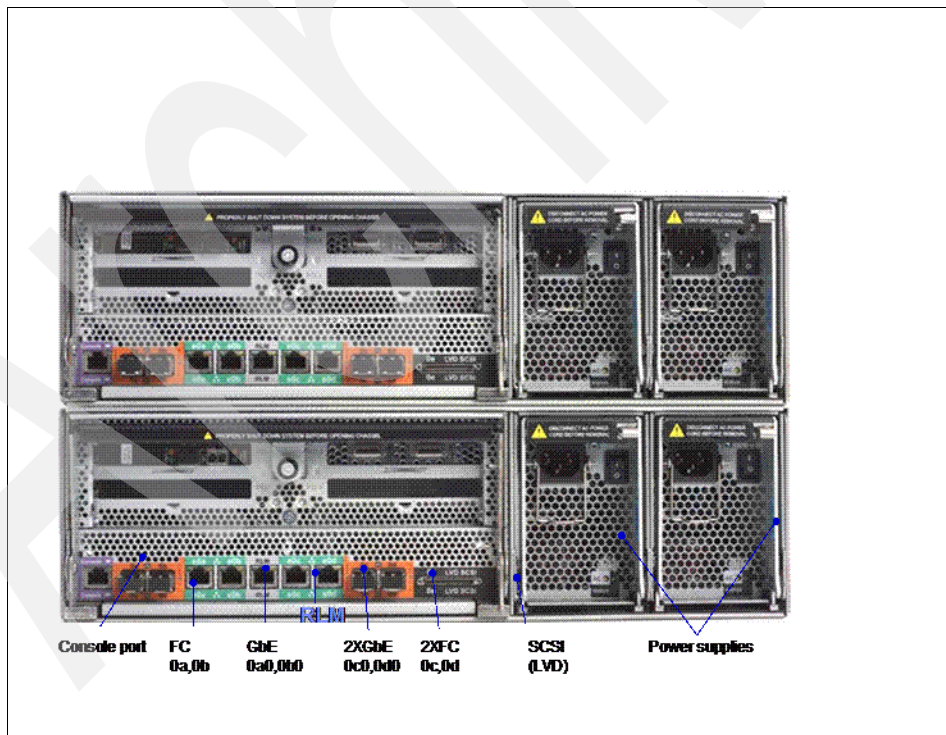


Figure 1-8 N5200 and N5500 series A20 rearview

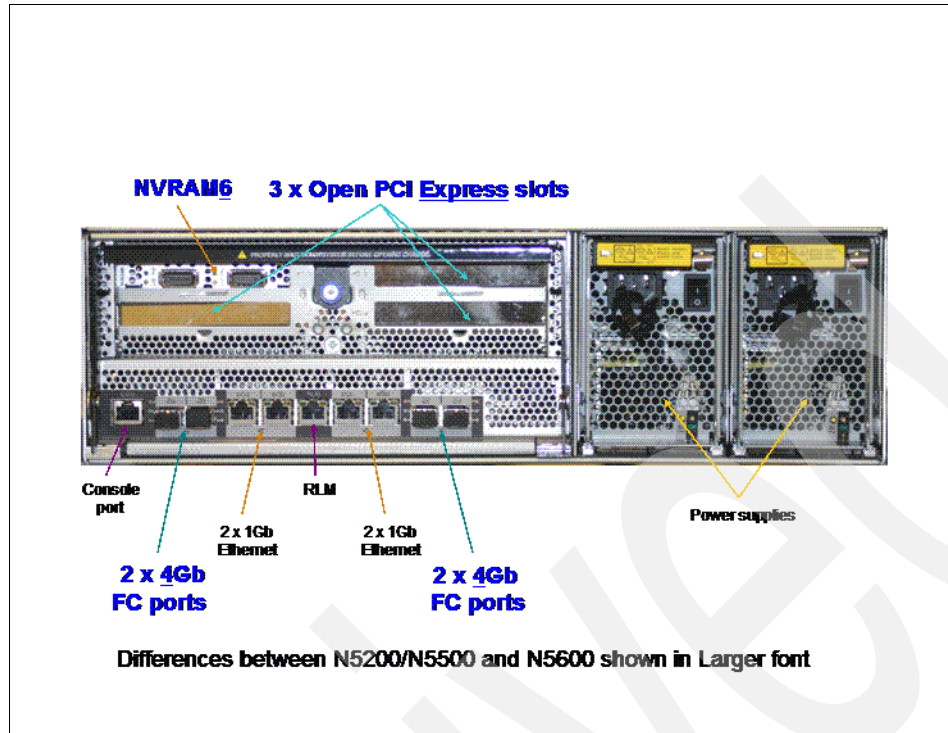


Figure 1-9 N5600 and N5300 rearview

There are no visible differences between the N5200 and N5500. The differences are in maximum storage capacity and CPU processing power as illustrated in Table 1-3 on page 8.

1.5.3 IBM System Storage N series 7000 models A10 and A20

The IBM System Storage N7000 series offers additional choices to organizations facing the challenges of enterprise data management. The IBM System Storage N7000 series is designed to deliver high-end enterprise storage and data management value with midrange affordability. Built-in enterprise serviceability and manageability features help support your efforts to increase reliability, simplify and unify storage infrastructure and maintenance, and deliver exceptional economy.

- The IBM N7000 A series comes in two Models, A10 and A20:
 - N7600
 - 2866-A10 Single Node
 - 2866-A20 Clustered
 - N7800
 - 2867-A10 Single Node
 - 2867-A20 Clustered
- FC or SATA (both may be used behind a single controller but not in the same drawer)

Like its N5000 predecessor, the N7000 series front of the unit has the LCD display and the standard 3 LEDs indicating system activity, status, and power. Externally the N7600 and

N7800 appear the same. The differences lay internally with increased CPU, memory, and NVRAM capability of the N7800 as compared to the N7600.



Figure 1-10 N7000 series front view

From the rear of the N7000 you can see the redundant power supplies, the NVRAM card, the Gigabit Ethernet interfaces, as well as the Fibre Channel interfaces. The console port and RLM port are also located on the rear (see Figure 1-11 on page 20).

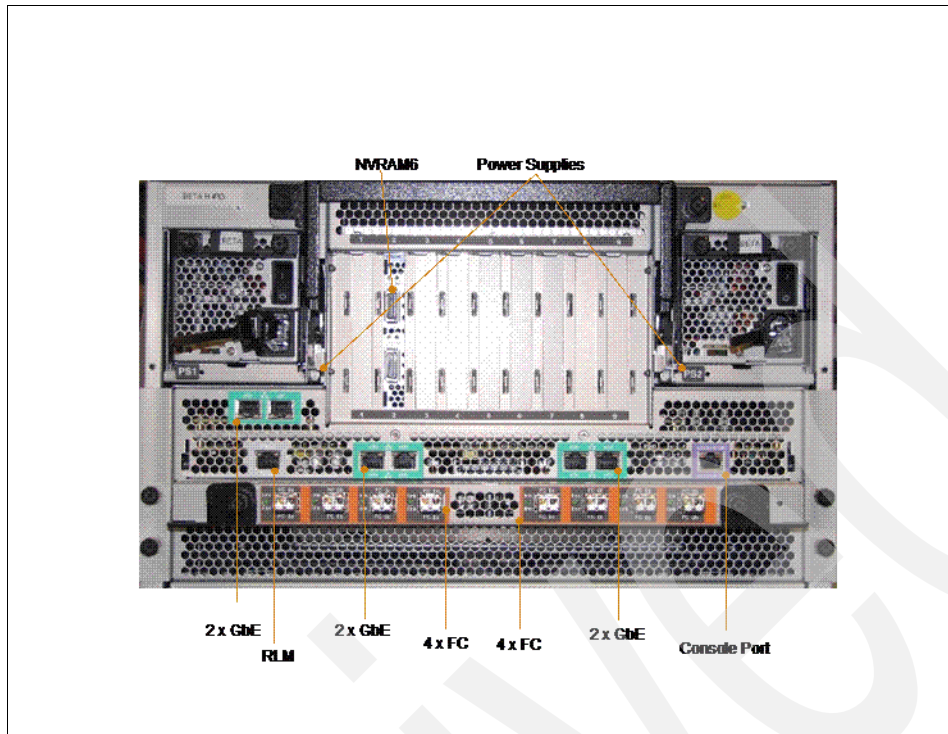


Figure 1-11 N 7000 rear view

Each N7000 node requires 6U of rack space. Each expansion unit requires 3U of rack space. Each N7000 node requires at least one expansion unit (see Figure 1-12).



Figure 1-12 N7000 rack space

A dual-node N7600 supports a maximum of 60 storage expansion units (EXN1000 and EXN2000 or EXN4000). A dual-node N7800 supports a maximum of 72 storage expansion units (EXN1000 and EXN2000 or EXN4000). See Figure 1-13. Each rack holds a maximum of twelve EXN1000s, EXN2000s, and EXN4000s. The N7000 products are IBM Customer Engineer installed, not Customer Setup.



Figure 1-13 Clustered N7000 with multiple expansion units

1.6 IBM System Storage N series gateways

The IBM System Storage N series Gateways, an evolution of the N5000 series product line, are a network-based virtualization solution that virtualizes tiered, heterogeneous storage arrays, allowing customers to leverage the dynamic virtualization capabilities available in Data ONTAP across multiple tiers of IBM and 3rd party storage (see Figure 1-14 on page 22). Like all IBM System Storage N series, the IBM N series Gateway family is based on the industry-hardened Data ONTAP microkernel operating system, which unifies block and file storage networking paradigms under a common architecture and brings a complete suite of IBM System Storage N series advanced data management capabilities for consolidating, protecting, and recovering mission-critical data for enterprise applications and users.

The industry's most comprehensive virtualization solution, the N series Gateways, provide proven and innovative data management capabilities for sharing, consolidating, protecting, and recovering mission-critical data for enterprise applications and users and seamlessly integrates into mission-critical enterprise-class SAN infrastructures. These innovative data management capabilities, when deployed with disparate storage systems, simplify heterogeneous storage management.

The N series Gateway will present shares, exports, or LUNs that are built on flexible volumes that reside on aggregates. The N series Gateway is also a host on the storage array SAN. Disks are not shipped with the N series Gateway. N series Gateways take storage array

LUNs (which are treated as disks) and virtualizes them through Data ONTAP, presenting a unified management interface.

The IBM N series Gateway offers customers new levels of performance, scalability, and a robust portfolio of proven data management software for sharing, consolidating, protecting, and recovering mission critical data. IBM System Storage N series seamlessly integrate into mission-critical SAN environments and provide a simple, elegant data management solution, decreasing management complexity, improving asset utilization, and streamlining operations to increase business agility and reduce total cost of ownership. These prospects are looking for ways to increase utilization, simplify management, improve consolidation, enhance data protection, enable rapid recovery, increase business agility, deploy heterogeneous storage services and broaden centralized storage usage by provisioning SAN capacity for business solutions requiring NAS, SAN, or IP SAN data access (see Figure 1-15 on page 23).

These prospects have the following:

- ▶ Significant investments or a desire to invest in a SAN architecture
- ▶ Excess capacity and an attractive storage cost for SAN capacity expansion
- ▶ Increasing requirements for both block (FCP, iSCSI) and file (NFS, CIFS) access
- ▶ Increasing local and remote shared file services and file access workloads

They are seeking solutions to cost effectively increase utilization, consolidate distributed storage, Direct Access Storage and file services to SAN storage, simplify storage management, and improve storage management business practices.

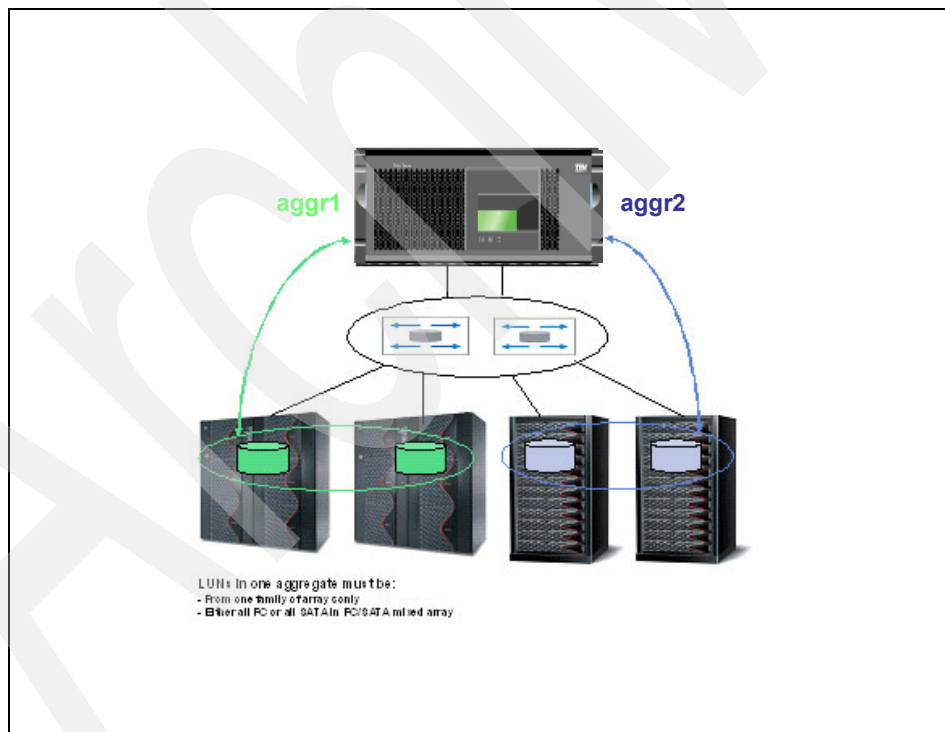


Figure 1-14 Heterogeneous storage

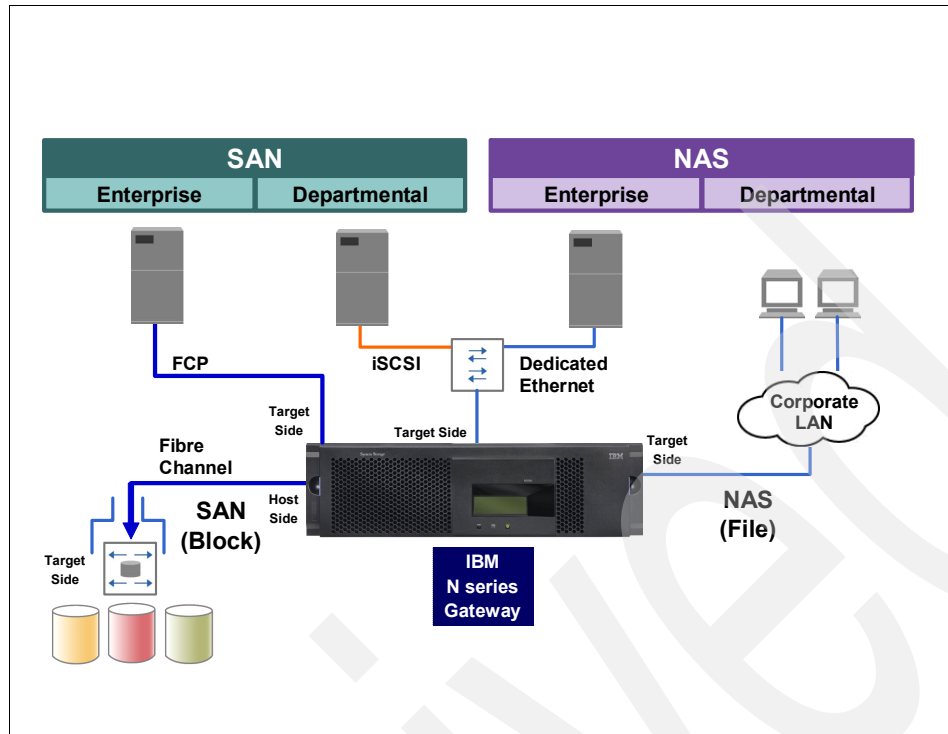


Figure 1-15 Gateway Topology

Two halves to set up

An IBM N series Gateway implementation can be thought of as a front-end implementation and a back-end implementation. The front-end setup includes configuring the N series Gateway for all protocols (NAS or FCP), implementing any snap features (Snapshot, SnapMirror, SnapVault, and so on), and setting up backup including NDMP dumps to tapes. The back-end implementation includes all tasks required to set up the IBM N series Gateways' system up to the point where it is ready for Data ONTAP installation. These include array LUN formatting, port assignment, cabling, switch zoning, assigning LUNs to the V-Series system, creating aggregates, and loading Data ONTAP.

IBM N series Gateway highlights

IBM System Storage N series Gateway provides a number of key features that enhance the value and reduce the management costs of utilizing a Storage Area Network (SAN). An IBM N series Gateway does the following:

- ▶ Simplifies storage provisioning and management.
- ▶ Lowers storage management and operating costs.
- ▶ Increases storage utilization.
- ▶ Provides comprehensive simple-to-use data protection solutions.
- ▶ Improves business practices and operational efficiency.
- ▶ Transforms conventional storage systems into a better managed storage pool. See Figure 1-16 on page 24.

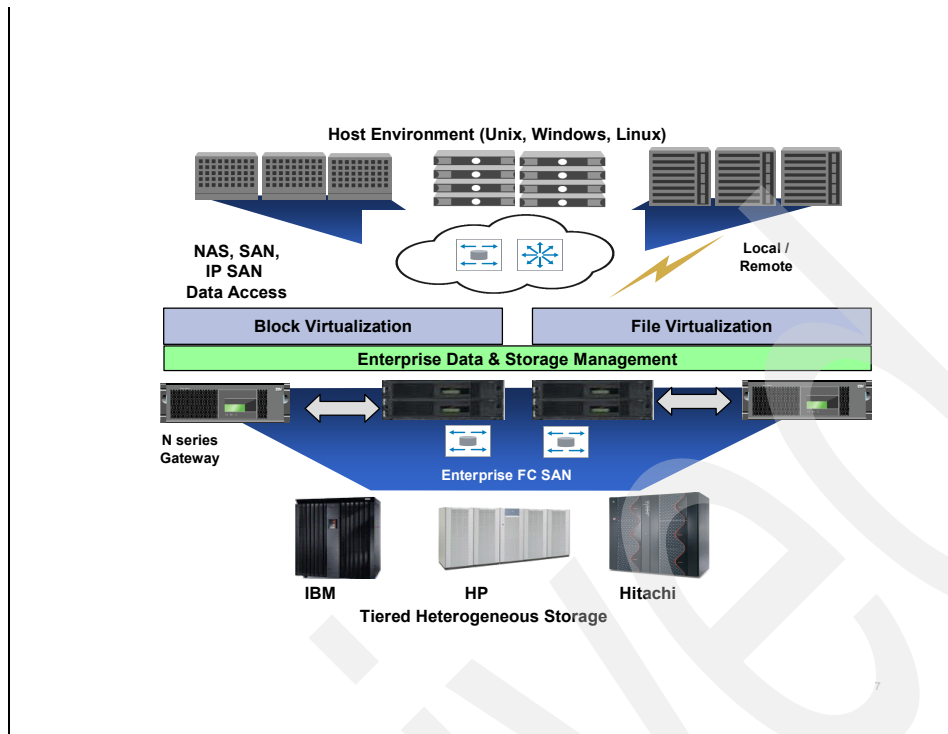


Figure 1-16 Tiered Heterogeneous Storage

Gateway RAID

Gateways use RAID0 on top of RAID1, RAID5, or RAID10 on RAID storage subsystems (see Figure 1-17). Physical disk operations such as scrubbing are disabled.

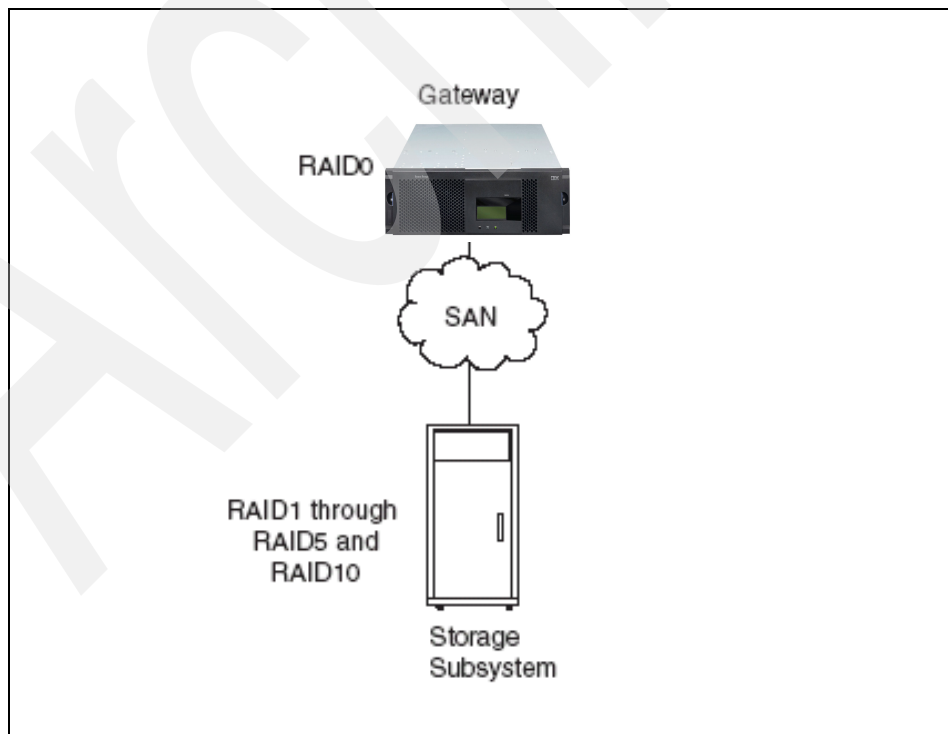


Figure 1-17 RAID Configuration

RAID0 is used to write data. See Example 1-1 for an example of volume status with the Gateway. It looks very similar to what you see on an IBM System Storage N series model except for the raid status.

Example 1-1 Vol status with gateway volumes

```
itsotuc2*> vol status -v vol3
      Volume State      Status      Options
      vol3 online      raid0, flex  nosnap=on, nosnapdir=off,
                                     minra=off,
                                     no_atime_update=off,
                                     nvfail=off,
                                     snapmirrored=off,
                                     create_ucose=on,
                                     convert_ucose=on,
                                     maxdirsize=31457,
                                     fs_size_fixed=off,
                                     guarantee=volume,
                                     svo_enable=off,
                                     svo_checksum=off,
                                     svo_allow_rman=off,
                                     svo_reject_errors=off,
                                     fractional_reserve=100,

      Containing aggregate: 'aggr0'

      Plex /aggr0/plex0: online, normal, active
      RAID group /aggr0/plex0/rg0: normal
```

1.6.1 IBM N5200, N5300, N5500, and N5600 Gateway models

The N5000 Gateway models are a good value for those wanting to extend the reach of their SANs. The N5000 Gateway incorporates a variety of reliability and availability features designed to support high demand operations. It houses hot, swappable, redundant power supplies and fans, supports multipath failover protection, and hosts dual pathing between the unit and its SAN-attached storage device. In addition, the clustering feature between two storage systems is designed to help reduce system downtime. From a hardware perspective the G10 and G20 models are identical to the A10 and A20 models of the N5200 N5300 N5500 and N5600. The differences lie in the spectrum of Data ONTAP features supported and enabled.

- ▶ N5200
 - 2864-G10
 - 2864-G20 Clustered model
- ▶ N5300
 - 2869-G10
 - 2869-G20 Clustered model
- ▶ N5500
 - 2865-G10
 - 2865-G20 Clustered model

- N5600
 - 2868-G10
 - 2868-G20 Clustered model

Table 1-8 Gateway capacity

Model	Maximum capacity
2864-G10	50TB
2864-G20	50TB per node
2869-G10	126TB
2869-G20	126TB
2865-G10	80TB
2865-G20	80 TB per node
2868	252TB
2868	252TB

Important: If you are going to enable the `cf.takeover.on_panic` option, ensure that a spare LUN is available for core dumps. If the `cf.takeover.on_panic` option is enabled and no spare LUN is available, no core dump file is produced on failure. (The `cf.takeover.on_panic` option controls whether a cluster partner immediately takes over for a panicked partner.)

Model	Maximum number of LUNs	Minimum number of LUNs
N5200 2864-G10 (non cluster model)	168	A stand-alone gateway must own at least one LUN. A cluster configuration must own at least two LUNs.
N5200 2864-G20 (cluster model)	For each node, single node N5200 2864-G10 values apply.	
N5300 2869 - G10	252	
N5300 2869 - G20	For each node, single node N5300 2869-G10 values apply.	
N5500 2865-G10 (non cluster model)	336	
N5500 2865-G20 (clustermodel)	For each node, single node N5500 2865-G10 values apply.	A stand-alone gateway must own at least one LUN.
N5600 2868-G10 (non cluster model)	504	
N5600 2868-G20 (clustermodel)	For each node, single node N5600 2868-G10 values apply.	

1.6.2 IBM Gateway models N7600 and N7800

The IBM System Storage N7000 series gateway models offer additional choice to organizations facing the challenges of enterprise data management. The IBM System Storage N7000 series is designed to deliver high-end enterprise storage and data management value with midrange affordability. Built-in enterprise serviceability and manageability features help support your efforts to increase reliability, simplify and unify storage infrastructure and maintenance, and deliver exceptional economy. The IBM N series Gateway models N7600 and N7800 deliver all the feature function that the N5000 series does but with increased processing, memory, NVRAM, and total storage capacity. The N7600 and N7800 are designed with the high end of the enterprise environments. The N7000 series Gateway hardware is identical to the A10 and A20 models. The differences in the two are the enabled features and disk attachment by Data ONTAP.

- ▶ The IBM N7000 A series comes in two Models, G10 and G20:
 - N7600
 - 2866-G10 Single Node
 - 2866-G20 Clustered
 - N7800
 - 2867-G10 Single Node
 - 2867-A20 Clustered
 - 2865-G20 Clustered model

1.7 LUN sizing for Gateways

The maximum and minimum sizes supported: Gateway support for LUN sizes is as follows:

- Maximum LUN size: 500 GB
- Minimum LUN size: 1 GB

Note: The Data ONTAP definition of a GB is as follows, one GB is equal to $1000 \times 1024 \times 1024$ bytes. Therefore, the maximum LUN size that Data ONTAP supports means $500 * 1000 * 1024 * 1024 = 524,288,000,000$ bytes.

1.8 Interoperability between Gateway G and A models

- ▶ Replication between SnapMirror on G model SnapMirror on A model. See Figure 1-18 on page 28.
 - Includes async, semisync, and synchronous

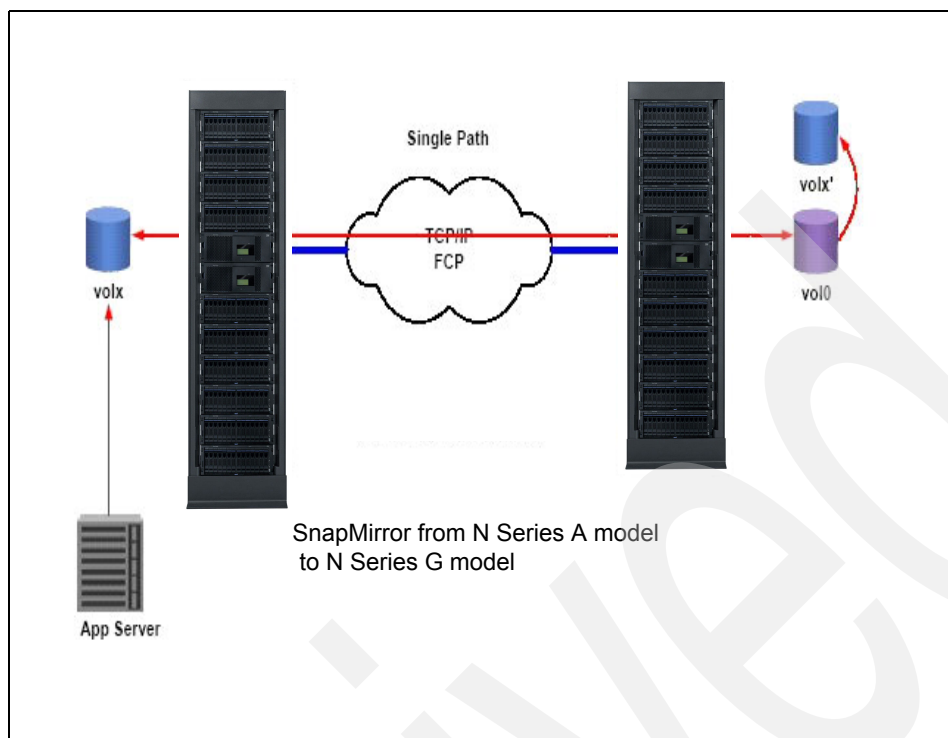


Figure 1-18 SnapMirror Interoperability

- Disk-to-disk backup from SnapVault primary on G model to SnapVault secondary on A model.



Figure 1-19 SnapVault interoperability

- Disk-to-disk backup from SnapVault primary on A model SnapVault secondary on G model

Archived

Archived

Introduction to Data ONTAP

IBM System Storage N series are driven by a robust, tightly-coupled, multi-tasking, real-time micro-kernel called Data ONTAP. This chapter describes some of the features and functionality the Data ONTAP operating system provides.

2.1 What is Data ONTAP

Data ONTAP is a robust, tightly-coupled, multi-tasking, real-time micro-kernel that minimizes complexity and improves storage system reliability. Data ONTAP has a look and feel similar to UNIX, but it is a proprietary kernel that is produced by Network Appliance Corporation.

Data ONTAP is compact—the software is less than 2% of the total size of most general purpose operating systems—thereby reducing the complexity of applying upgrades or performing routine maintenance. Following are some additional benefits of the kernel:

- ▶ No third-party application software is allowed to be installed on Data ONTAP, thereby reducing resource contention and application management overhead.
- ▶ No third-party scripts or executables are allowed to execute against the kernel, which secures the kernel from malicious viruses or poor programming effects. In fact, all external operations are performed via the access services interface of Data ONTAP.
- ▶ IBM System Storage N series side software (both standard and optionally enabled) is built directly into the Kernel.

At the lowest level, the Data ONTAP kernel is comprised of the following three basic elements (Figure 2-1 on page 32):

- ▶ A network interface driver
- ▶ A RAID manager
- ▶ The Write Anywhere File Layout (WAFL) file system

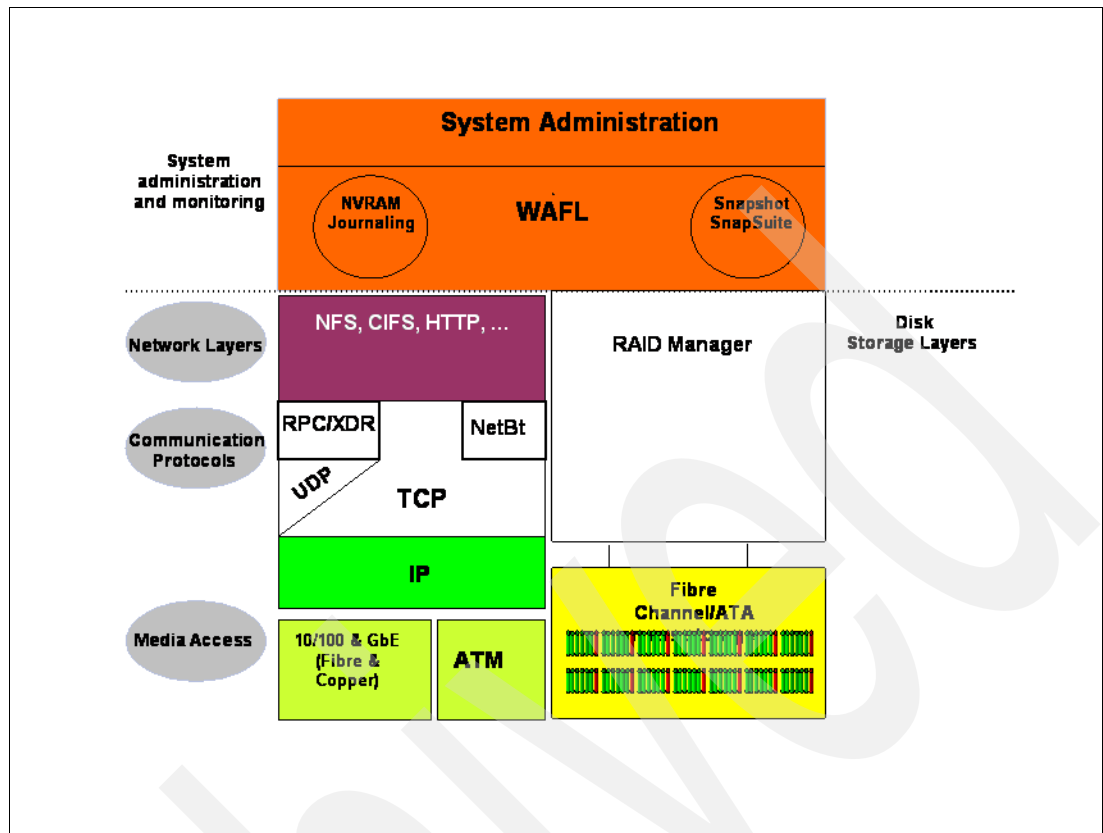


Figure 2-1 Data ONTAP

2.1.1 The network interface driver

A network interface driver within Data ONTAP is responsible for receiving all incoming NFS, CIFS, FCP, iSCSI, HTTP, and FTP requests. As each request is received, it is logged in non-volatile RAM (NVRAM), an acknowledgement is immediately sent back to the requestor, and any processing needed to satisfy the request is initiated. After initiated, this processing runs uninterrupted (and continuously), as long as possible. This approach differs from that of traditional file servers, which employ separate processes for handling the network protocol stack, the remote file system semantics, the local file system, and the disk subsystem.

2.1.2 The RAID manager

The Redundant Array of Inexpensive Disks (RAID) technology is designed to protect against loss of data in the event that disk failure occurs. Although RAID technology can be implemented in five different levels (each of which have their own advantages and disadvantages), levels 1, 3, and 5 are the most common forms used. In order to understand how the IBM System Storage N series implements RAID 4 technology, it helps to understand what levels of RAID are available and how RAID actually works.

RAID levels

RAID level 1 is simply disk mirroring. When RAID 1 is used, all data is duplicated on two separate disks. RAID 1 is very safe, but its usage doubles the amount of disk space normally required.

RAID 3 uses a single parity disk and one or more data disks. Data is written to the data disks in stripes, but the stripe size used is so small that each individual read or write operation performed must access all the disks in the RAID array. For instance, the first byte in a block of data might be on the first disk, the second byte on the second disk, and so on. RAID 3 (Figure 2-2) is a good fit for applications that require very high data rates for single large files, such as super computing and graphics processing. However, it performs poorly when used with multi-user applications that require many unrelated disk operations to be performed in parallel because every operation performed causes traffic to be generated on each disk in the array.

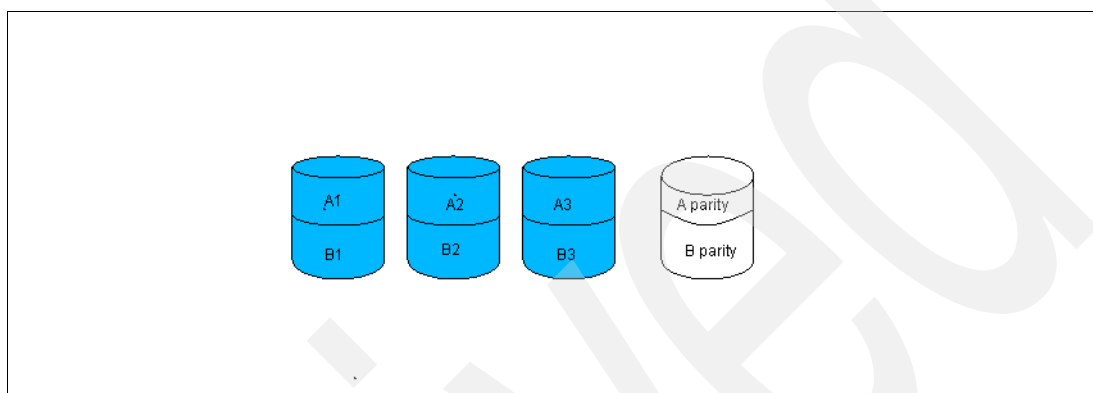


Figure 2-2 RAID3

Similar to RAID 3, RAID 4 uses a single parity disk and one or more data disks. However, with RAID 4, data is written to the data disks in much larger stripes. As a result, each data disk in a RAID 4 array can usually satisfy a separate user request, allowing more requests to be processed simultaneously.

RAID 5 is similar to RAID 4, but instead of keeping parity blocks on a separate parity disk, it cycles parity blocks among all of the disks in the RAID array (parity for the first stripe is stored on the first disk, parity for the second stripe is stored on the second disk, and so on). The primary advantage of RAID 5 is that it eliminates the need for a single parity drive, which can become a bottleneck if large amounts of data are written to the RAID group in parallel. The major disadvantage with RAID 5 is that it is not practical to add a single disk to a disk array after the array is created. Instead, if the size of an existing RAID 5 array needs to be increased, the number of disks to be added must match the current size of the array. For example, if a RAID 5 implementation uses seven disks in each array, then disks must be added to the array seven at a time.

IBM System Storage N series RAID implementation

The IBM System Storage N series uses RAID 4 technology; however, unlike most RAID 4 and RAID 5 implementations, which are architected without thought to file system structure or activity, the IBM N series' RAID 4 implementation is heavily optimized to work in tandem with the WAFL file system. By optimizing the file system and the RAID layer together, the IBM N series RAID design provides all the benefits of RAID 4 protection, without incurring the performance disadvantages that are often associated with general-purpose RAID 4 solutions. And because IBM N series' RAID 4 design does not interleave parity information like generic implementations of RAID 5, the overall system can be expanded quickly and easily, even though RAID protection is present. The disk layout for the IBM N series' RAID 4 technology is illustrated in Figure 2-3 on page 34.

How the parity disk is used

With RAID 3 and RAID 4, if one block on a disk goes bad, the parity disk within that disk's RAID group is used to recreate the data in that block, and the block is mapped to a new location on disk. A similar approach is used if an entire data disk fails. After the failed disk is replaced, the parity disk is used in conjunction with the remaining data disks to recalculate its contents, thereby preventing data from being lost.

A single IBM System Storage N series RAID 4 array is comprised of one disk that is used for parity and up to 27 disks that are used for storing data. Each disk in the array (referred to as a *RAID group*) is made up of 4 KB blocks; therefore, each stripe consists of one block from each data disk and one block from the parity disk. The parity block in each stripe allows data to be recalculated if any one block (on a data disk) in the stripe is lost.

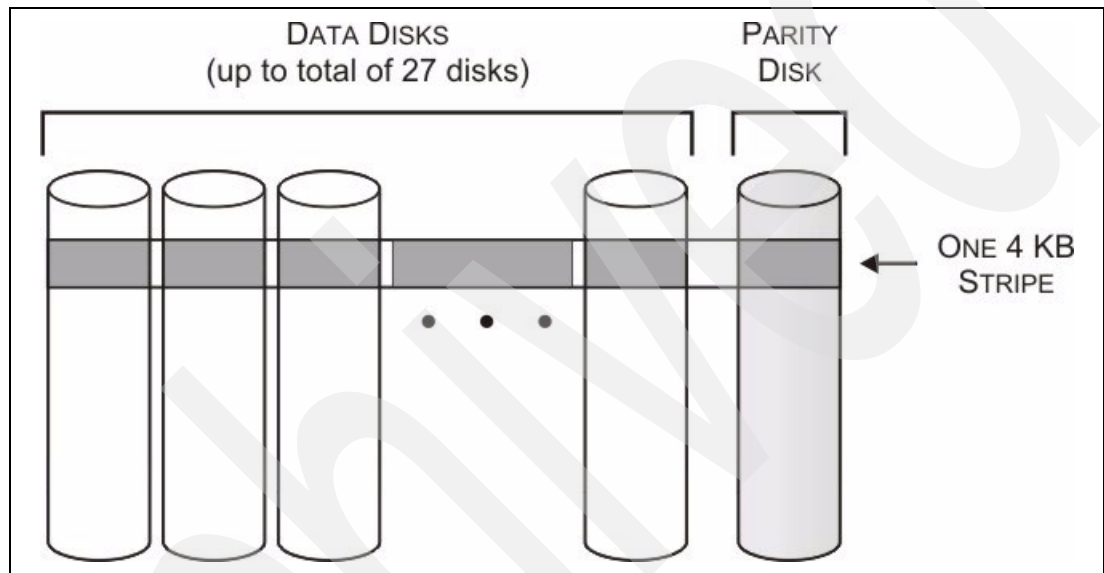


Figure 2-3 IBM N series RAID 4 disk layout

To understand how the parity disk works, it helps to think of each 4 KB block on a disk as though it were a really big integer (32,768 bits long), and that RAID 4 is responsible for performing simple math operations using these integers. With this in mind, the parity block can be thought of as being the big integer that is the sum of all blocks in the stripe. A simple example is in Figure 2-4 on page 35.

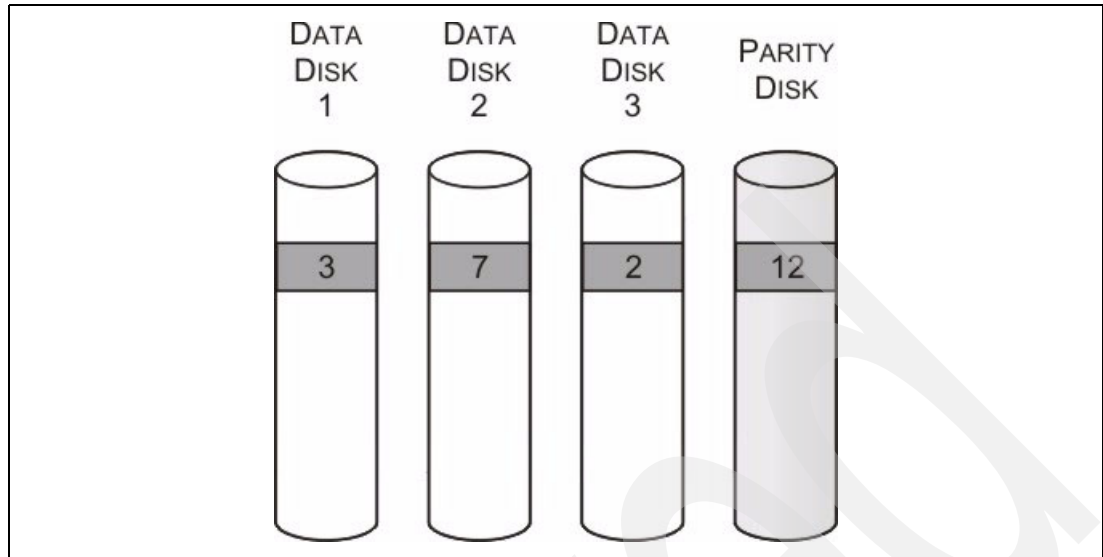


Figure 2-4 Example of how the parity disk is used

In the example in Figure 2-4, if one of the data disks in the RAID group fails, for instance “Data 2”, the data stored on that disk can be reconstructed again by performing a simple arithmetic operation similar to the following:

$$\begin{aligned}\text{Data 2} &= \text{Parity} - \text{Data 1} - \text{Data 3} \\ &= 12 - 3 - 2 \\ &= 7\end{aligned}$$

In reality, the RAID system uses EXCLUSIVE-OR instead of addition and subtraction, and the numbers used are much larger, but the end result is the same. Using addition and subtraction on small numbers makes the process easier to understand.

Lost data is recalculated on the fly, so the IBM System Storage N series will continue to run, even if one of the data disks in a RAID group fails—The entire contents of a failed disk are recalculated and written to a new disk immediately after the failed disk is replaced—On the other hand, if the parity disk itself fails, it must be replaced (after which parity values are recalculated and written to it), but no data stored on data disks within the RAID group are lost.

Eliminating the parity disk bottleneck

Most RAID storage system vendors avoid using RAID 4 technology because with general-purpose file systems the parity disk can become a bottleneck. To better understand how the parity disk can become a bottleneck, it helps to examine how most general-purpose RAID 4 file systems work. The Berkeley Fast File System (FFS) is a RAID 4 file system that was designed to optimize write operations for individual files. Because of its design, FFS typically writes blocks for different files to widely separated locations on disk. The illustration shown in Figure 2-5 on page 36 shows how FFS might allocate blocks for three unrelated files in a RAID 4 array. While each data disk used might receive only two write operations, the parity disk will receive six (three times as many). More importantly, the parity disk writes are widely spread, causing time consuming seek operations to be performed. And because the FFS file system is not aware of the underlying RAID 4 disk layout, when read operations are performed, FFS tends to generate requests for data that are scattered throughout the data disks, which causes the parity disk to seek excessively.

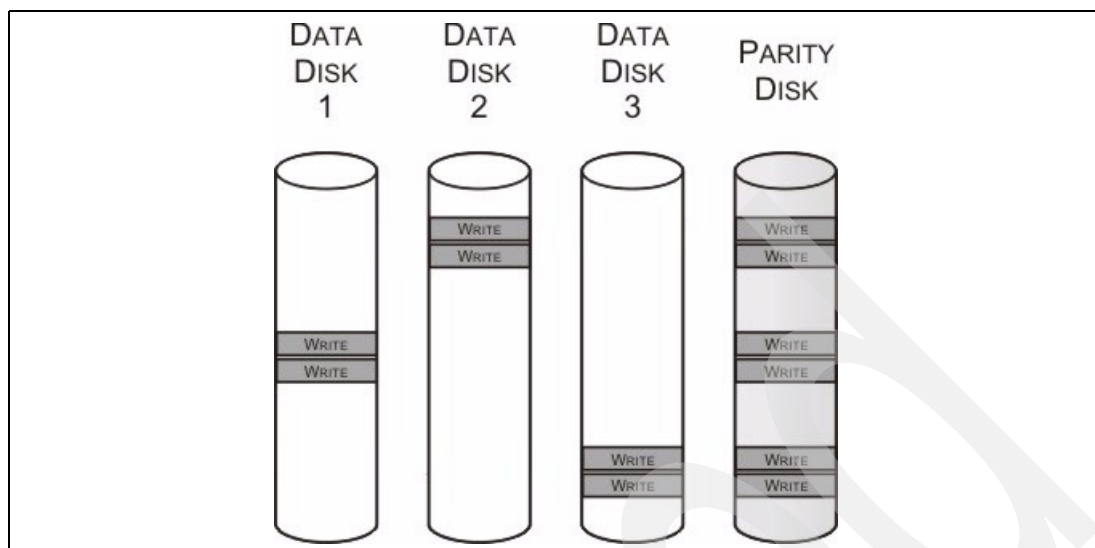


Figure 2-5 Example of FFS disk write patterns

In contrast, Data ONTAP's WAFL file system writes blocks in a pattern that is designed to minimize seek operations on the parity disk. The illustration shown in Figure 2-6 shows how WAFL would allocate the same blocks to make RAID 4 operate efficiently. Here, WAFL only used three stripes, so only three parity blocks were updated, and they are all located near each other. WAFL always writes blocks to stripes that are in close proximity of each other, eliminating the need for long seeks on the parity disk. WAFL also writes multiple blocks to the same stripe whenever possible, further reducing operations on the parity disk.

RAID reconstruction

Whenever a disk drive in the disk subsystem of an IBM System Storage N series fails, the system is placed in what is referred to as degraded mode, and requests for data from the failed disk are served by reconstructing the data "on the fly" with no interruption in file service. A new disk drive can be substituted for the failed one at any time, and the data stored on the failed disk is automatically rebuilt on the replacement disk (using information stored on the parity disk) without any interruption in file service.

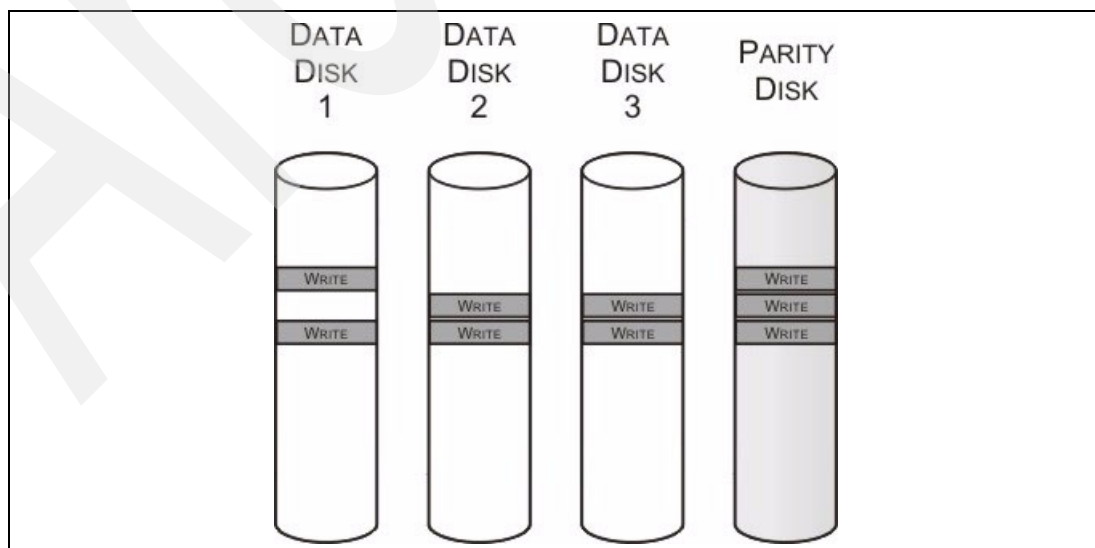


Figure 2-6 Example of WAFL disk write patterns

An IBM System Storage N series can be configured such that one or more “hot spare” disk drives are available. When that is the case, a hot spare is immediately substituted for a failed disk whenever the N series product is in degraded mode. The availability of multiple hot spares allows for the automatic replacement of subsequent failed disks.

The reconstruction process can be assigned a low priority (and take a long time to complete) if incoming service requests need to be handled quickly as a disk is being rebuilt. Or it can be assigned a high priority (and be completed quickly) if sluggish service request performance for a brief period of time is acceptable.

RAID scrubbing

See Figure 2-7 for an example. Far more likely than the possibility of a disk drive failing in a RAID group, is the possibility that there may be an unknown bad block (media error) on an otherwise intact disk. If there are no failed disks within a RAID group, the storage system compensates for bad blocks by using parity information to recompute the bad block's original contents, which is then remapped to a “spare” block elsewhere on the disk when data in that block is accessed. However, if a bad block is encountered while the IBM N series is in degraded mode (after a disk failure but before reconstruction has completed), then that block's data can be irrecoverably lost.

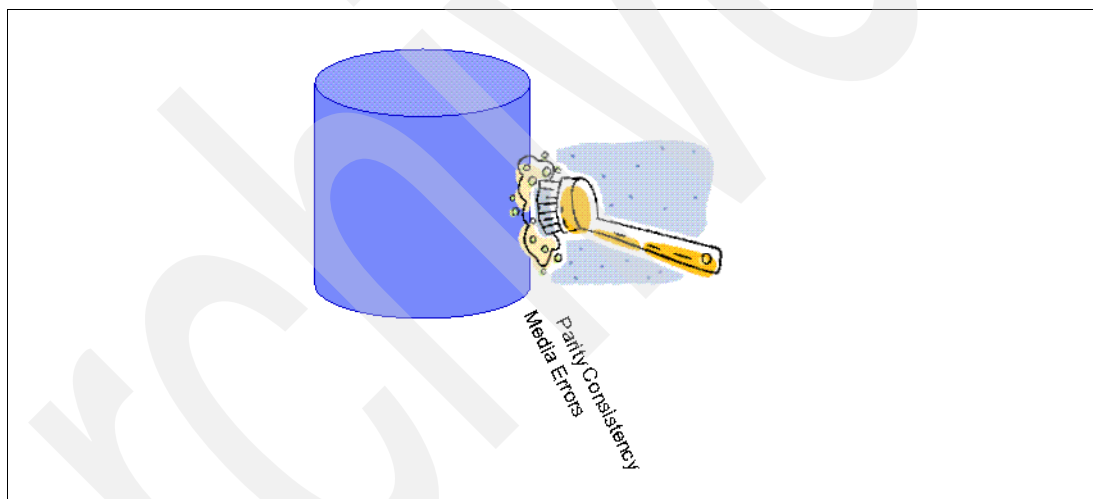


Figure 2-7 Disk Scrubbing

To protect against this scenario, IBM System Storage N series routinely verify all data stored in the system using a process known as *RAID scrubbing*. By default, this process is performed once per week, early on Sunday morning, although it can be rescheduled or suppressed altogether. During the RAID scrubbing process, all data blocks are read from RAID groups that contain no failed drives and if a media error is encountered, the bad block's data value is recomputed and the data value itself is rewritten to a spare block. Otherwise, parity is recomputed and verified and if the computed parity value does not match the corresponding parity value stored on disk, the parity value on disk is rewritten. It is important to note that all non-degraded RAID groups are scrubbed in parallel.

IBM System Storage N series RAID-DP

Modern disk architecture continues to evolve, and as a result disk drives are orders of magnitude larger than they were when RAID was first introduced. Unfortunately, as disk drives have gotten larger, reliability has not improved. More importantly, the likelihood that an uncorrectable bit error will occur is now much higher, since bit error rates have stayed the

same. These three factors, larger disks, unimproved reliability, and increased bit errors with larger media, all have serious consequences for the ability of single-parity RAID to protect data.

In response to these issues, Data ONTAP uses a new type of RAID protection called RAID-DP. RAID-DP stands for *RAID Double Parity*, and it significantly increases the fault tolerance from failed disk drives over traditional RAID implementations. In fact, when all relevant numbers are plugged into the standard mean time to data loss (MTTDL) formula for RAID-DP versus single-parity RAID, RAID-DP is on the order of 10,000 times more reliable on the same underlying disk drive array. At the lowest level, RAID-DP offers protection against the following:

- ▶ Double disk failure within the same RAID group
- ▶ Single disk failure, followed by a bad block/bit error during reconstruction.

How RAID-DP works

At the most basic level, RAID-DP adds a second parity disk to each RAID 4 disk array used. A traditional RAID 4 array is comprised of some number of data disks and one parity disk. A RAID-DP array consists of some number of data disks and two parity disks. Furthermore, where the parity disk in a RAID 4 disk array contains row parity values that are calculated horizontally across all disks in the array, the second parity disk in a RAID-DP array contains parity values that are calculated diagonally. Figure 2-8 shows how diagonal parity values are calculated.

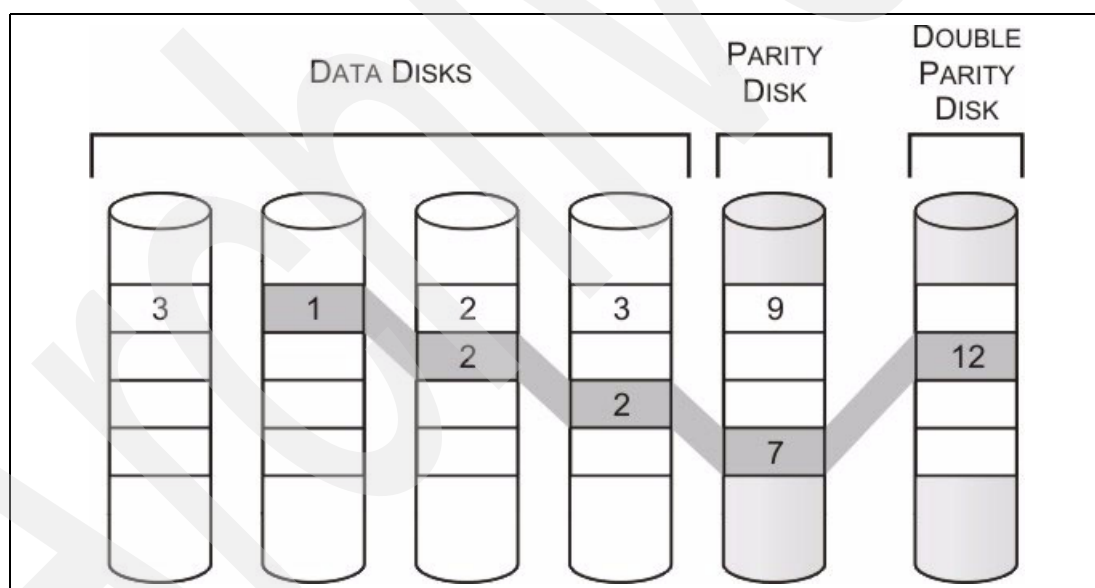


Figure 2-8 Diagonal parity used with RAID-DP

In reality, both horizontal and diagonal parity values are generated using EXCLUSIVE-OR operations instead of addition and subtraction, and the numbers used are much larger. But the end result is the same. One of the most important items to note is that the diagonal parity stripe includes an element from the row parity disk as part of its diagonal parity sum. RAID-DP treats all disks in the original RAID 4 construct (data and row parity disks) the same.

Figure 2-9 on page 39 shows a RAID-DP scenario with data for each block provided, along with corresponding row and diagonal parity stripes.

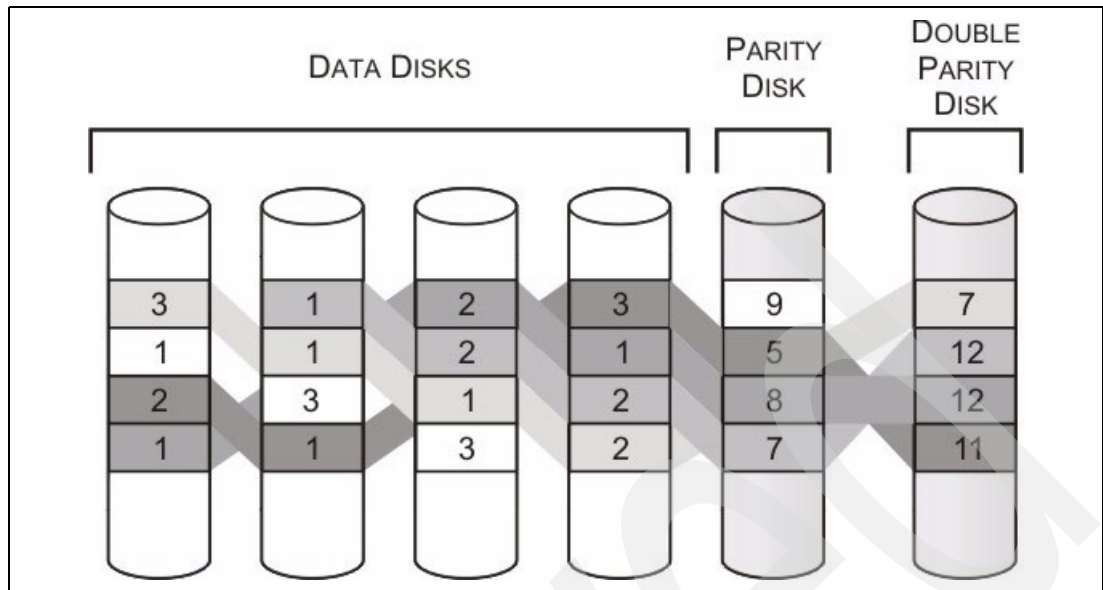


Figure 2-9 A RAID-DP scenario with data and parity values

One condition that is apparent from the example shown in Figure 2-9 is that the diagonal stripes wrap at the edges of the row parity construct. Two important conditions for RAID-DP's ability to recover from double disk failures may not be readily apparent in this example. The first is that each diagonal parity stripe misses one and only one disk, and each diagonal stripe misses a different disk. This results in the second condition—that there is one diagonal stripe that does not get parity data generated for it—nor is a corresponding parity value stored on the second diagonal parity disk. In this example the omitted diagonal stripe is the white non-patterned blocks.

Proving that RAID-DP really does recover all data in the event of a double disk failure can be done in two manners. One is using mathematical theorems and proofs, and the other is to simply go through a double disk failure and subsequent recovery process.

It is important to note that the same RAID-DP diagonal parity conditions covered in this example hold true in real storage deployments that involve dozens of disks in a RAID group and millions of rows of data written horizontally across the RAID 4 group. And, while it is easier to illustrate RAID-DP with the smaller example in Figure 2-9, recovery of larger sized RAID groups work exactly the same regardless of the number of disks in the RAID group.

It is important to note that if a double disk failure occurs, RAID-DP automatically raises the priority of the reconstruction process so that the recovery completes faster. As a result, the time to reconstruct data from two failed disks is slightly less than the time to reconstruct data from a single disk failure. A second key feature of RAID-DP with double disk failure is that it is highly likely one disk failed some time before the second, and at least some information was already recreated with traditional row parity. RAID-DP automatically adjusts for this occurrence by starting recovery where two elements are missing from the second disk failure.

2.1.3 The Write Anywhere File Layout (WAFL) file system

WAFL is a UNIX compatible file system that is optimized for network file access. In many ways, WAFL is similar to other UNIX file systems, such as the Berkeley Fast File System (FFS) and the TransArc Episode file system:

- ▶ WAFL is block based (4 KB blocks, no fragments)

- ▶ WAFL uses *inodes* to describe its files
- ▶ WAFL treats directories as specially formatted files

Like FFS, each WAFL inode contains pointers to indicate which blocks belong to a file. Unlike FFS, all the block pointers in a WAFL inode refer to blocks at the same level. Thus, inodes for small files use the block pointers within the inode to point to data blocks. Inodes for larger files point to indirect blocks that point to actual file data. Inodes for larger files point to doubly indirect blocks and so forth. For very small files, data is stored *directly* in the inode itself in place of block pointers. By storing data and blocks this way, filesystem I/O can be reduced since the access of the inode brings in the first layer of block pointers and in some cases, actual file content.

Another unique feature of WAFL is in its ability to store sufficient meta-data, allowing it to function with any of the current mainstream operating systems (Unix, Linux, and Windows), using a number of communications protocols (NFS, CIFS, HTTP, and so forth.), as well as inter operate with block-level protocols like FCP and iSCSI. It also contains unique optimizations that effectively increase its ability to move data between disk blocks and network interfaces.

Meta-data lives in files

Like the TransArc Episode file system, WAFL uses a set of special-purpose files to store meta-data. WAFL's three most important meta-data files are as follows:

- ▶ *Inode file*, which contains all inodes for the file system
- ▶ *Free block-map file*, which identifies all free blocks available
- ▶ *Free inode-map file*, which identifies all free inodes available.

The terms block-map and inode-map are used instead of bit-map and inode-map because these files require more than one bit for each entry. Figure 2-10 shows the layout of these three meta-data files.

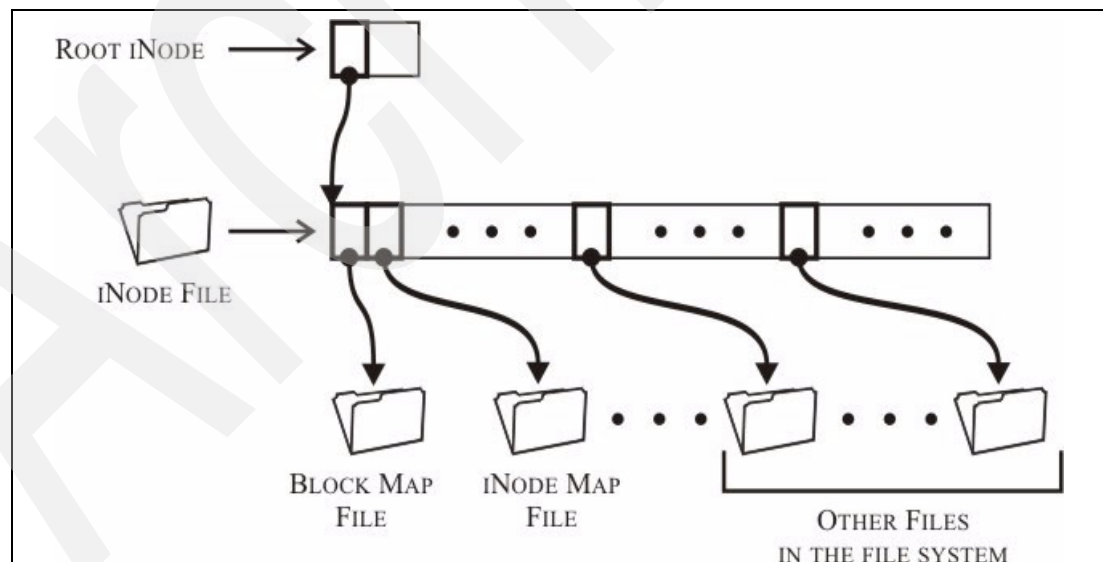


Figure 2-10 Layout used by the WAFL file system

Because WAFL meta-data lives in actual files (albeit completely hidden files) this information is treated as any other data, for example, its files receive the same protections and accelerations provided for user data files.

By keeping meta-data in files, meta-data blocks can be written anywhere on disk in fact, that is where the name WAFL, which stands for *Write Anywhere File Layout*, came from. WAFL has complete flexibility in its write allocation policies because no blocks are permanently assigned to fixed disk locations (as they are in FFS). This write-anywhere design allows WAFL to operate efficiently with the RAID disk subsystem by scheduling multiple writes to the same RAID stripe whenever possible to avoid the 4-to-1 write penalty that RAID traditionally incurs when just one block in a stripe is updated.

Keeping meta-data in files also makes it easy to increase the size of the file system on the fly. When a new disk is added, the IBM System Storage N series automatically increases the sizes of the meta-data files. The system administrator can increase the number of inodes in the file system manually as well if the default is too small but this is generally unnecessary. Finally, the write-anywhere design enables the copy-on-write technique that is used by Snapshots. In order for Snapshots to work, WAFL must be able to write all new data, including meta-data, to new locations on disk; instead, of overwriting existing data with new data values. If WAFL stored meta-data at fixed locations on disk, this would not be possible.

A tree of blocks

A WAFL file system is essentially a tree of blocks. At the root of the tree is the root inode, as shown in Figure 2-10 on page 40. The root inode is a special inode that describes the inode file. The branches of the tree consists of the inodes that describe the rest of the files in the file system, including the block-map and inode-map files. The leaves of the tree are the actual data blocks of all the files stored in the system. Figure 2-11 shows a detailed view of this tree of blocks.

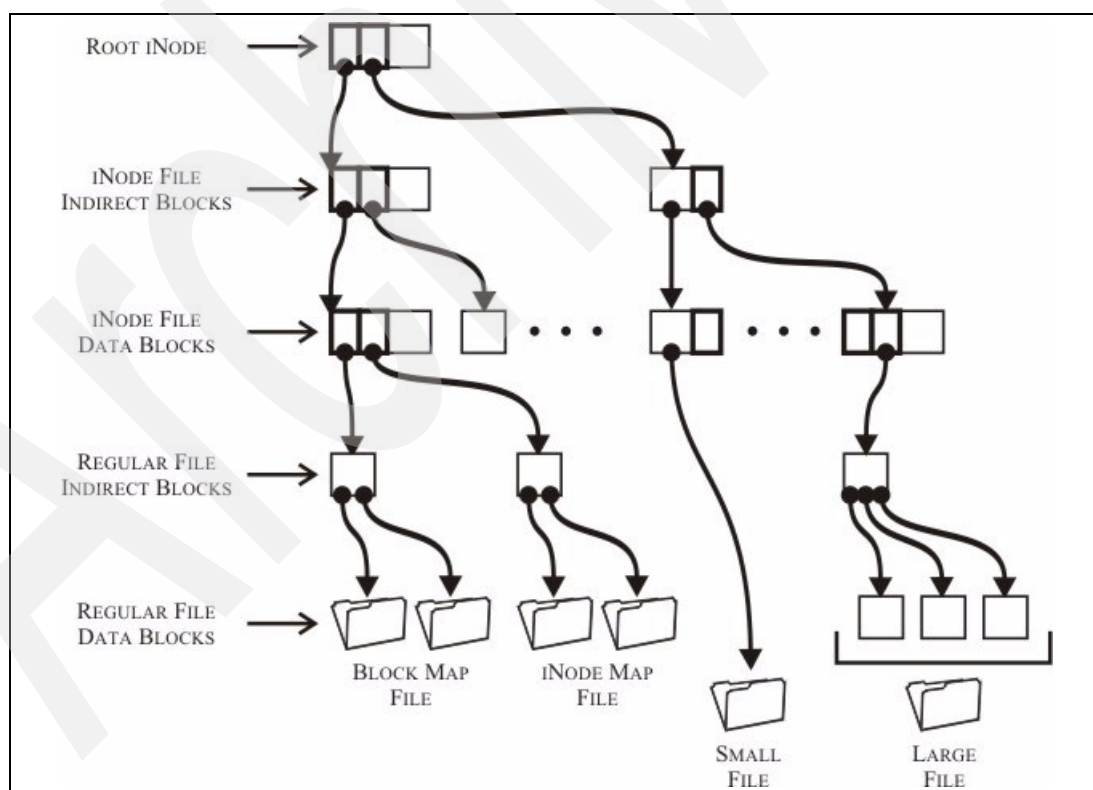


Figure 2-11 A detailed view of WAFL's tree of blocks

As you can see in Figure 2-11, files are made up of individual blocks and large files have additional layers of indirection between the inode and the blocks that contain the actual file data. In order for WAFL to boot, it must be able to find the root of this tree, so the one

exception to WAFL's write-anywhere rule is that the block containing the root inode must live at a fixed location on disk where WAFL can find it.

A word about write allocation

Write performance is especially important for network file servers. Observations show that as read caches get larger at both the client and server, writes begin to dominate the I/O subsystem. This effect is especially pronounced with NFS, which allows very little client-side write caching. The result is that the disks on an NFS server may have five times as many write operations as reads.

The WAFL design was motivated largely by a desire to maximize the flexibility of its write allocation policies. This flexibility takes the following three forms:

1. WAFL can write any file system block (except the one containing the root inode) to any location on disk. In FFS, meta-data, such as inodes and bit maps, is kept in fixed locations on disk. This prevents FFS from optimizing writes by, for example, storing both the data for a newly updated file and its corresponding inode in close proximity to each other on disk. Since WAFL can write meta-data anywhere on disk, it can optimize writes more creatively.
2. WAFL can write blocks to disk in any order. FFS writes blocks to disk in a carefully determined order so that *fsck* (a UNIX file system repair utility) can restore file system consistency after an unclean shutdown occurs. WAFL can write blocks in any order because the on-disk image of the file system changes only when WAFL writes a consistency point. The one constraint is that WAFL must write all the blocks in a new consistency point before it writes the root inode for the consistency point.
3. WAFL can allocate disk space for many NFS operations at once in a single write episode. FFS allocates disk space as it processes each NFS request. WAFL gathers up hundreds of NFS requests before scheduling a consistency point, at which time it allocates blocks for all requests in the consistency point at once. Deferring write allocation improves the latency of NFS operations by removing disk allocation from the processing path of the reply, and it avoids wasting time allocating space for blocks that are removed before they reach disk.

Together, these three features give WAFL extraordinary flexibility in its write allocation policies. The ability to schedule writes for many requests at once enables more intelligent allocation policies, and the fact that blocks can be written to any location and in any order allows a wide variety of strategies. In short, the following applies:

- ▶ WAFL improves RAID performance by writing multiple stripes to disk using the same number of I/O operations that other RAID implementations use to write a single stripe (Figure 2-12 on page 43).
- ▶ WAFL reduces seek time by writing blocks to locations that are near each other on disk.
- ▶ WAFL reduces head-contention when reading large files by placing sequential blocks for a file on a single disk in the RAID array (rather than across multiple disks) whenever possible.

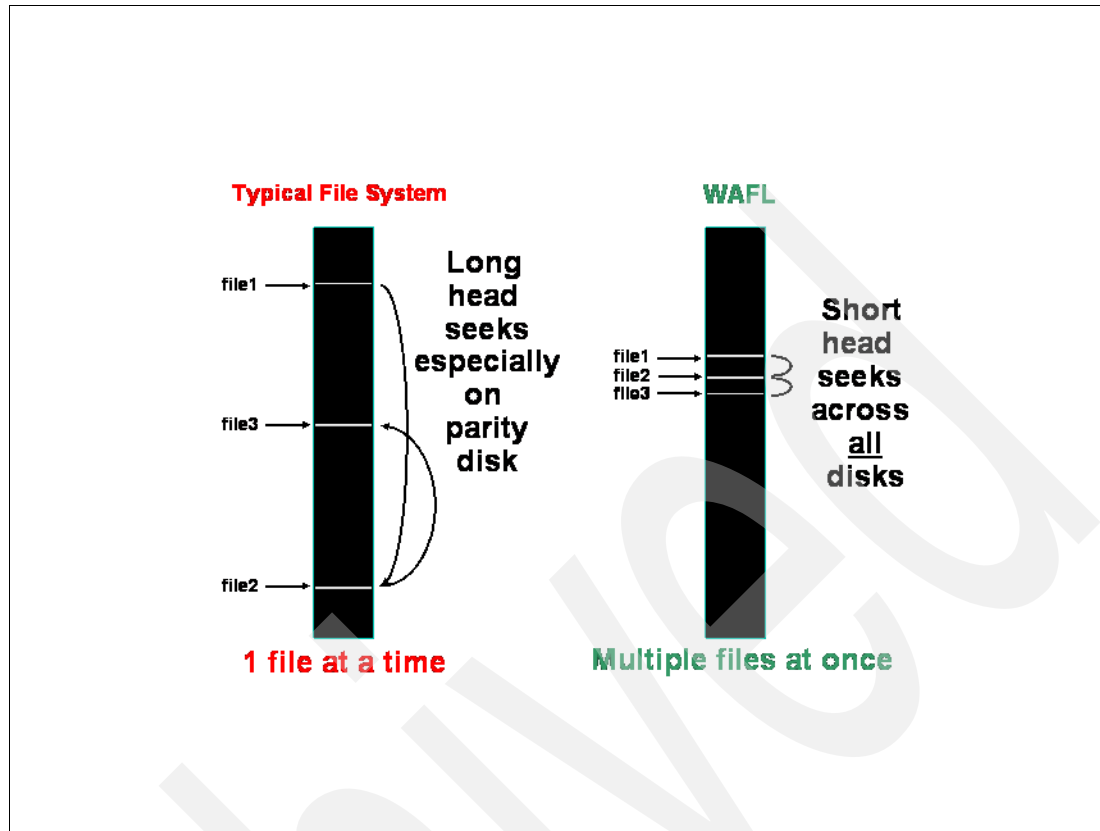


Figure 2-12 WAFL data placement

2.1.4 File system consistency and Non-Volatile RAM

In most storage systems, RAID management and file services can put a heavy strain on write performance. RAID uses a time-consuming read-modify-write sequence to maintain parity, while file services tries to ensure that data is safely stored before replying to network requests. WAFL reduces the impact of RAID management by writing to disk locations that minimize RAID's write performance penalty.

Non-volatile RAM or NVRAM is special memory that is equipped with battery-backup that allows it to store data for days, even when system power is off. NVRAM reduces response time by caching data and preventing NFS/CIFS requests from being dropped because of delays in receiving "packet received" acknowledgements.

During normal operation, WAFL automatically creates a special copy of the root inode (copies of the root inode are known as *Snapshots*) every 10 seconds, thereby creating a consistency point within an IBM N series system. (Consistency points are created more frequently if the I/O load is heavy.) WAFL then uses NVRAM to keep a running log of NFS update requests it processed since the last consistency point was taken. WAFL actually divides the NVRAM into two separate log areas of equal size¹. When one log area gets full, WAFL switches to the other log area, establishes a new consistency point, and writes the changes stored in the first log area to disk. WAFL attempts to create a consistency point every 10 seconds, even if the log is not full, to prevent the on-disk image of the file system from getting too far out of date.

¹ If the N-series is clustered, the NVRAM is actually divided in half where one half is used by the local node and the second half is a mirror of the NVRAM on the partner. Each of these halves are divided again in half to create the two halves used for consistency point processing by WAFL.

During a normal system shutdown, the N series product turns off file and block services, flushes all cached operations to disk, writes one last consistency point, and turns off NVRAM to increase battery life. Thus, on a clean shutdown the NVRAM does not contain any unprocessed I/O requests. When an unexpected shutdown occurs, the NVRAM may contain unprocessed I/O requests so it remains on (and running from battery power if the shutdown was caused by a loss of power). Later, when the IBM N series is restarted, it reverts to the last consistency point established, looks in NVRAM for any uncompleted requests, and replays all requests found that are not yet externalized to disk. This allows the N System to reboot in a short amount of time regardless of how much storage was attached and active at the time the outage occurred—when an IBM N series system boots, there is no need to perform time consuming file system checks.

Logging only I/O requests that will modify the filesystem has several advantages over the traditional technique of using NVRAM (Figure 2-13) to cache writes at the disk driver layer. In particular, it is a much more efficient use of space—it is very common for a simple I/O request to necessitate the updating of many blocks on disk. Modified data blocks in turn require the update of parity information. Logging in this manner also prevents the N series product from being locked into performing I/O on specific disk blocks. It actually provides the opportunity to combine a number of update requests into a single string of disk operations (referred to as coalescing).

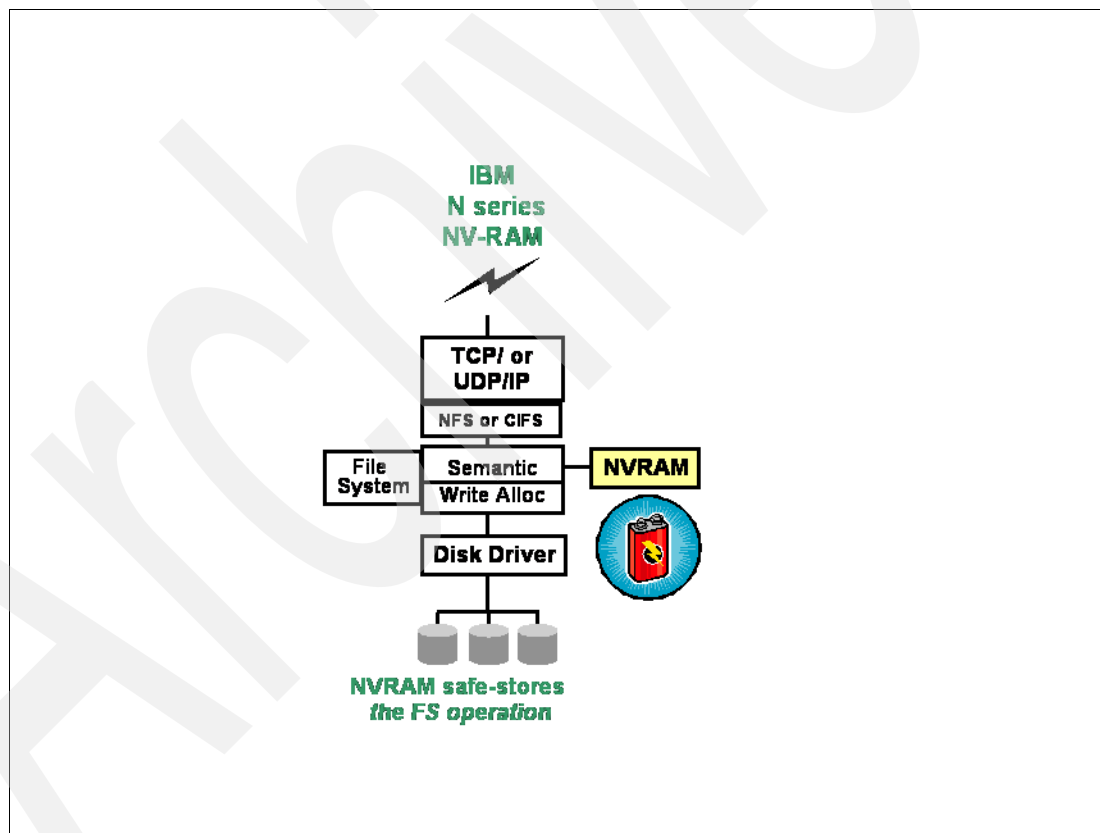


Figure 2-13 NVRAM

High performance processing

Processing an NFS or CIFS request and caching the resulting disk writes generally takes much more NVRAM than simply logging the information required to replay the request. For example, to move a file from one directory to another using NFS, the file system must update the contents and inodes of both the source and target directories. WAFL uses about 150 bytes to log the information needed to replay a rename operation. Rename, with its factor of a

200 difference in NVRAM usage, is an extreme case, but even for a simple 8 KB write, caching disk blocks will consume 8 KB for the data, 8 KB for the inode update, and for large files another 8 KB for the indirect block. WAFL logs just the 8 KB of data along with about 120 bytes of header information. With a typical mix of NFS operations, WAFL can store more than 1000 operations per megabyte of NVRAM.

Using NVRAM as a cache of unwritten disk blocks turns it into an integral part of the disk subsystem. A failure in traditional NVRAM can corrupt the file system in ways that *fsck* (a UNIX file system repair utility) cannot detect or repair. If something goes wrong with WAFL's NVRAM, WAFL may lose a few I/O requests, but the on-disk image of the file system remains intact. This is important because NVRAM is reliable, but not as reliable as a RAID disk array.

A final advantage of logging NAS requests is that it improves response times. To reply to a request, a file system without any NVRAM must update its in-memory data structures, allocate disk space for new data, and wait for all modified data to reach disk. A file system with an NVRAM write cache follows the same steps with one exception, it copies modified data into NVRAM instead of waiting for the data to reach disk. WAFL can reply to an NFS request much more quickly because it need only update its in-memory data structures and log the request. It does not allocate disk space for new data or copy modified data to NVRAM.

2.2 Snapshots and SnapRestore

Snapshots and SnapRestore are two of the main benefits of the IBM System Storage N series. The following sections discusses both features.

2.2.1 IBM System Storage N series Snapshots

One of the benefits the WAFL file system provides is the ability to make read-only copies of the way its entire file system looks at any given point in time, as well as to make those copies available to system administrators via “special” subdirectories that appear in the current (active) file system. Each read-only copy of the file system is called a *Snapshot*, and an IBM System Storage N series can keep up to 255 Snapshots online at once to provide quick and easy access to old versions of files.

Understanding that the WAFL file system is a tree of blocks whose beginning is the root inode is the key to understanding Snapshots. Data ONTAP creates a new Snapshot by making a duplicate copy of the root inode. This duplicate copy then becomes the root of a tree of blocks that represents the state of the file system at the point-in-time the Snapshot was taken, just as the root inode represents the active file system. When a Snapshot inode is created, it points to the exact same disk blocks as the root inode. Thus, a new Snapshot consumes just enough disk space to store the root inode copy. Figure 2-14 on page 46 illustrates how Data ONTAP creates a new Snapshot by making a duplicate copy of the root inode.

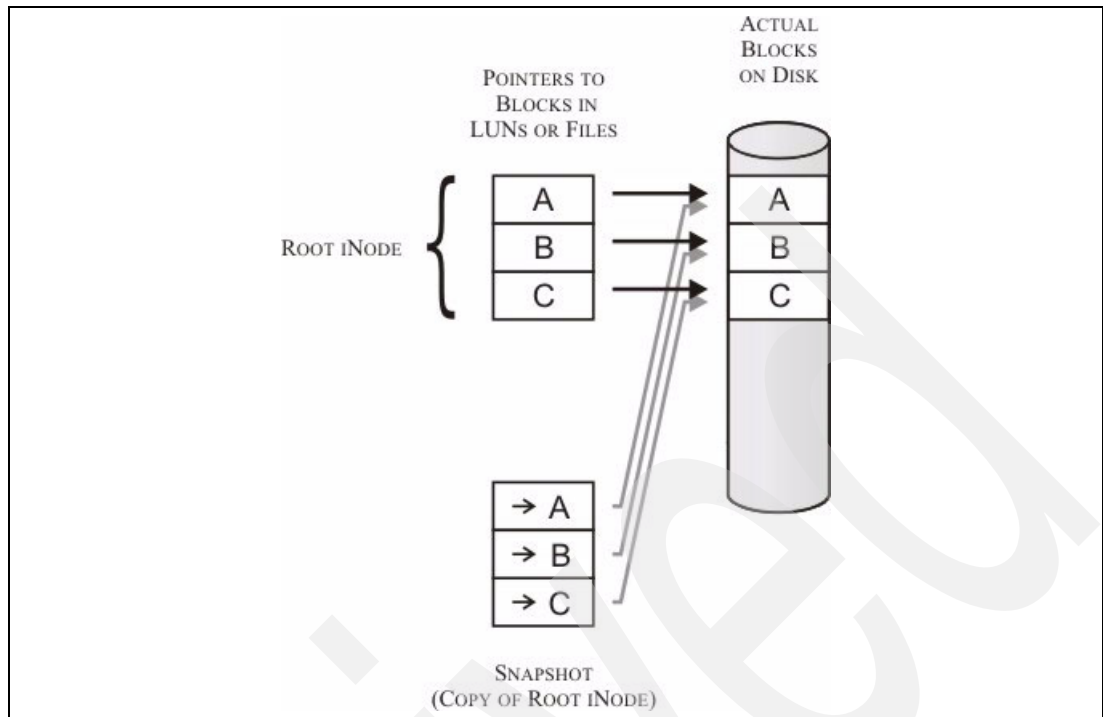


Figure 2-14 A Data ONTAP Snapshot

By duplicating just the root inode, Data ONTAP can create Snapshots in just a few seconds and each new Snapshot requires only a minimal amount of additional disk storage space (and generates very little disk I/O). Snapshot performance is important because Data ONTAP creates a Snapshot every few seconds to allow for quick recovery of an IBM System Storage N series after an unexpected system shutdown occurs.

Data ONTAP avoids changing blocks a Snapshot refers to by writing modified data to new locations on disk. Additionally, WAFL uses a “copy-on-write” technique to avoid duplicating disk blocks that are the same in both the active file system and any existing Snapshot. Only when blocks in the active file system are modified or removed do Snapshots referring to those blocks begin to consume disk space. Figure 2-15 on page 47 shows how an IBM System Storage N series looks when data is changed after a Snapshot is taken.

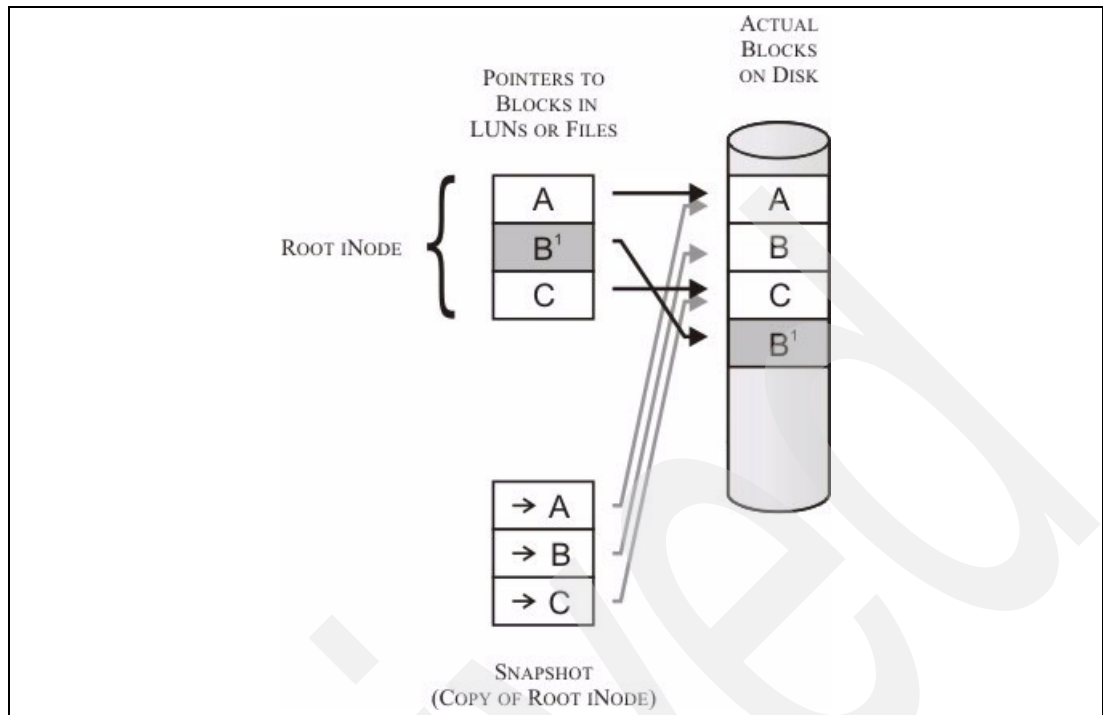


Figure 2-15 IBM System Storage N series when data is changed after a Snapshot is taken

In Figure 2-15, when a user modifies data block B, WAFL writes the new data to block B' on disk and changes the root inode for the active file system to point to the new block. The Snapshot still references the original block B, which remains unaltered on disk. Because disk block B is participating in a Snapshot, it is marked by Data ONTAP as being "in use" and is not returned to the available disk block pool. If another Snapshot is taken at this time, the meta-data that the new Snapshot describes is for disk blocks A,B',C, and D. Figure 2-16 on page 48 shows how an IBM System Storage N series looks if a Snapshot is taken each time a block on disk is modified.

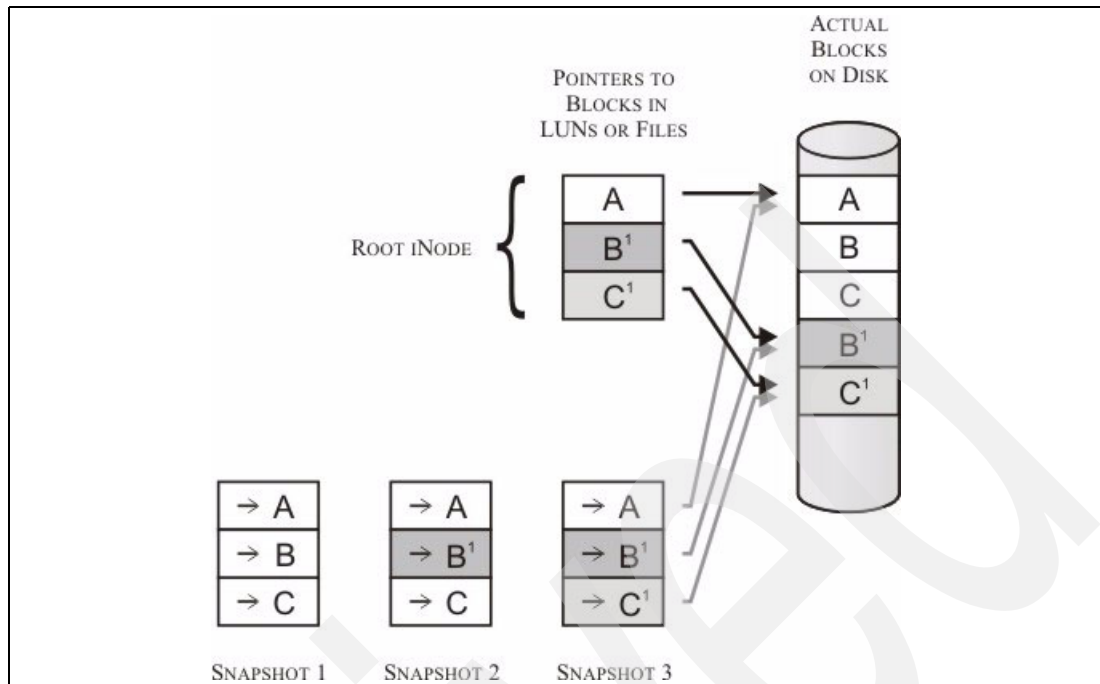


Figure 2-16 How multiple Snapshots on an IBM System Storage N series look

Over time, as files in the active file system are modified or deleted, a Snapshot can reference more and more blocks that are no longer used by the active file system. The rate at which files change determines how long Snapshots can be kept on line before they consume an unacceptable amount of disk space.

2.2.2 Other storage system's Snapshots

Other storage systems use a BASE + OFFSET architectural paradigm, where volumes are defined by a BASE, and blocks are accessed at an OFFSET from the BASE. As a result, any data block that become part of a Snapshot must be physically copied from its original offset location to a new offset location before the original block can be altered.

Other storage system's begin creating a new Snapshot by making a duplicate copy of the pointers that refer to blocks on disk. Then, a special storage area known as a *copy out* region is created on disk, and a pointer to this location is stored in the Snapshot along with the block pointers. Thus, each new Snapshot consumes disk space to store active file system block pointers and to hold a copy out region. Figure 2-17 on page 49 illustrates how other storage systems create a new Snapshot.

Before other storage systems can change a block a Snapshot refers to, they must first read the contents of the original block in the active file system, create a new block in the copy out region, and write the original block's contents to a new block. Then, they must update the block pointer in the Snapshot so that it references the block just created in the copy out region. Finally, when both of these operations are completed, the modified data is written to its original offset location on disk. Figure 2-18 on page 49 shows how other storage systems will look when data is changed after a Snapshot is taken.

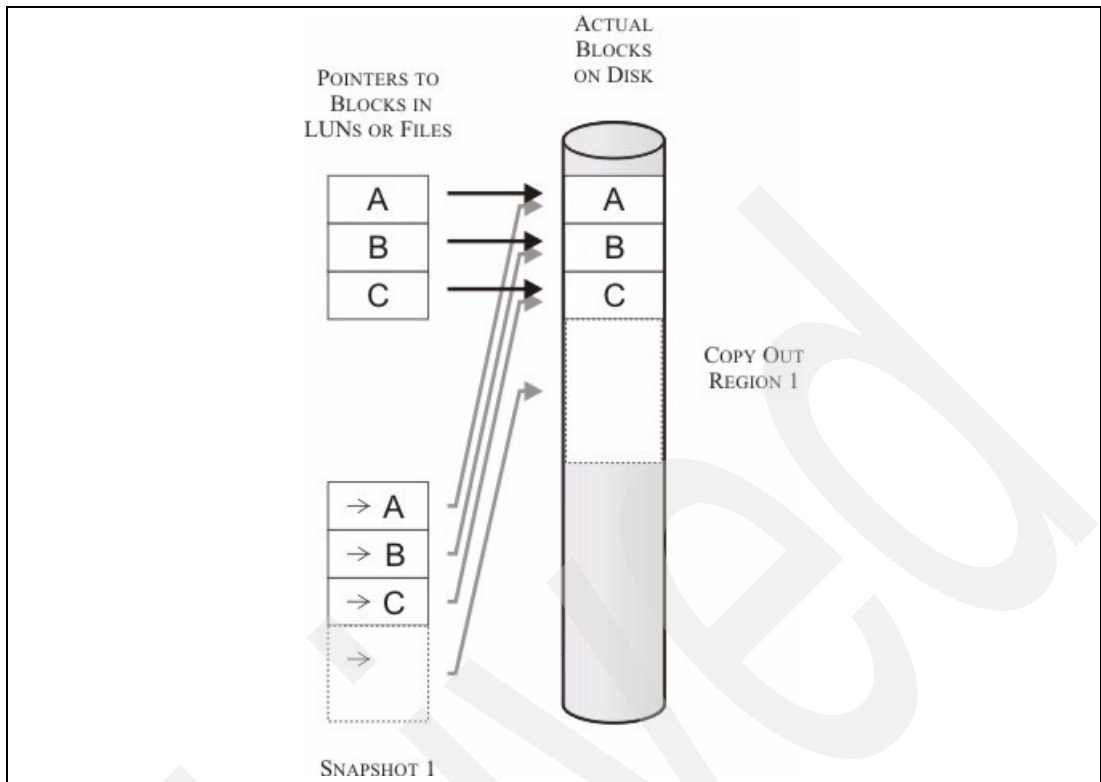


Figure 2-17 Another storage system's Snapshot

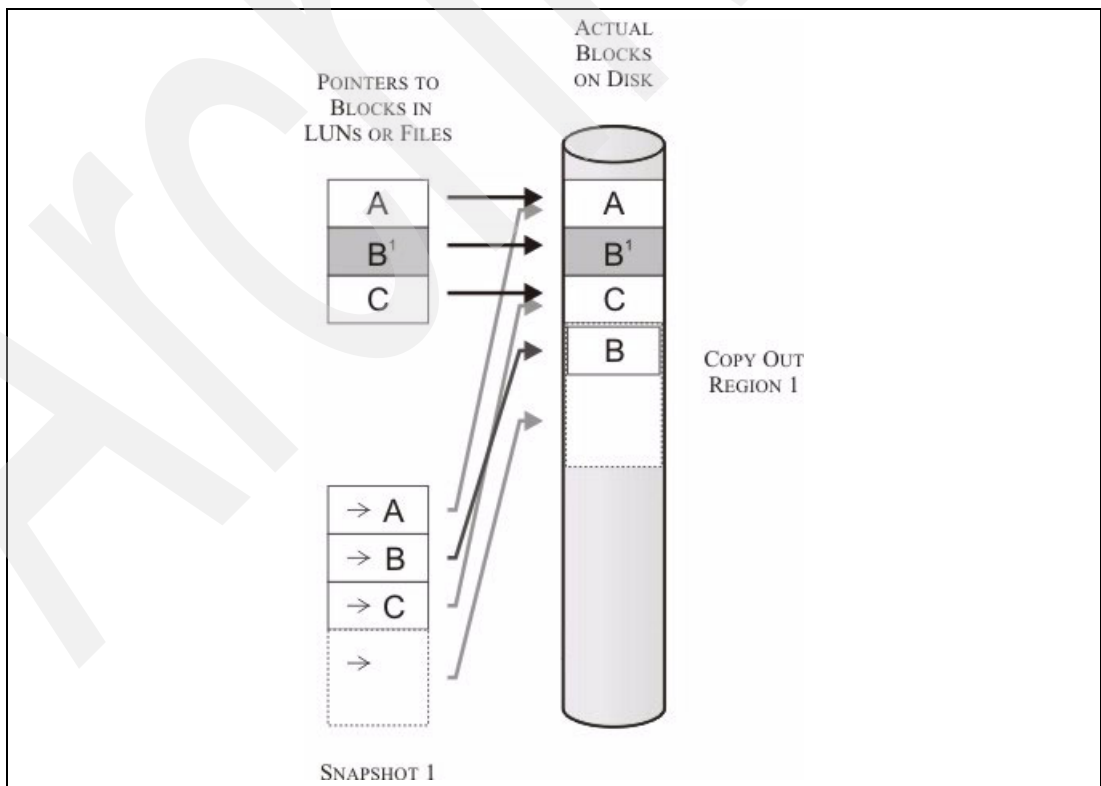


Figure 2-18 Other storage systems when data is changed after a Snapshot is taken

That means that after a Snapshot is taken, each update operation performed generates three I/O requests (1 read - old data, 1 write - old data, and 1 write - new data). However, the amount of I/O required, along with the amount of additional storage space needed is magnified each time a new Snapshot is taken. When multiple Snapshots exist and the block a Snapshot refers to is changed, the storage system must read the contents of the original block in the active file system, create a new block in *each* copy out region that exists, write the original block's contents to *every* new block created, and then update the block pointer in *every* Snapshot so that it references the block just created in that Snapshot's copy out region. Finally, when all of these operations are complete, the modified data is written to its original offset location on disk. Figure 2-19 shows how other storage systems look if two blocks are modified after two different Snapshots are taken.

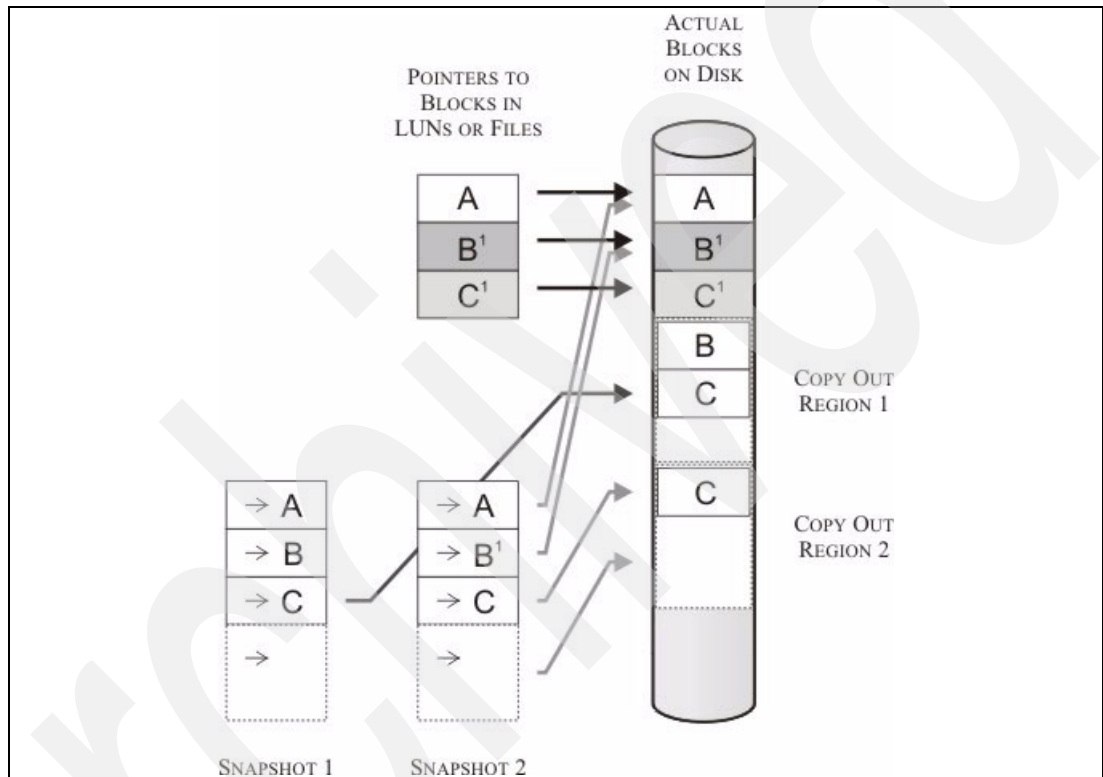


Figure 2-19 How multiple Snapshots on other storage systems look

It is also important to note that other storage systems make an educated guess when it comes to deciding on how much storage space to allocate for a Snapshot's copy out region. The wrong guess can result in inefficiencies in the way storage space is managed: if the copy out region created is too small, it can fill up and cause the Snapshot to become invalid (in which case it may need to be recreated). If the copy out region is too large, the volume can fill up with data leaving a large portion of disk space unused, yet unavailable to the active file system.

2.2.3 SnapRestore

SnapRestore leverages the Snapshot feature of Data ONTAP to perform near-instantaneous data restoration. It can be used to recover a damaged or deleted file or to recover a corrupted database, application, or damaged file system. Using SnapRestore, a system administrator can return a file, a LUN, an entire file system, or a volume to an earlier preserved state.

The way SnapRestore works is simple: after you select an existing Snapshot that you want to use for reversion, Data ONTAP replaces the root inode that represents the active file system with the inode copy that was created when the Snapshot was taken, if reverting at the volume level. (If reverting at a lower level, pointers within the inode file are replaced). This causes a volume or file to be returned to the state it was in at the time the Snapshot was created. As a result, any changes made to the volume or file (new data added, existing data altered or deleted) after the Snapshot was taken is lost. Figure 2-20 illustrates what happens when SnapRestore is used to revert a volume from an existing Snapshot.

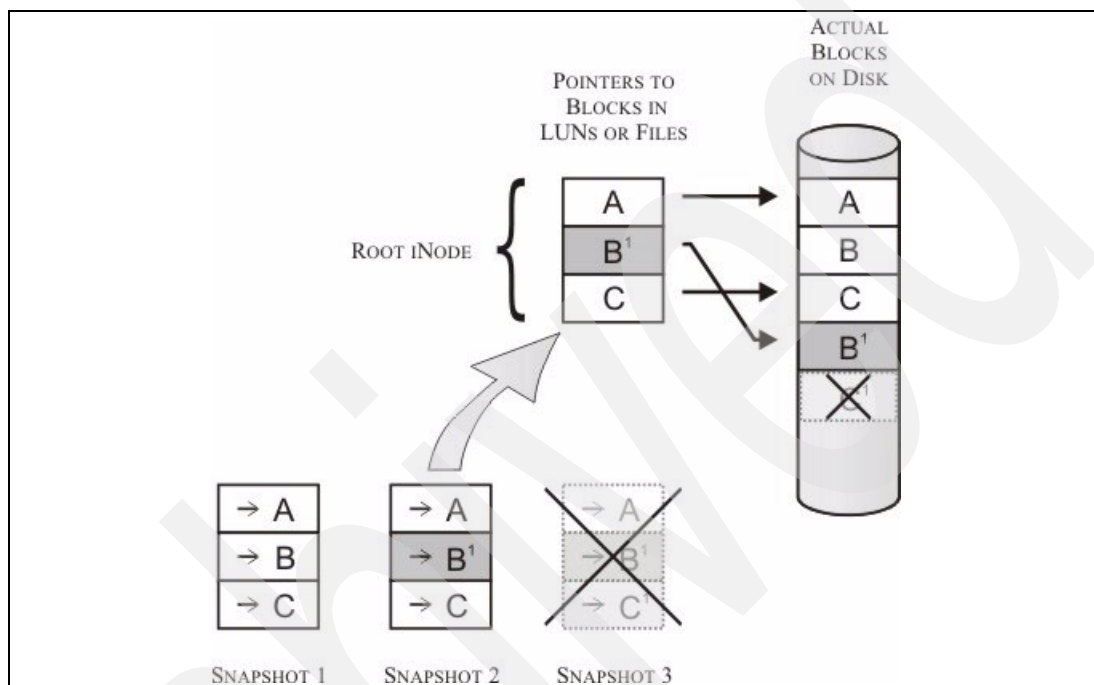


Figure 2-20 Reverting a volume with SnapRestore

Important: You cannot undo a SnapRestore operation. After volume or file reversion is performed, the volume or file cannot be returned to the state it was in before the reversion took place.

It is important to note that SnapRestore only reverts file contents and attributes. It does not revert volume attributes such as the volume option settings, RAID group size, and maximum number of files per volume. The one exception to this rule is if the root volume is reverted—Data ONTAP option settings are stored in a registry in the `/etc` directory on the root volume; thus, if the root volume is reverted, the registry is reverted.

Important: When you revert a volume using a specific Snapshot, you lose all Snapshots that are more recent than the Snapshot that was used for the volume reversion.

If you look closely at Figure 2-20, you will notice that when the volume was reverted using Snapshot 2, Snapshot 3 was lost along with the block C' on disk. That is because at the time Snapshot 2 was created, Snapshot 3 did not exist.

2.3 Continuous Availability and disaster recovery

The IBM N series offers a cost effective means to Continuous Availability and disaster recovery. The following sections will detail the options to protect your data.

2.3.1 SnapMirror

SnapMirror is a software product that allows a data set to be replicated between IBM N series, over a network, for backup or disaster recovery purposes. After an initial baseline transfer of the entire data set, subsequent updates only transfer new and changed data blocks from the source to the destination, which makes SnapMirror highly efficient in terms of network bandwidth utilization. At the end of each replication event, the mirror target volume becomes an identical copy of the mirror source volume. The destination file system is available for read-only access, or the mirror can be “broken” to enable writes to occur at the destination. After the mirror is broken, it can be reestablished by replicating the changes made at the destination back onto the source file system.

A SnapMirror environment is established by adding a second IBM N series to the network and configuring it as a mirror destination. Figure 2-21 illustrates the initial asynchronous synchronization of a simple SnapMirror environment. In this example, a Snapshot is created at the start of the synchronization process and copied to the mirror destination. Then, all corresponding data blocks are copied to the destination in the background while data continues to change at the source.

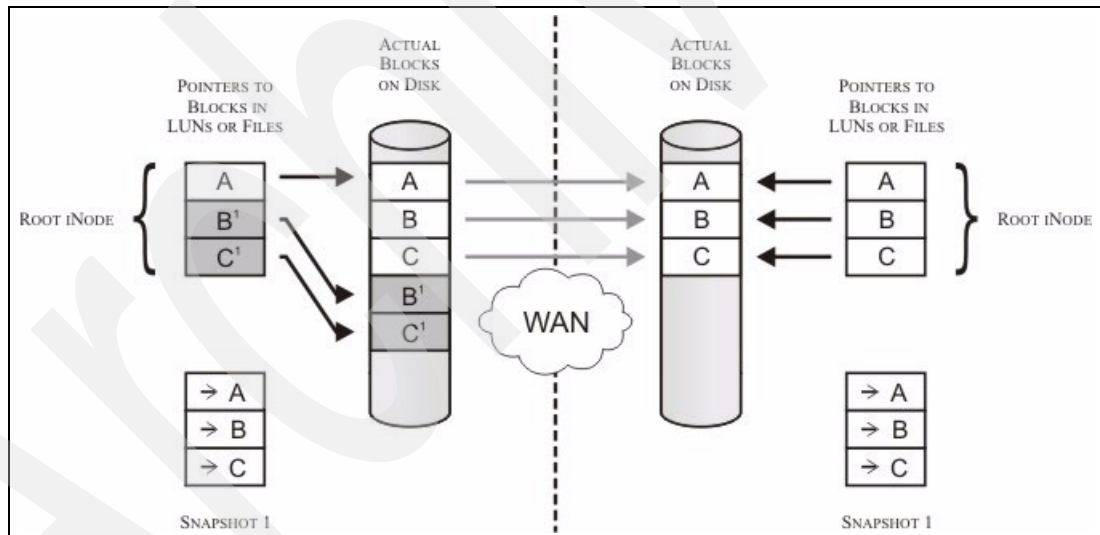


Figure 2-21 Initial asynchronous synchronization of a simple SnapMirror environment

Figure 2-22 on page 53 illustrates a subsequent asynchronous synchronization of the SnapMirror environment shown in Figure 2-21. During the subsequent synchronization process, another Snapshot is taken and copied to the destination. Then, only the data blocks that were added or changed since the last synchronization are copied to the destination IBM N series. This has no effect on the blocks and Snapshot transferred during the previous synchronization process.

In the traditional asynchronous mode of operation, updates of new and changed data from the source to the destination occur on a schedule that is defined by the storage administrator. These updates could be as frequent as once per minute or as infrequent as once per month, depending on user needs. In synchronous mode (which is also available), updates are sent

from the source to the destination as they occur, rather than on a predefined schedule. When configured correctly, this mode of operation can guarantee that data written on the source system is protected on the destination even if the entire source system fails due to natural or human disaster. A semi-synchronous mode is also provided, which can minimize loss of data in a disaster while also minimizing the performance impact of performing replication at the source system. In order to maintain consistency and provide ease of use, the asynchronous and synchronous interfaces are identical with the exception of a few additional parameters in the configuration file.

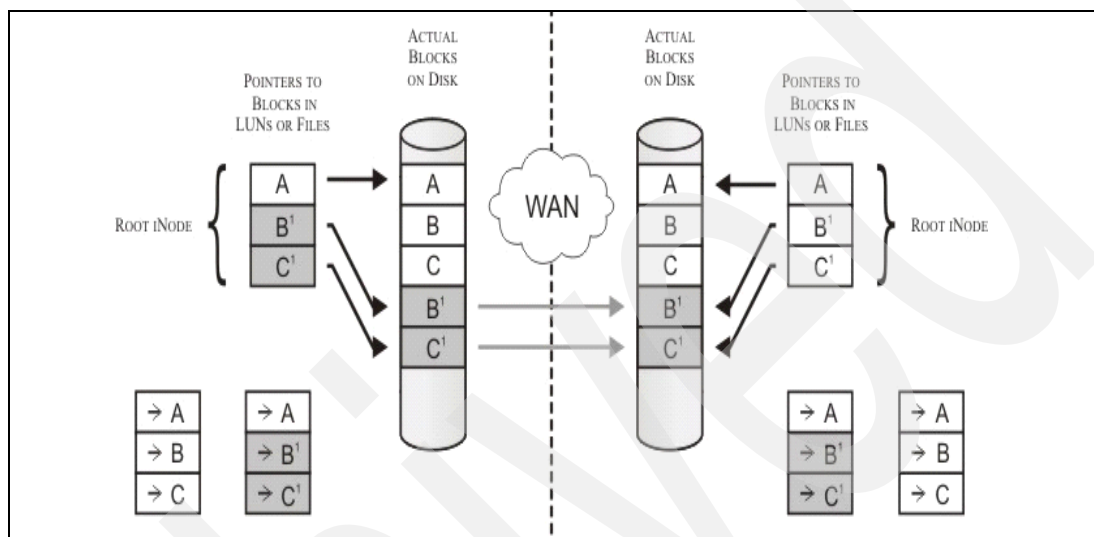


Figure 2-22 Subsequent asynchronous synchronization of a simple SnapMirror environment

Architecturally, SnapMirror software is a logical extension of the WAFL file system and in particular, the Snapshot feature. Using Snapshots, you can create a read-only copy of an entire filer volume. Two sequential Snapshots can then be compared and the differences identified. Because this comparison takes place at the block level, only the changed blocks need be sent to the mirror target. By implementing the update transfers asynchronously, data latency issues inherent with remote synchronous mirroring techniques are eliminated. The elegance of these two design features becomes particularly apparent when running mirror pairs over WAN topologies.

2.3.2 SnapVault

SnapVault is a low overhead, disk-based online backup of heterogeneous storage systems for fast and simple restores.

SnapVault is a separately licensed feature in Data ONTAP that provides disk-based data protection for IBM System Storage N series. SnapVault replicates selected Snapshots from multiple client IBM System Storage N series to a common Snapshot on the SnapVault server, which can store many Snapshots. These Snapshots on the server have the same function as regular tape backups. Periodically, data from the SnapVault server can be dumped to tape for extra security.

SnapVault software protects data residing on a SnapVault primary by maintaining online backup copies (Snapshots) on a SnapVault secondary system. A SnapVault primary system corresponds to a backup client in a traditional tape backup architecture. The SnapVault secondary is always an IBM System Storage N series running Data ONTAP.

One of the unique benefits of SnapVault is that users do not require special software or privileges to perform a restore of their own data. Any user who wants to perform a restore of his own data may do so without the intervention of a system administrator, saving end-user time and money, as well as freeing up valuable administrator time. If required by policies, data recovery can be restricted to authorized individuals as well.

Recovering a file from a SnapVault backup is simple. Just as the original file was accessed via an NFS mount or CIFS share, the SnapVault secondary may be configured with NFS exports and CIFS shares. As long as the destination qtrees are accessible to the users, restoring data from the SnapVault secondary is as simple as copying from a local Snapshot image. Recovery of an entire data set can be performed the same way if the user has appropriate access rights.

SnapVault and SnapMirror often work hand in hand (Figure 2-23). However, when using SnapMirror in combination with SnapVault, it is important to keep in mind that because SnapMirror and SnapVault both use the same Snapshot copies, they cannot run simultaneously. Therefore, they must be ran independently of each other if both are used in the same environment.

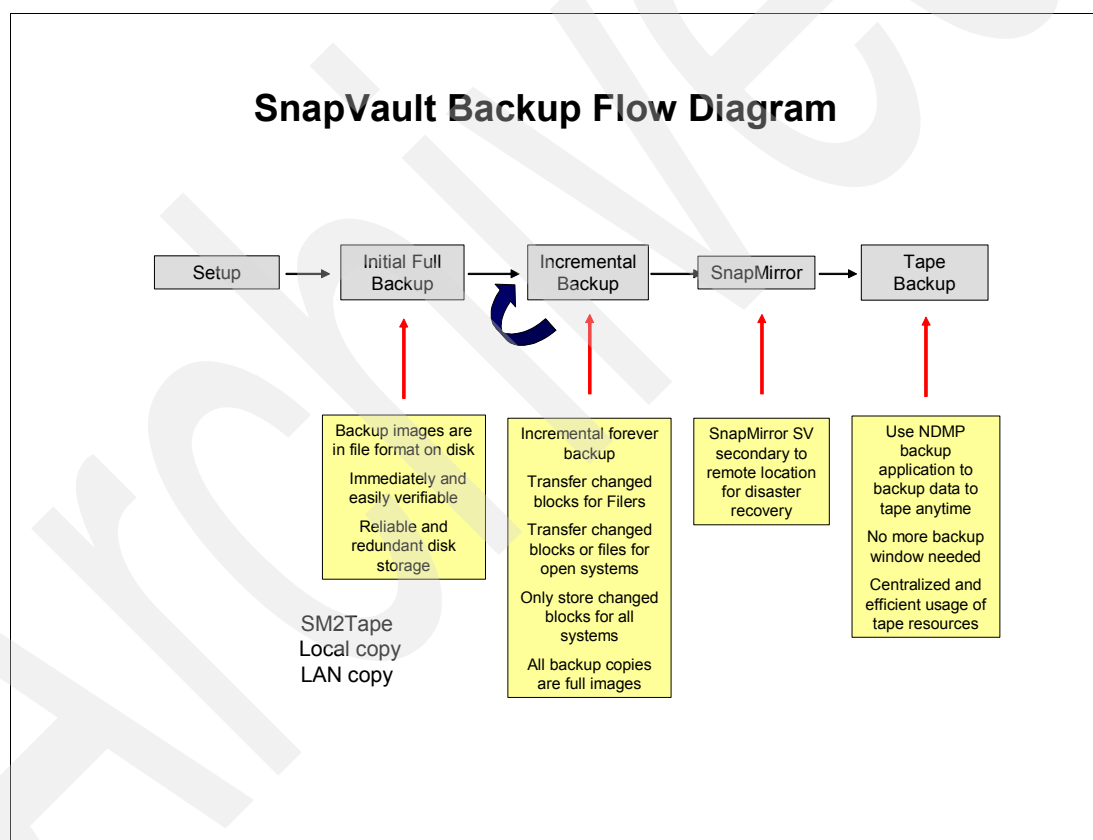


Figure 2-23 SnapVault and SnapMirror

2.4 Managing storage

There are multiple areas and means to manage the IBM System Storage N series' storage. The following sections detail how to accomplish efficient management of your N series storage.

2.4.1 Aggregates and RAID Groups

An *aggregate* is simply a pool of disks (Figure 2-24 on page 56) that are comprised of one or more RAID groups. A *RAID group* is a collection of one or more data disks, along with a parity disk, and a double-parity disk if RAID-DP is used. The minimum number of disks allowed in a single RAID group on an IBM N series is two if a RAID 4 configuration is used and three if a RAID-DP configuration is used. The maximum number of disks allowed for a RAID-DP configuration is 28 (26 data disks and two parity disks). The maximum number allowed for a RAID 4 configuration is 14 (13 data disks and one parity disk). The default RAID group type used for an aggregate is RAID-DP but can be changed to RAID 4. Figure 2-24 on page 56 shows two aggregates and their underlying RAID groups. In this example, the first aggregate (Aggregate A) consists of three RAID groups that are using RAID-DP. The second aggregate (Aggregate B) consists of one RAID group that is using RAID 4. Aggregate A has a total of 12 data disks available for storage. Aggregate B has a total of three data disks.

If necessary, additional disks can be added to an aggregate after it is created. However, disks must be added such that the RAID group specification for the aggregate remains intact. For example, in order to add disks to the first aggregate (Aggregate A) shown in Figure 2-24 on page 56, a minimum of three disks are required—one data, one parity, and one double-parity. These three disks are placed in a fourth RAID group.

Larger versus smaller RAID groups

Configuring an optimum RAID group size for an aggregate often requires a trade-off. You must decide which factor, speed of recovery, assurance against data loss, or maximizing data storage space, is most important for the aggregate that you are creating.

Advantages of large RAID groups

Large RAID group configurations offer the following advantages:

- ▶ More data drives available. A volume configured into a few large RAID groups requires fewer drives reserved for parity than that same volume configured into many small RAID groups.
- ▶ Better system performance. Read/write operations are usually faster over large RAID groups than over smaller RAID groups.

Advantages of small RAID groups

Small RAID group configurations offer the following advantages:

- ▶ Shorter disk reconstruction times. In case of disk failure within a small RAID group, data reconstruction time is usually shorter than within a large RAID group.
- ▶ Decreased risk of data loss due to multiple disk failures. The probability of data loss through double disk failure within a RAID 4 group or triple disk failure within a RAID-DP group is lower within a small RAID group than in a large RAID group.

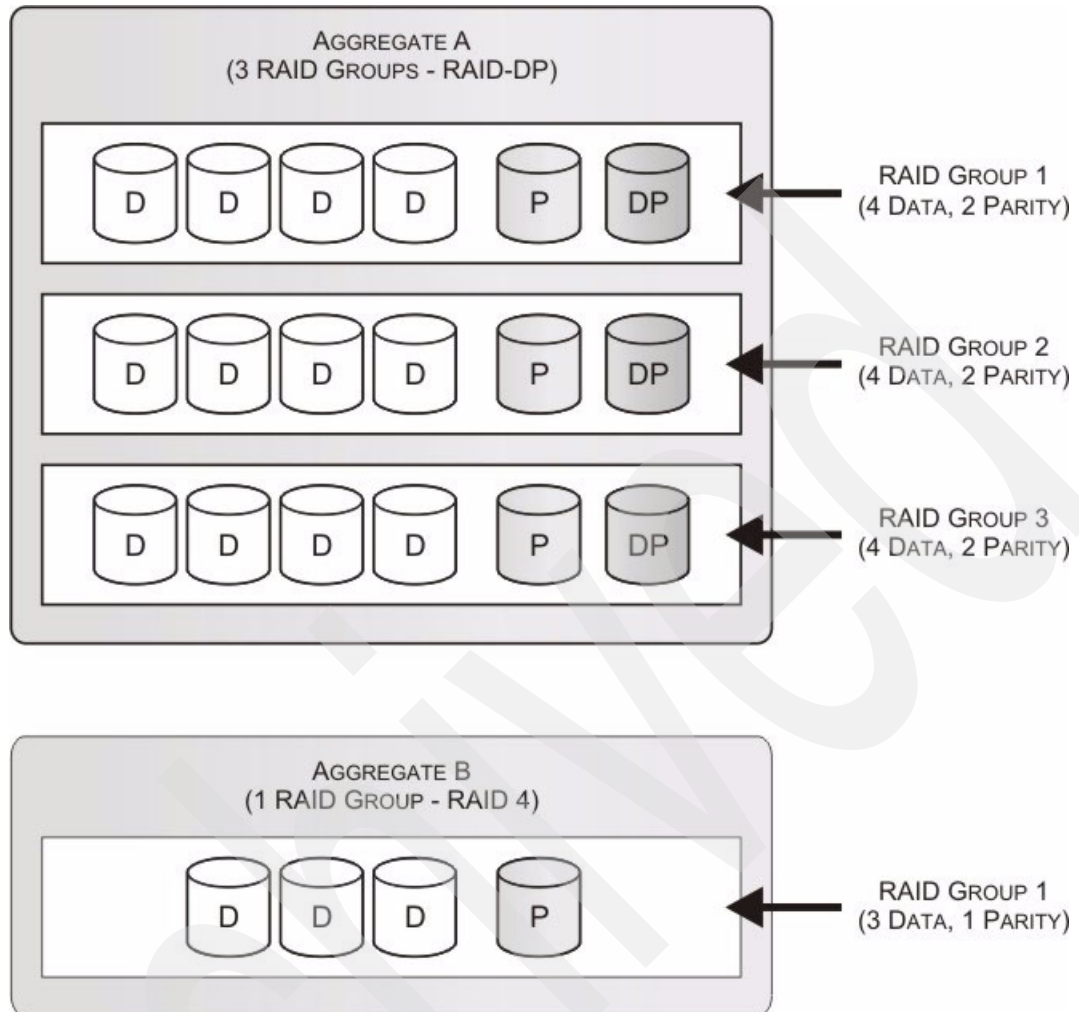


Figure 2-24 Aggregates and RAID groups

2.4.2 Introducing FlexVols

From a system administrator's point-of-view, volumes—not aggregates or RAID groups—are the primary unit of data management. A FlexVol volume is simply a “pool” of storage that can be sized according to how much data is to be stored in it, rather than on the physical size of the disk drives used. FlexVol volumes are logical data containers; therefore, they can be sized, resized, managed, and moved independently of their underlying physical storage. FlexVol volumes are not bound by the limitations of the disks on which they reside.

Each volume depends on its containing aggregate for all its physical storage, that is, for all storage in the aggregate's disks and RAID groups. Because the FlexVol is managed separately from the aggregate, you can create small FlexVol volumes (20 MB or larger), and you can increase or decrease the size of FlexVol volumes in increments as small as 4 KB.

A system administrator can reconfigure relevant FlexVol volumes at any time. The reallocation of storage resources does not require any downtime and is transparent to users regardless of whether a file system is used by the FlexVol or a LUN mapped to a host in a block environment. Furthermore, reallocation of storage resources is non disruptive to all clients connected to the FlexVol being resized.

Improved performance

FlexVol volumes (Figure 2-25) have all the data spindles in an aggregate available to them at all times. Furthermore, a FlexVol volume can share its containing aggregate with other FlexVol volumes. Thus, a single aggregate can be the shared source of all the storage used by all the FlexVol volumes contained by that aggregate. The unused space is managed by the aggregate so that unallocated space in one FlexVol does not impact the space used in another FlexVol within the same aggregate. Thus, for I/O-bound applications, FlexVol volumes can run much faster than equivalent-sized traditional volumes.

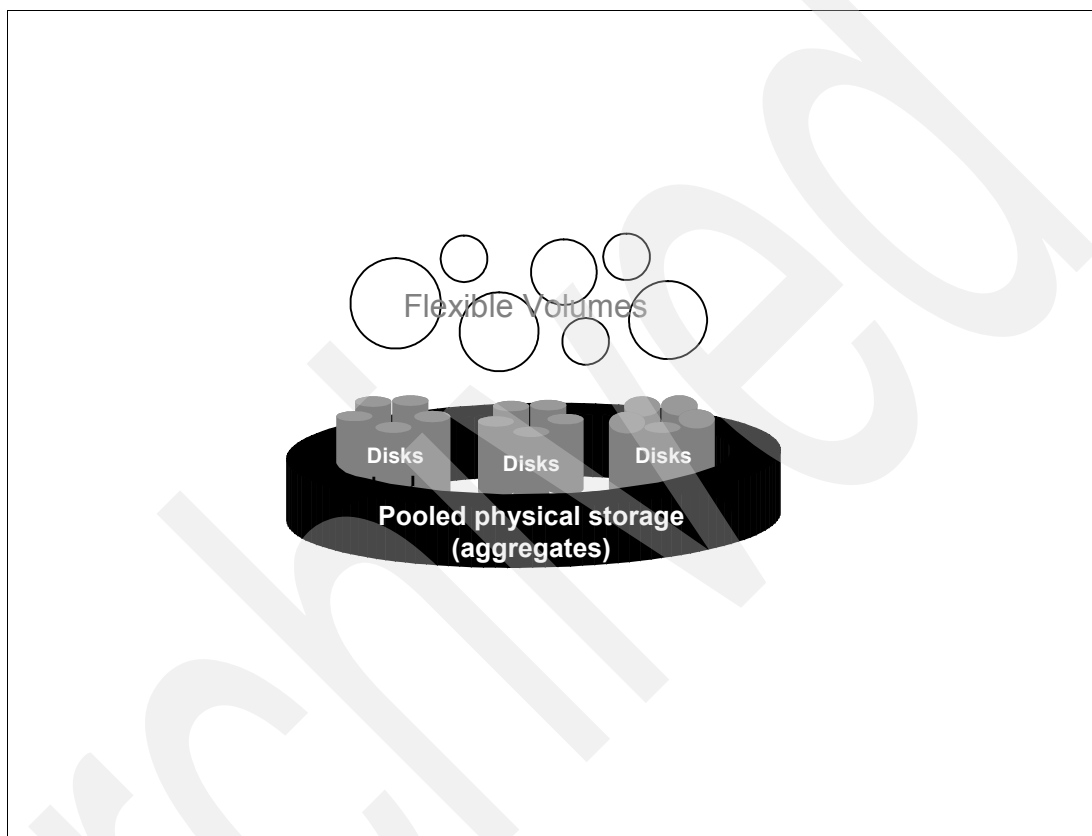


Figure 2-25 FlexVols

Improved disk utilization

With FlexVol there is no preallocation required, for example no special scripts, provisioning, or formatting of physical or logical drives. Free space is aggregated and shared to reduce space waste. This in turn increases utilization and reduces total free space available.

When additional physical storage space is required, the system administrator can increase the size of the aggregate by assigning additional disks to it. As soon as new disks become part of an aggregate, their capacity and I/O bandwidth are available to all of the flexible volumes in that aggregate.

Overall FlexVol capacity can also be over allocated where the set capacity of all the flexible volumes on an aggregate exceeds the total physical space available. Increasing the capacity of one FlexVol volume does not require changing the capacity of another FlexVol in the aggregate or changing the capacity of the aggregate itself.

Capacity guarantees in FlexVol

Along with FlexVol volumes, Data ONTAP introduces the storage management concept of *guarantees*. A guarantee differs from the previous management concept of *space reservations*, which is familiar to customers using the iSCSI or Fibre Channel facilities of Data ONTAP. Guarantees extend the control by allowing administrators to set policies that determine how much storage is actually pre allocated when volumes or files are created. This allows administrators to effectively implement the concept of *thin provisioning*. This concept of thin provisioning enables the administrator to, in effect, oversubscribe their storage safely. Provision your storage space once, then grow as needed in the aggregate.

Guarantees, set at the volume level, determine how the aggregate pre allocates space to a flexible volume. When you create a FlexVol volume within an aggregate, you specify the capacity and, optionally, the type of guarantee.

There are three types of guarantees:

1. **Volume:** A guarantee type of volume ensures that the amount of space required by the flexible volume is always available from its aggregate. This is the default setting for flexible volumes.
2. **File:** With the file guarantee type, the aggregate guarantees that space is always available for writes to space-reserved LUNs or other files.
3. **None:** A flexible volume with a guarantee type of none reserves no space, regardless of the space reservation settings for LUNs in that volume. Write operations to space-reserved LUNs in that volume might fail if the containing aggregate does not have enough available space.

2.4.3 Cloning FlexVols with FlexClone

A FlexClone volume is a writable point-in-time image of a FlexVol volume or another FlexClone volume. FlexClone volumes add a new level of agility and efficiency to storage operations. They take only a few seconds to create and can be created without interrupting access to their parent FlexVol volume. Furthermore, FlexClone volumes have the same high performance as their parent volumes.

FlexClone volumes have all the capabilities of a FlexVol volume, including the ability to grow or shrink, to be the source of a Snapshot copy, or to be the source of another FlexClone volume. The technology that makes this all possible is integral to how Data ONTAP manages storage. As we saw earlier, the IBM Storage System N series uses a Write Anywhere File Layout (WAFL) file system to manage disk storage. New data that gets written to a volume does not have to go on a specific spot on the disk; instead, it can be written anywhere. WAFL then updates the metadata to integrate the newly written data into the right place in the file system. If the new data is meant to replace older data, and the older data is not part of a Snapshot copy, WAFL marks the blocks containing the old data as “reusable”. This can happen asynchronously and does not affect performance. Snapshots work by making a copy of the metadata associated with a volume. Data ONTAP preserves pointers to all the disk blocks currently in use at the time a Snapshot copy is created. When a file is changed, the Snapshot copy still points to the disk blocks where the file existed before it was modified, and changes are written to new disk blocks. As data is changed in the parent FlexVol volume, the original data blocks stay associated with the Snapshot copy rather than getting marked for reuse. All the metadata updates are just pointer changes, and the filer takes advantage of locality of reference, NVRAM, and RAID technology to keep everything fast and reliable.

You can think of a FlexClone volume as a transparent writable layer in front of a Snapshot copy. A FlexClone volume is writable, so it needs some physical space to store the data that is written to the clone. It uses the same mechanism used by Snapshot copies to get available

blocks from the containing aggregate. Whereas a Snapshot copy simply links to existing data that was overwritten in the parent, a FlexClone volume stores the data written to it on disk (using WAFL) and then links to the new data as well. The disk space associated with the Snapshot copy and FlexClone is accounted for separately from the data in the parent FlexVol volume. When a FlexClone volume is first created, it needs to know the parent FlexVol volume and a Snapshot copy of the parent to use as its base. The Snapshot copy can already exist, or it can get created automatically as part of the cloning process. The FlexClone volume gets a copy of the Snapshot copy metadata and then updates its metadata as the clone volume is created. Figure 2-26 shows how a FlexClone volume looks when it is first created. Creating a FlexClone volume takes just a few moments because the metadata copied is very small compared to the actual data.

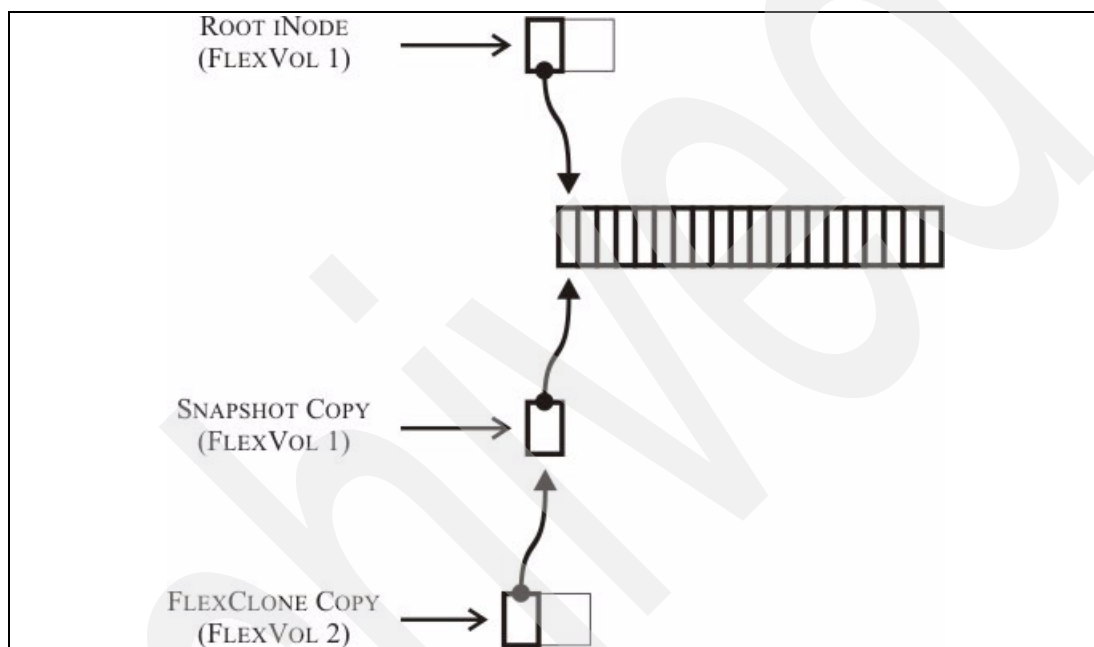


Figure 2-26 A FlexClone volume after creation

After a FlexClone is created, the parent FlexVol volume can change independently of the FlexClone volume because the Snapshot copy is there to keep track of the changes and prevent the original parent's blocks from being reused while the Snapshot copy exists. The same Snapshot copy is read-only and can be efficiently reused as the base for multiple FlexClone volumes. Space is used very efficiently, since the only new disk space used is either associated with the small amounts of metadata or updates and additions to either the parent FlexVol volume or the FlexClone volume. (FlexClone volumes leverage the Data ONTAP WAFL file system to only store changed blocks.) Initially, a clone and its parent share the same storage. More storage space is consumed only when one volume or the other is changed.

FlexClone volumes appear to the storage administrator just like any FlexVol volume, which is to say that they look like a regular volume and have all of the same properties and capabilities. In fact, FlexClone volumes are treated just like a FlexVol volume for most operations. Using the CLI, FilerView or Operations Manager, one can manage volumes, Snapshot copies, and FlexClone volumes including getting their status and seeing the relationships between the parent, Snapshot copy, and clone. The main limitation is that Data ONTAP forbids operations that would destroy the parent FlexVol volume or base Snapshot copy while dependent FlexClone volumes exist. Other caveats are that management information in external files associated with the parent FlexVol volume is not copied, quotas for the clone volume get reset rather than added to the parent FlexVol volume, and LUNs in

the cloned volume are automatically marked offline until they are uniquely mapped to a host system.

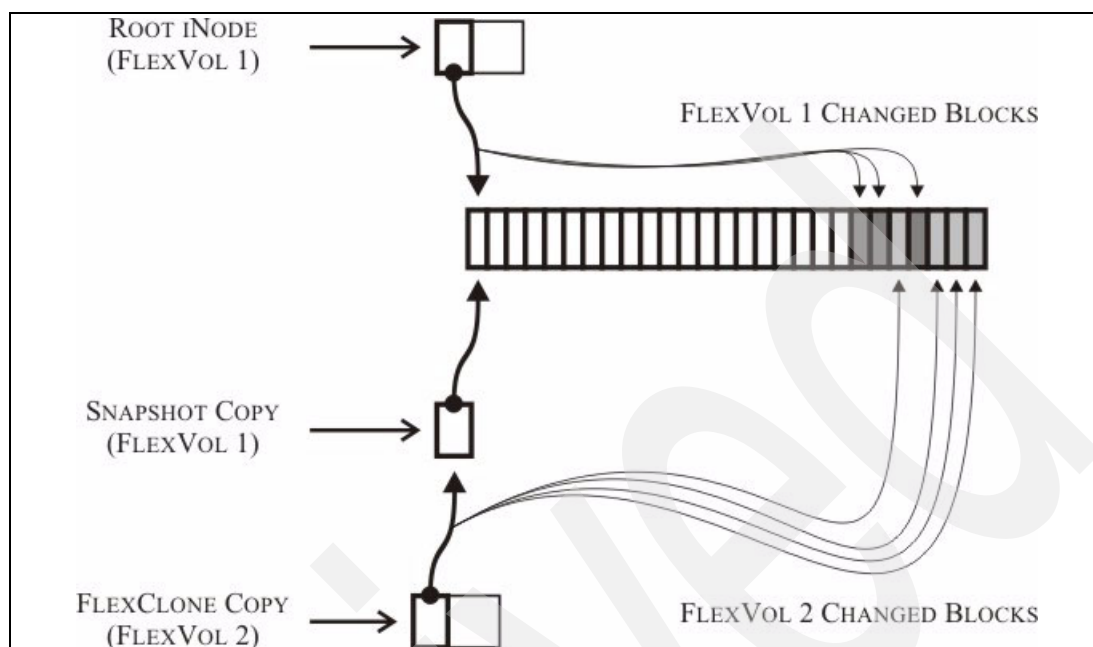


Figure 2-27 How changes to parent FlexVol volumes and FlexClone volumes are stored

Lastly, a FlexClone volume can be split from the parent volume to create a fully independent volume. However, splitting a clone requires adequate free space in the aggregate to copy the shared blocks. Splitting the clone to create a fully independent volume also uses resources. While the split is occurring, free blocks in the aggregate are used to copy blocks shared between the parent and the clone. This incurs disk I/O operations and can potentially compete with other disk operations in the aggregate. The copy operation also uses some CPU and memory resources, which may impact the performance of a fully loaded filer. Data ONTAP addresses these potential issues by completing the split operation in the background and sets priorities in a way that does not significantly impact foreground operations. It is also possible to manually stop and restart the split operation if some critical job requires the full resources of the filer.

Practical applications of FlexClone technology

Conceptually, FlexClone volumes are great for any situation where testing or development occurs, any situation where progress is made by locking in incremental improvements, and any situation where there is a desire to distribute data in changeable form without endangering the integrity of the original.

For example, imagine a situation where a database administrator needs to apply a database product fix pack to a production database. The cost and risk of a mistake are too high to apply the fix pack to the production database without verifying that the fix pack does not adversely affect the database. Ideally, there is an instant writable copy of the production database available at minimal cost in terms of storage and service interruptions. By using FlexClone volumes, the DBA gets just that, an instant point-in-time copy of the production data that is created transparently and uses only enough space to hold the desired changes. They can then test the fix pack using the FlexClone volume. If the fix pack does not adversely affect the database clone, it can be applied to the production database, and the clone can be destroyed. On the other hand, if the fix pack creates a problem, the DBA can just destroy the working clone and report the problem to the database vendor. When everything is finally

working just the way it should, the DBA can either split off the clone to replace their current production volumes or document their steps and apply them to the production system during the next maintenance window. The results are less risk, less stress, and higher levels of service for the database customers.

Archived

Introduction to enterprise databases

In part 2 we provide a summary introduction to the databases we discuss in this Redbooks publication. We intend for this introduction section to familiarize the Storage Administrator or N series Administrator with the concepts of the databases discussed in this book.

Archived

Introduction to IBM DB2 Universal Database

DB2 Universal Database Version 8 (DB2 UDB or simply, DB2) is a relational database management system (RDBMS) developed by IBM. It is available on Linux, AIX, HP-UX, Solaris, and Windows (commonly referred to as Linux, Unix, and Windows, or LUW for short that refers to all three) operating environments. It is also available in various editions, ranging from a standalone version that supports a single user to a fully scalable version that allows your database to span over multiple servers running the same operating system. This scalability allows DB2 to grow along with the exponential growth of data that enterprises contend with today. This exponential growth of data drives the need for flexible storage systems like the IBM N series.

This chapter is geared towards the storage administrator and contains information regarding DB2, which has relevance in an NAS or SAN environment.

3.1 DB2 Universal Database editions

DB2 UDB Version 8 for LUW environments is available in the following editions:

DB2 UDB Personal Edition (PE) is a complete RDBMS for a single user. It does not support remote client connections. This version is available only on Windows and Linux operating systems. This edition might be better suited to the N3700 models using CIFS or NFS.

DB2 UDB Personal Developer's Edition (PDE) enables a developer to design and build single user desktop applications. This version is available only on Windows and Linux operating systems and includes DB2 Personal Edition as well as the DB2 Extenders™. This edition also might be better suited for N3700 models using CIFS or NFS.

DB2 UDB Express Edition is a low cost RDBMS with self-tuning and self-configuring autonomic capabilities. This is ideal for businesses that require a database with minimal administration and management. It includes the same functionality as DB2 Workgroup Server Edition and also features on-click installation and other ease-of-use features. It is available for Windows and Linux operating systems and supports up to two processors. This edition could utilize the N3700 and N5000 series and protocols, such as CIFS, NFS, SAN, or iSCSI.

DB2 UDB Workgroup Server Edition (WSE) is designed for deployment in a small-to-medium sized business environment. It is available on LUW servers running on, at most, four processors. This product can be licensed by user or by processor (known as DB2 UDB Workgroup Server Unlimited Edition). This edition can store its database files and logs on the N3700, N 5000, and N 7000 series using CIFS, NFS, SAN, or iSCSI.

DB2 UDB Enterprise Server Edition (ESE) is the most flexible and powerful of all of the editions. It is geared towards mid-to-large size businesses and is available on LUW servers of any size with any number of processors. It is also fully scalable, accessible, and extensible. This version includes the Data Partitioning Feature (DPF), which allows partitioning of data over one or more servers running the same operating system. This edition benefits from the capabilities of the N5000 or N7000 series using CIFS, NFS, SAN or iSCSI.

DB2 UDB Universal Developer's Edition is a low cost package for a single application developer to design, build, and test applications before deployment in a production environment. This product includes all the functionality of DB2 Enterprise Server Edition, but also includes the DB2 Extenders, Warehouse Manager, and Intelligent Miner™. This edition could be used on the N3700, N5000, and N7000 series using CIFS, NFS, SAN, or iSCSI.

The edition used for the writing of this book is DB2 UDB Version 8.2 Enterprise Server Edition for AIX 5.2 maintenance level 7.

3.2 The universal database

DB2 UDB is truly the universal database supporting a wide variety of platforms and applications. By hosting DB2 UDB on the IBM N series you can combine both the features of dynamic growth and high performance.

3.2.1 Universal access

DB2 UDB provides universal access to all forms of electronic data. This includes traditional relational data as well as structured and unstructured binary information, documents, and text in many languages, graphics, images, multimedia (audio and video), and information specific

to operations such as engineering drawings, maps, insurance claim forms, numerical control streams, or any type of electronic information.

Access to a wide variety of data sources can be accomplished with the use of DB2 UDB and its complimentary products: WebSphere® Information Integrator and DB2 Connect™.

Examples of datasources that can be accessed include: DB2 UDB for z/OS®, DB2 UDB for OS/400®, Oracle, MS SQL Server, Sybase, NCR Teradata, and IBM Informix® databases.

3.2.2 Universal application

DB2 UDB supports a wide variety of application types. It can be configured to perform well for both online transaction processing (OLTP) as well as for decision support systems (DSS). It can also be used as the underlying database for an online analytical processing (OLAP) system.

DB2 UDB is also accessible from or can be integrated into a wide variety of application development environments. In addition to being able to embed SQL statements within source code files written in standard programming languages such as C, C++, and Visual Basic®, DB2 UDB fully supports Java™ technology and is accessible from Java applets, servlets, and applications. DB2 UDB also participates in Microsoft's OLE DB as both a provider and a consumer.

3.2.3 Universal extensibility

Data is stored in most relational databases according to its data type and DB2 UDB is no exception. In order to support a wide variety of data types and formats, DB2 UDB contains a rich set of built-in data types, along with a set of functions that are designed to manipulate each of these data types. DB2 UDB also provides a way to create user-defined data types (UDTs) and supporting user-defined functions (UDFs); consequently, the base data types provided can be extended to provide data types that are specific to your business needs.

Using UDTs and UDFs, IBM went one step farther and created several different sets of user-defined data types and functions to manage particular kinds of data that have begun to emerge over the last few years. Collectively, these sets of data types and functions are referred to as extenders. Combined with the current set of extenders available, DB2 can store and manipulate image, audio, video, text, XML, and spatial data, just to name a few.

3.2.4 Universal scalability

Scalability—the capability of a system to increase total throughput under an increased load when resources are added—is crucial to a business when dealing with the ever-growing amount of data available today.

The superior scalability of DB2 UDB is made possible through a combination of features that are built into the base product. These include intra-partition parallelism as well as inter-partition parallelism. With intra-partition parallelism, database operations are subdivided into multiple parts, which are then executed in parallel within a single database partition. With inter-partition parallelism, database operations are subdivided into multiple parts, which are then executed in parallel across one or more partitions of a multi-partition database. In addition, the DB2 UDB database engine is designed to take advantage of I/O parallelism (the process of reading or writing to two or more I/O devices at the same time) whenever possible, and it is capable of interacting with disk I/O subsystems that were designed with RAID technology in mind. The N series product mirrors the scalability capability of DB2 with its many benefits and features.

3.2.5 Universal reliability

DB2 UDB runs reliably across multiple hardware and operating systems. DB2 UDB uses write-ahead transaction logging as a preventative measure, and as a result, it can usually resolve database problems that are caused by power interruptions and application failures without any additional intervention. By using DB2 UDB on the N series product, further protection is provided with RAID 4 or RAID-DP.

An additional protection is provided with backup (and recovery) programs. And to help establish such a program, DB2 UDB provides a set of utilities that can be used to accomplish the following:

- ▶ Create a backup image of a database.
- ▶ Return a database to the state it was in when a backup image was made (by restoring it from a backup image).
- ▶ Reapply (or roll-forward) some or all changes made to a database since the last backup image was made after the database is returned to the state it was in when the backup image was made.

Backups can be scheduled to run automatically, and both full and incremental backup images can be made. Backups can also be tailored for a single tablespace or for an entire database, and backup images can be taken while a database is online or offline. An additional copy of these database images may be kept with little cost using IBM System Storage N series Snapshot.

3.2.6 Universal management

DB2 supplies a comprehensive list of tools to help the database administrator (DBA) manage their DB2 environment. Following are some of the more commonly used GUI tools available to the DBA:

Control Center	Primary administration tool for DB2 that includes the capability to create, modify, and delete database objects and change registry, database, and database manager parameters.
Command Editor	A GUI interface for running SQL, database, or operating system commands.
Task Center	Allows for the creation, scheduling, and execution of various types of tasks, including DB2 scripts and OS scripts.
Journal	Provides a GUI interface to view historical information about tasks, database operations, messages, and notifications.
Visual Explain	Provides a graphical view of optimization-associated cost information and visual drill-down views of a query's access plan.
Health Center	Allows the analysis of the general health of DB2. An example of this is checking on sufficient resources and ensuring tasks are completed within acceptable periods of time.
Activity Monitor	A tool to help monitor application performance.

All of the information presented in the Control Center can also be accessed via a command line interface known as the Command Line Processor (CLP).

3.3 The DB2 query optimizer

The query optimizer is the key component in the performance of any Enterprise Database Server. IBM has made more than a 25-year investment in enhancements to the DB2 cost-based, rule-driven optimizer with the goal of keeping it the best in the industry. A major component of this effort is the implementation of the Starburst extensible optimizer in DB2 UDB. This allows IBM to add new intelligence to the optimizer without having to modify the entire query-compilation process.

As DB2 UDB is enhanced with new features and functionality, the optimizer performance improves. An example is extending the optimizer to understand OLAP SQL extensions and multiple levels of parallel query processing. The latter is particularly important in clusters where each node is an SMP. IBM has also built into the optimizer knowledge about how to work with underlying disk sub-systems.

The optimizer takes advantage of its knowledge of the characteristics of the hardware environment: CPU speed, I/O speed, network bandwidth (for federated queries), and buffer pool allocations. In addition it knows the characteristics of the data itself, for example, the size of the table being queried, the partitioning scheme used (if any), the uniqueness of the data, the existence of indexes, and the existence of automatic summary tables. All of these are taken in account when choosing an optimal execution strategy for a given SQL statement. Such evaluations become more important as the size of a database increases.

The query optimizer can also perform query re-write operations automatically, if necessary. This feature allows DB2 UDB to generate query data access plans, which were optimized for performance, without changing the intent of the original queries or the result sets produced. This capability is especially important for environments where the SQL is being generated by a tool—which is the case for many decision support applications.

3.4 DB2 utilities

In addition to providing a rich object-relational database management system, each edition of DB2 UDB contains a comprehensive set of utilities that enable you to work with objects and data stored in DB2 UDB databases. These utilities are designed to support a wide range of traditional database configurations, as well as small OLAP and OLTP database environments. They incorporate a highly scalable software architecture that allows them to execute on a wide variety of hardware platforms.

This book does not attempt to provide a complete list of the DB2 utilities. Instead, we focus on those utilities that are most likely to be used with network attached storage (NAS) or storage area networks (SAN). For more information about DB2 UDB utilities, please refer the DB2 Information Center at the following Web site:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp>

3.4.1 Data movement utilities

Although a database is normally thought of as a single self-contained entity, there are times when it becomes necessary for a database to exchange data with the outside world. For this reason, several of the utilities provided with DB2 UDB are used to move data between databases and external files.

Export

The Export utility extracts specific portions of data from a database and externalizes it to non-delimited ASCII (ASC), delimited ASCII (DEL), Worksheet (WSF), or PC Integrated Exchange Format (PC/IXF or IXF) formatted files. These files can then be used to populate tables in a variety of databases (including the database the data was extracted from) or to provide input to software applications such as spreadsheets and word processors.

Import

The Import utility provides a way to read data directly from ASC, DEL, WSF, or PC/IXF formatted files and stores the data in a specific database table. When the Export utility is used to externalize data in a table to a PC/IXF formatted file, the table structure and definitions of all of the table's associated indexes are written to the file along with the data. Because of this, the Import utility can create or re-create a table and its indexes as well as populate the table if data is being imported from a PC/IXF formatted file. When any other file format is used, if the table or updateable view receiving the data already contains data values, the data being imported can either replace or be appended to the existing data, provided the base table receiving the data does not contain a primary key that is referenced by a foreign key of another table. In some situations, data being imported can also be used to update existing rows in a base table.

Load

Like the Import utility, the Load utility reads data directly from ASC, DEL, or PC/IXF formatted files or the new CURSOR file type (new in Version 8), and stores the data in a specific database table. The CURSOR file type allows you to load the results of a SQL query into a table without having to first export the data to a file. However, unlike when the Import utility is used, the table that the data is stored in must already exist in the database before the load operation is initiated. The Load utility ignores the table structure and index definition information stored in PC/IXF formatted files. Likewise, the Load utility does not create new indexes for a table, it only rebuilds indexes that were already defined for the table being loaded.

The most important difference between the Import utility and the Load utility relates to performance. Because the Import utility inserts data into a table one row at a time, each row inserted must be checked for constraint compliance (such as foreign key constraints and table check constraints) and all activity performed must be recorded in the database's transaction log files. The Load utility, on the other hand, inserts data into a table much faster than the Import utility because instead of inserting data into a table one row of data at a time, it builds data pages using several individual rows of data and then writes those pages directly to the tablespace container that the table's structure and any preexisting data were stored in. Existing primary/unique indexes are then rebuilt after all data pages constructed are written to the container and duplicate rows that violate primary or unique key constraints are deleted (and copied to an exception table, if appropriate).

Autoloader

In a partitioned database environment, the Autoloader utility performs the necessary steps needed to balance data being loaded from single or multiple flat files into the partitions that comprise a partitioned table. The Autoloader utility splits all data being loaded using the partition map for the table, pipes the split data to the appropriate partition, and concurrently loads each portion of the data into each partition of the partitioned table.

DB2MOVE

The DB2MOVE utility facilitates the movement of a large number of tables between DB2 databases. This utility queries the system catalog tables for a particular database and

compiles a list of all user tables found. It then exports the contents and table structure of each table found to a PC/IXF formatted file. The set of files produced can then be imported or loaded to another DB2 database on the same system, or they can be transferred to another workstation platform and imported or loaded to a DB2 database that resides on that platform. (This is the best method to use when copying or moving an existing database from one platform to another.)

The DB2MOVE utility can be run in one of three modes: EXPORT, IMPORT, or LOAD. When run in EXPORT mode, the DB2MOVE utility invokes the Export utility to extract data from one or more tables and externalize it to PC/IXF formatted files. It also creates a file named db2move.lst that contains the names of all tables processed, along with the names of the files to which that the table's data was written. When run in IMPORT mode, the DB2MOVE utility invokes the Import utility to re-create a table and its indexes from data stored in PC/IXF formatted files. In this mode, the file db2move.lst establishes a link between the PC/IXF formatted files needed and the tables into which data will be imported. When run in LOAD mode, the DB2MOVE utility invokes the Load utility to populate tables that are already created with data stored in PC/IXF formatted files. Again, the file db2move.lst is used to establish a link between the PC/IXF formatted files needed and the tables into which data will be imported.

DB2LOOK

The DB2LOOK utility is a special utility that generates the DDL SQL statements needed to re-create existing objects in a given database. In addition to generating DDL statements, DB2LOOK can also collect statistical information that was generated for objects in a database from the system catalog tables and save it (in readable format) in an external file. In fact, by using DB2LOOK, it is possible to create a clone of an existing database that contains both its data objects and current statistical information about each of those objects.

3.4.2 Data maintenance utilities

The way data stored in tables is physically distributed across tablespace containers can have a significant impact on the performance of applications that access the data. How data is stored in a table is affected by insert, update, and delete operations that are performed on the table. For example, a delete operation may leave empty pages that for some reason never get reused. Or update operations performed on variable-length columns may result in larger column values that cause an entire row to be moved to a different page because they no longer fit on the original page. In both scenarios, internal gaps are created in the underlying tablespace containers. As a consequence, the DB2 Database Manager may have to read more physical pages into memory in order to retrieve the data needed to satisfy a query.

Because situations such as these are almost unavoidable, DB2 UDB provides a set of data maintenance utilities that optimize the physical distribution of all data stored in a table.

Reorganize table (REORG)

The Reorganize Table utility removes gaps in tablespace containers by rewriting the data associated with a table to contiguous pages in storage (similar to the way a disk defragmenter works). With the help of an index, the Reorganize Table utility can also place the data rows of a table in a specific physical sequence, thereby increasing the cluster ratio of the selected index. This approach also has an attractive side effect if the DB2 Database Manager finds the data needed to satisfy a query stored in contiguous space and in the desired sort order. The overall performance of the query is improved because the seek time needed to read the data is shorter and a sort operation may no longer be necessary. The Reorg operation is a write intensive process. The N series product reduces the write overhead associated with writing to disk with NVRAM.

Run statistics (RUNSTATS)

Although the system catalog tables contain information such as the number of rows in a particular table, the way storage space is utilized by tables and indexes, and the number of different values found in a column, this information is not automatically kept up-to-date. Instead, it has to be generated periodically by running the Run Statistics utility. The information that is collected by the Run Statistics utility is used in the following two ways: to provide information about the physical organization of the data in a table and to provide information that the DB2 Optimizer can use when selecting the best path to use to access data that will satisfy the requirements of a query.

Redistribute data (REDISTRIBUTE)

The Redistribute Data utility is designed to physically move data between partitions when data is stored in a partitioned database environment. This utility is typically used to move data between partitions when a new logical data partition is added to or removed from a database partition group. It can also be used to redistribute data if the way data is distributed across existing database partitions is not uniform.

3.4.3 Other utilities

db2inidb

The db2inidb utility is used primarily to initialize a split mirror copy of a database. A mirrored copy of a DB2 database can be initialized in one of three ways:

- ▶ AS SNAPSHOT: The split mirror copy is initialized as a clone of the original database.
- ▶ AS MIRROR: The split mirror copy is initialized as a backup image of the original database and placed in roll-forward pending state.
- ▶ AS STANDBY: The split mirror copy is initialized as a standby database and placed in roll-forward pending state so it can be continuously synchronized with the original database. The standby copy of the database can then be used in place of the original database if, for some reason, the original database goes down.

The db2inidb utility is discussed in further detail in a later chapter.

db2rfpen

The db2rfpen utility places a database that was restored from an offline backup into roll-forward pending state. When a database is restored from an online backup, this happens automatically and the db2rfpen utility is not needed. (After a database is restored from a backup image, it must be placed in the roll-forward pending state before a roll-forward recovery operation can be performed against it.)

The db2rfpen utility is discussed in further detail in a later chapter.

db2relocatedb

The db2relocatedb utility allows a DBA to change the location of one or more tablespace containers or an entire database without having to perform a backup and a redirected restore operation. It also provides a way to rename a database and change the instance that a database belongs to per specifications stored in a configuration file that is provided by the user.

The db2relocatedb utility is discussed in further detail in a later chapter.

3.5 DB2 terminology and concepts

This section provides a basic overview of DB2 terminology and concepts you should understand when working with DB2 and the IBM System Storage N Series (also known simply as “N series”).

3.5.1 Installing DB2

When installing DB2 for Windows, you can choose where you want DB2 to be installed. By default, it is installed in C:\Program Files\IBM\SQLLIB. On UNIX and Linux environments, the DB2 product is installed one of the following directories:

AIX /usr/opt/db2_08_01

HP-UX, Linux, Solaris /opt/IBM/db2/V8.1

Restriction: DB2 UDB does not support installation of the DB2 product on an NFS mount (for example, NFS mounting /usr/opt/db2_08_01 7 or /opt/IBM/db2/V8.1).

3.5.2 Instances

DB2 UDB sees the world as a hierarchy of several different types of objects. Workstations on which any edition of DB2 UDB is installed are known as system objects, and they occupy the highest level of this hierarchy.

When any edition of DB2 Universal Database is installed on a particular workstation (or system), program files for the DB2 Database Manager are physically copied to a specific location on that workstation. Within this system, one or more instances of the DB2 Database Manager can be created. Instances comprise the next level in the object hierarchy. Each instance references the DB2 Database Manager program files that were stored on that workstation during the installation process. Thus, each instance behaves like a separate installation of DB2 Universal Database, even though all instances within a particular system share the same binary code. Although all instances share the same physical code, each can be run concurrently. The important thing to note is that each instance has its own environment and settings separate from other instances. This allows for one system to have multiple database environments. For example, on a single system one instance can be used as a development environment, while a second instance can be used as a testing environment. This allows testing to be performed without affecting the development environment.

N series accommodates the instance design by providing the ability to assign aggregates and FlexVols without regards to standard sizes. This enables you to increase storage utilization (Figure 3-1 on page 74).

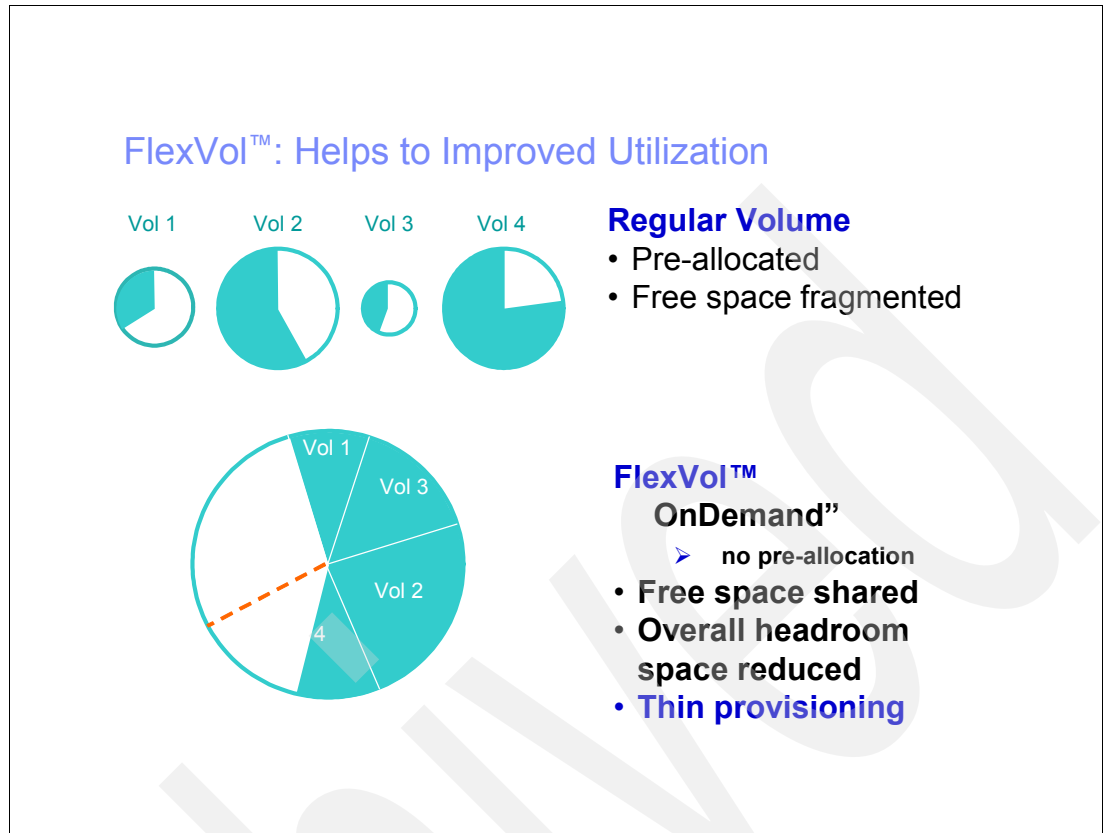


Figure 3-1 FlexVol utilization

3.5.3 Databases

In its simplest form, a DB2 Universal Database database is a set of related database objects. When you create a DB2 database, you are establishing an administrative relational database entity that provides an underlying structure for an eventual collection of database objects (such as tables, views, indexes, and so on). This underlying structure consists of a set of system catalog tables (along with a set of corresponding views), a set of table spaces in which both the system catalog tables and the eventual collection of database objects will reside, and a set of files that handle database recovery and other bookkeeping details.

DB2 UDB allows multiple databases to be defined within a single database instance. Each database has its own configuration file, which allows characteristics of the database, such as memory usage and logging, to be fine tuned for optimum performance.

In Figure 3-2 on page 75, we can see the relationship between a database server, instances, and databases. Two instances (DB2_DEV and DB2_PROD) exist on one server (DB_SERVER), and two databases (PAYABLE and RECEIVABLE) exist within each instance. Both instances share the same binary code; however, each has its own configuration file. Likewise, each database within each instance has its own configuration file.

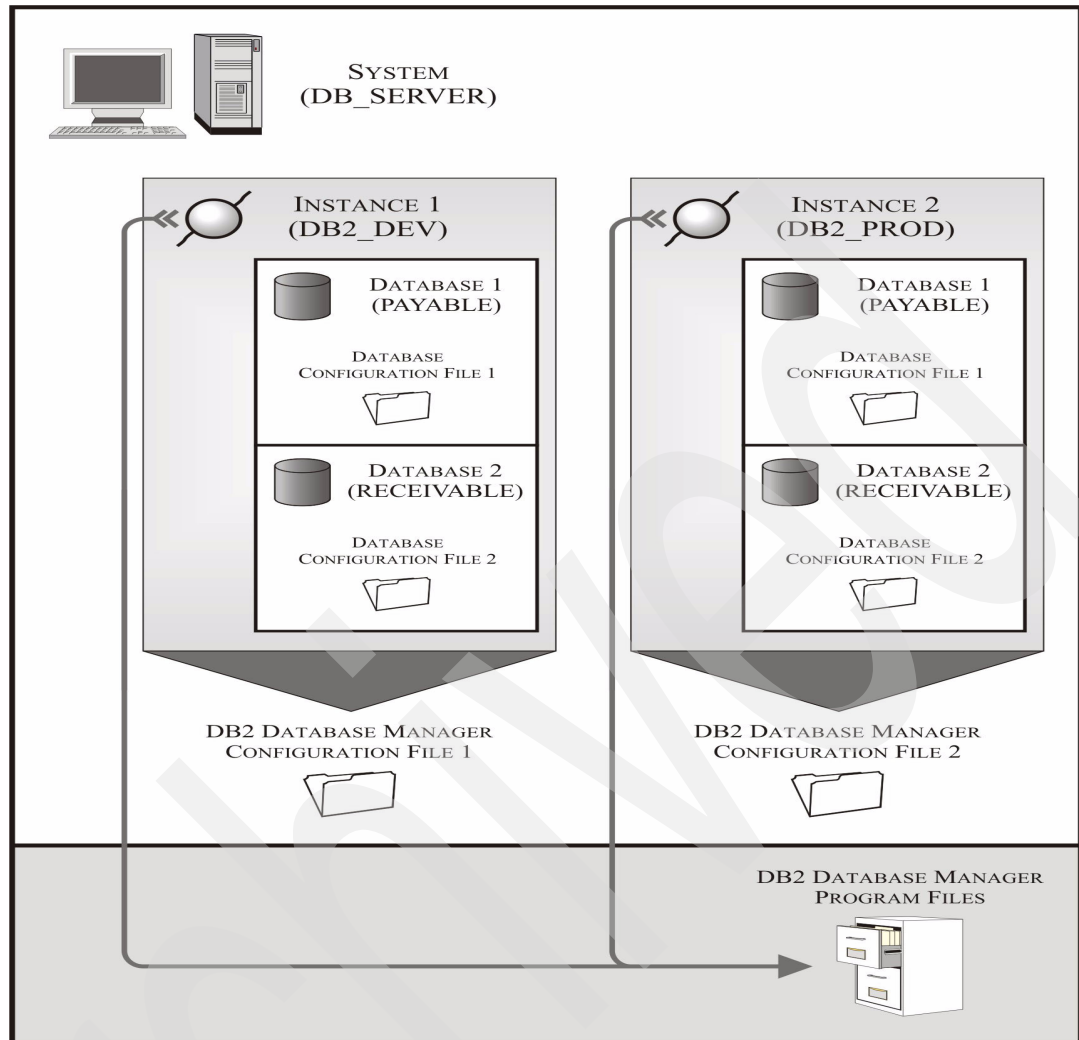


Figure 3-2 The relationship between database servers, instances, and databases.

3.5.4 Buffer pools

A buffer pool is an area of main memory allocated to the DB2 Database Manager for the purpose of caching table and index data pages as they are read from disk or modified. Using a set of heuristic algorithms, the DB2 Manager Database prefetches pages of data that it thinks a user is about to need into one or more buffer pools, and it moves pages of data that it thinks are no longer needed back to disk. This approach improves overall system performance because data can be accessed much faster from memory than from disk, especially if SATA (Serial Advanced Technology Attachment) disks are used. The less often the DB2 Database Manager needs to perform disk I/O, the better the performance.

Each time a new database is created, one default buffer pool, named **IBMDEFAULTBP**, is also created as part of the database initialization process. The size of this default buffer pool is dependent on the platform and the page size specified at the time of database creation.

Prefetchers and page cleaners

DB2 employs the use of physical processes or threads, called agents, to handle the reading and writing of data between disk and buffer pools.

Prefetcher agents transfer data from disk into the buffer pool, anticipating their need by an application. Prefetching helps improve performance by reducing the time spent waiting for disk I/O to complete.

To prevent a buffer pool from becoming full, page cleaner agents are used to write modified pages (known as "dirty" pages) to disk at a predetermined interval (by default, when the buffer pool is 60 percent full) to guarantee the availability of buffer pool pages for future read operations.

3.5.5 Table spaces

When setting up a new database, one of the first tasks that must be performed is to map the logical database design to physical storage on a system. This is where table spaces come into play. Table spaces control where data in a database is physically stored on a system and provide a layer of indirection between the database and the container objects in which the actual data resides.

When a database is first created, the following three table spaces are also created and associated with the default buffer pool IBMDEFAULTBP as part of the database initialization process:

SYSCATSPACE	Stores the system catalog tables and views associated with the database.
USERSPACE1	Stores all user-defined objects (such as tables, indexes, and so on) along with user data.
TEMPSPACE1	Stores temporary tables that might be created in order to resolve a query.

Additional table spaces can be created as needed. Table spaces are managed in two different ways: system managed and database managed.

With *system managed space (SMS)* table spaces, the operating system's file manager is responsible for allocating and managing the storage space used by the tablespace. SMS table spaces consist of one or more directories containing individual files (representing data objects such as tables and indexes) that are stored in the file system.

With *database managed space (DMS)* table spaces, the tablespace creator (and in some cases, the DB2 Database Manager) is responsible for allocating the storage space used and the DB2 Database Manager is responsible for managing it. Essentially, a DMS tablespace is an implementation of a special-purpose file system that was designed specifically to meet the needs of the DB2 Database Manager.

Regardless of how they are managed, three types of table spaces exist: regular, temporary, and large. You can use regular table spaces for any type of data except temporary tables. To store long or large object (LOB) data such as images, audio, video, or long text fields, you use a large tablespace. Large table spaces must be a DMS tablespace. Temporary table spaces store declared global temporary tables.

3.5.6 Containers

A *container* is an allocation of physical storage to which the DB2 Database Manager is given exclusive access. Containers essentially provide a way of defining what location on a specific storage device is made available for storing database objects. Each tablespace must contain at least one container.

Containers may be assigned from file systems by specifying a directory (known as *path* containers), file (known as *file* containers), or raw devices (known as *device* containers). A single tablespace can span many containers, but each container can only belong to one tablespace. Figure 3-3 illustrates the relationship between buffer pools, table spaces, and containers.

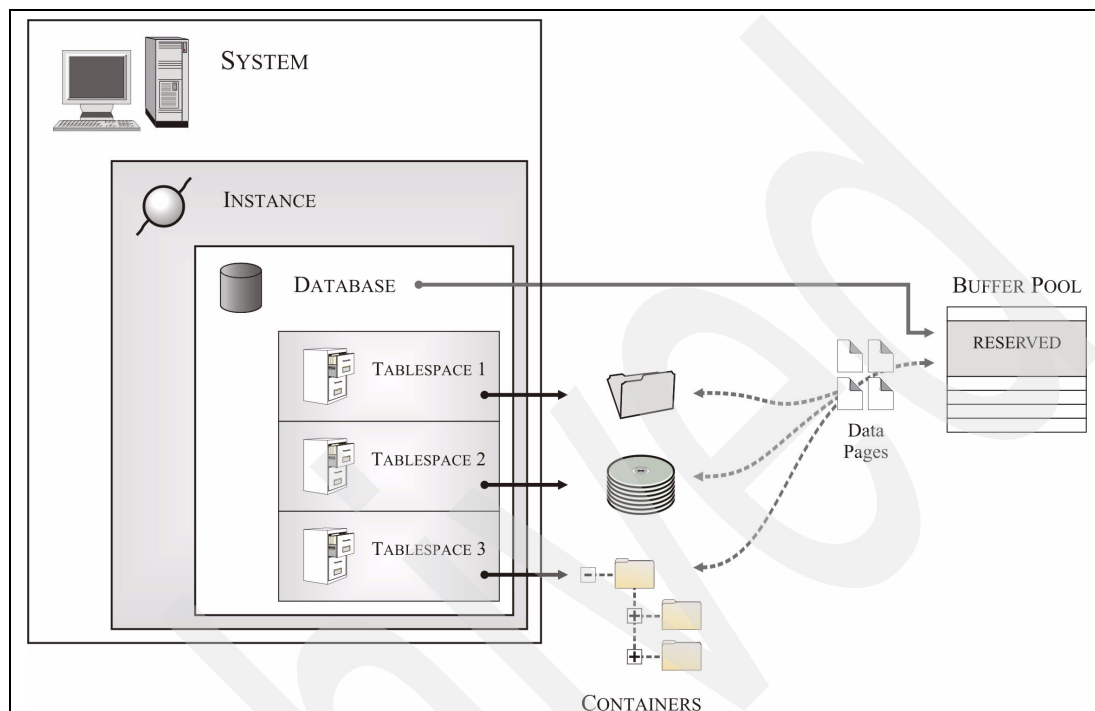


Figure 3-3 The relationship between bufferpools, tablespaces, and containers.

It is important to note that SMS table spaces can reference *file*, *device*, and *path* containers; whereas, DMS table spaces can only reference *file* and *device* containers.

Furthermore, in DB2 and N series product environments configured to use NFS (network file system) and CIFS (common internet file system), only *file* and *path* containers are supported. In DB2 and N series product environments configured to use iSCSI (internet small computer system interface) or FCP (fibre-channel protocol), all three types of containers are supported.

3.5.7 Characteristics that affect tablespace performance

In addition to how a tablespace is managed, the following three other tablespace characteristics must be taken into consideration in order to design a database for optimum performance:

- ▶ Page size
- ▶ Extent size
- ▶ Prefetch size

Page size

With DB2 UDB, data is transferred between tablespace containers and buffer pools in discrete blocks that are called pages. (The memory reserved to buffer a page transfer is called an I/O buffer.) The actual page size used by a particular tablespace is determined by the page size of the buffer pool that the tablespace is associated with. Four different page

sizes are available: 4K, 8K, 16K, and 32K. By default, all table spaces that are created as part of the database creation process are assigned a 4K page size.

Extent size

When a tablespace spans multiple containers, data is stored on all of its respective containers in a round-robin fashion. The extent size of a tablespace represents the number of pages of table data that are to be written to one container before moving to the next container in the list. This helps balance data across all containers that belong to a given tablespace (assuming all extent sizes specified are equal). Figure 3-4 illustrates how extents are used to balance data across multiple containers.

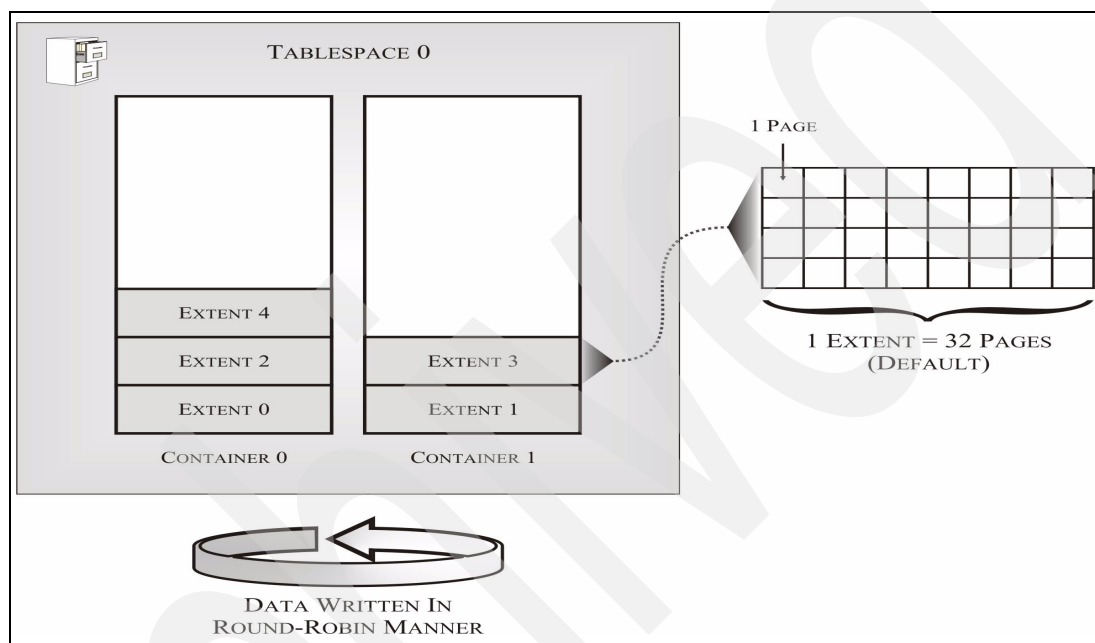


Figure 3-4 Illustration of how extents are used to balance data across multiple containers

Prefetch size

The prefetch size of a tablespace identifies the number of pages of table data that are to be read in advance of the pages currently being referenced by a query. This is in anticipation that they will be needed to resolve the query. The overall objective of prefetching is to reduce the time waiting for disk I/O to complete.

The recommended value for the prefetch size is a multiple of all three of the following:

- ▶ The number of tablespace containers.
- ▶ The number of physical disks under each container (if a RAID device is used). N series uses RAID 4 and RAID-DP so it is the number of physical disks per volume or aggregate.
- ▶ The extent size for the tablespace

For example, if the extent size is 16 pages and the tablespace has two containers, you could set the prefetch size to 32 pages. If there are five physical disks per container, then you might set the prefetch size to 160. The proper setting of this parameter promotes the use of parallel I/O, which we discuss later on in this chapter.

3.5.8 Registry and performance-related environment variables

In addition to DB2 Database Manager and database configuration parameters, DB2 utilizes several registry and environment variables to configure the system where it is installed. Because changes made to registry and environment variables impact the entire system, changes should be carefully considered before they are made. After setting any registry variable, the DB2 Database Manager must be stopped (db2stop), and then restarted (db2start) for the changes to take effect.

Set the following registry variable when working with N series:

- DB2_PARALLEL_IO

We discuss this variable in further detail in Chapter 13, “DB2 and N series performance considerations” on page 273.

3.5.9 Backup and recovery using N series

One of the traditional tools used to prevent catastrophic data losses when media failures occur is a backup image. A backup image is essentially a copy of an entire database or of one or more of the table spaces that make up a database. After created, a backup image can be used at any time to return a database to the state it was in at the time the image was made. Using the traditional backup method generates a significant load on both the database server and the storage system and can take hours to complete. During this time, system availability and performance are decreased. In such environments, backups are scheduled during off-peak hours and frequent backups are not feasible.

However, when using N series to store your database, backups can be created within a few seconds and have virtually no impact on the database server or the storage system. Also, restoring a database from a Snapshot backup of your data and logs is much faster than restoring from conventional backups. This is possible through the use of Snapshot and SnapRestore Technology available on the N series product. (Figure 3-5 on page 80).

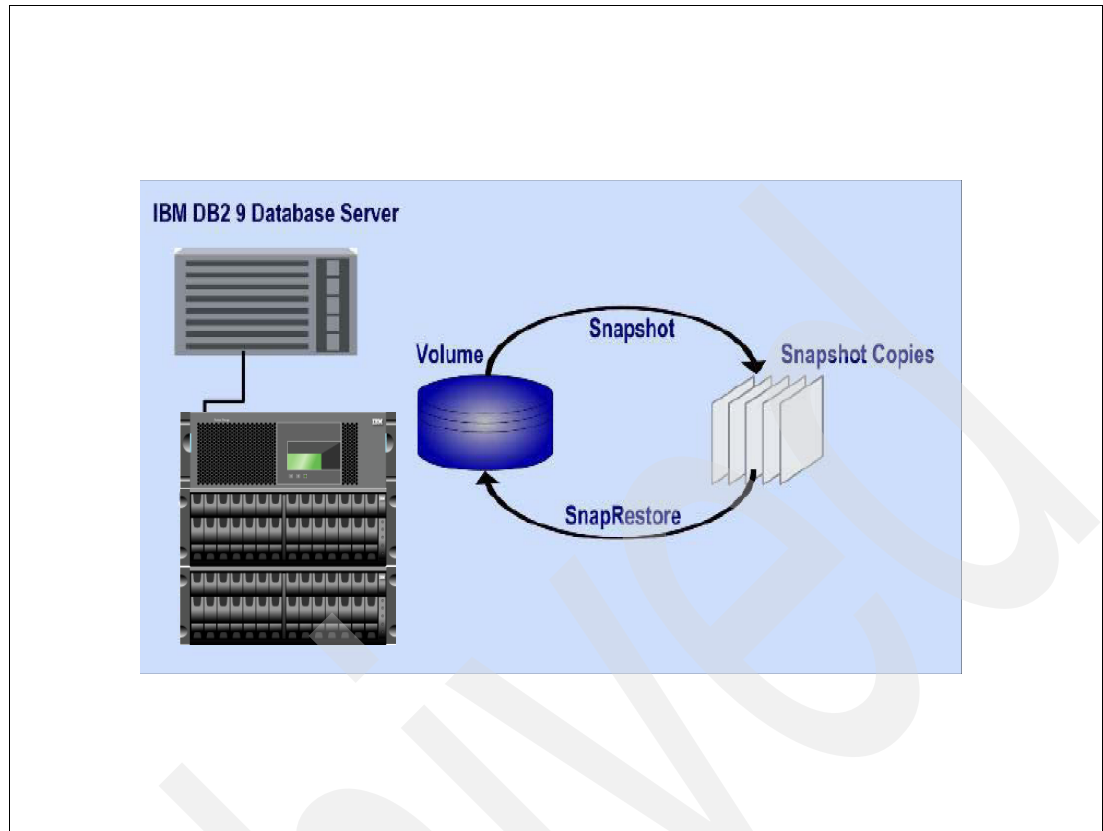


Figure 3-5 Database backup

3.5.10 Recovery logs

Recovery logs are files that keep track of changes that are made to a database. Each time a new record is inserted into a base table, that record is first created directly in the buffer pool associated with the tablespace that identifies where the table's data will be stored. Each time a record is updated or deleted, the page containing the record is retrieved from storage and copied to the appropriate buffer pool, where it is then modified by the update and delete operation being performed. After this modification is made, a record reflecting the action is written to the log buffer, which is another designated storage area in memory. (The actual amount of memory reserved for the log buffer is controlled by the *logbufsiz* database configuration parameter.)

- ▶ If an insert operation was performed, a record containing the new row's data values is written.
- ▶ If a delete operation is performed, a record containing the row's original values is written.
- ▶ If an update operation is performed, a record containing the row's original values combined with the row's new values is written. (In most cases, the log record for an update operation is produced by performing an EXCLUSIVE OR operation on the row's original values with the row's updated values.)

Eventually, when the transaction that performed the insert, update, or delete operation is terminated, a corresponding COMMIT or ROLLBACK record is written to the log buffer as well.

Whenever buffer pool I/O page cleaners are activated, the log buffer itself becomes full, or if a transaction is committed or rolled back, all records stored in the log buffer are immediately written to one or more transaction log files stored on disk. This constant flushing of the log

buffer minimizes the number of log records that can be lost if a system failure occurs. As soon as all log records associated with a particular transaction (including a corresponding COMMIT or ROLLBACK record) are successfully externalized to one or more log files, the effects of the transaction itself are copied to appropriate tablespace containers for permanent storage. (The modified data pages themselves remain in memory, where they can be quickly accessed if necessary. Eventually they are overwritten.) This process, called write-ahead logging, guarantees that changes made to data are always externalized to log files before they are recorded in the database. The transaction logging process just described is outlined in Figure 3-6.

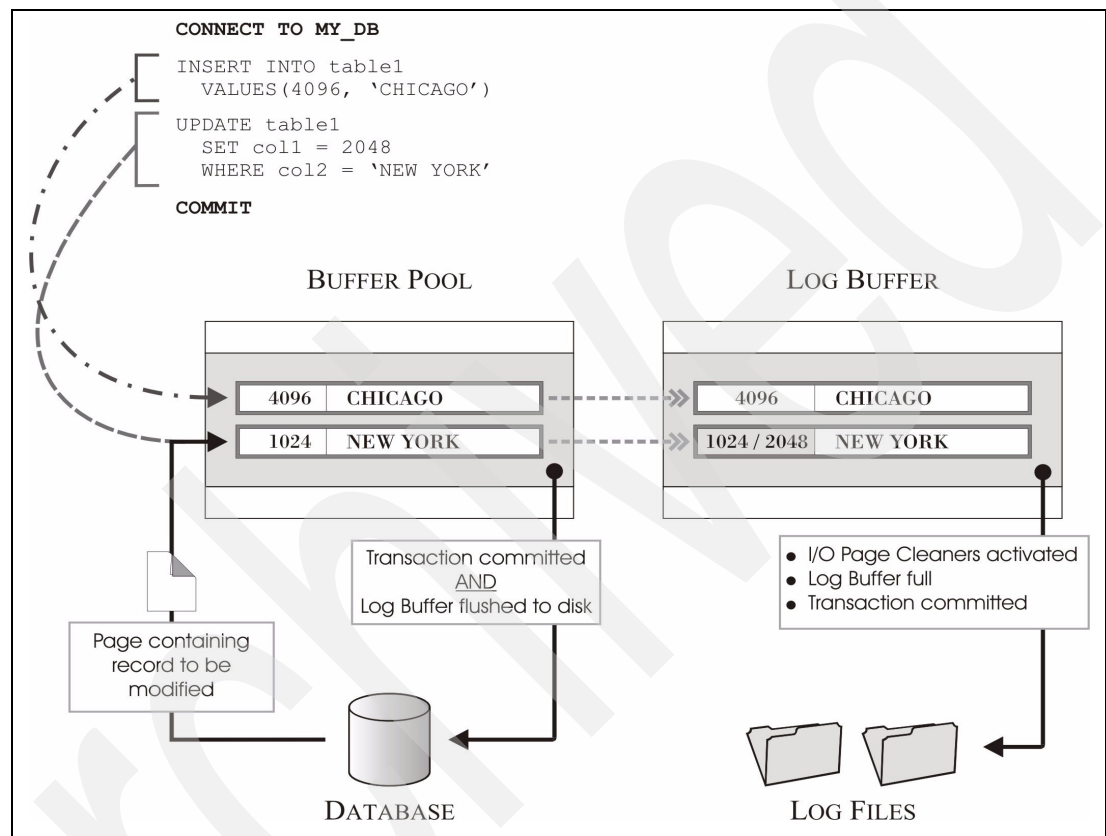


Figure 3-6 How Transaction Logging Works

Because multiple transactions may be working with a database at any point, a single log file may contain log records that belong to several different transactions. In order to keep track of which log records belong to which transaction, every log record is assigned a special *transaction identifier* that ties it to the transaction that created it. By using transaction IDs, log records associated with a particular transaction can be written to one or more log files at any time without impacting data consistency. Eventually the COMMIT or ROLLBACK record for the operation that terminated the transaction is logged as well.

When a database is first created, three log files (known as primary log files) are allocated. However, the number of primary log files used and the amount of data each log file is capable of storing is controlled by the values assigned to the *logprimary* and *logfilsiz* parameters in the database's configuration file. Furthermore, the way in which primary log files are created and used for a particular database is determined by the logging strategy chosen for that database. Two very different strategies are available: *circular* and *archival* logging.

Circular logging

With circular logging, a group of log files are defined and used to provide transaction logging support. Records are written to the log files in a round-robin fashion as transactions are processed and records are removed from a log file when the associated transactions are either committed and externalized to disk or rolled back.

When using circular logging, only full, offline backup images of the database are taken and roll-forward recovery cannot be performed. When a database is placed in an inconsistent state, it can be returned to a consistent state by utilizing the records stored in the active log to resolve any in-doubt or in-flight transactions. However, if a database becomes damaged or destroyed, it can only be salvaged by using the `RESTORE DATABASE` command in conjunction with a full, offline backup image that was taken earlier. Any changes made to the database after the backup image was taken are lost.

Archival logging

Archive logging is used specifically for rollforward recovery. Following is how archive logging works: DB2 starts out with an active log where transaction log records are stored. After the active log file becomes full, a new active log file is created in the database's log directory, and the current active log file becomes an archive log file. These logs can then be stored in a safe place in the event that rollforward recovery is required.

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to rebuild a database either to the end of the logs or to a specific point in time. The archived log files can be used to recover changes made after the backup was taken. This is different from circular logging where you can only recover to the time of the backup, and all changes made after that are lost.

Taking online backups is only supported if the database is configured for archive logging. During an online backup operation, all activities against the database continue to be logged. When an online backup image is restored, the logs must be rolled forward at least to the point in time at which the backup operation completed. For this to happen, the logs must be archived and made available to DB2 when the database is restored.

After an online backup is complete, DB2 forces the currently active log to be closed, and as a result, it is archived. This ensures that your online backup has a complete set of archived logs available for recovery. Obviously, the more archive log files you have online, the faster the recovery process will be. Because every change to a row includes the before-and-after image of the row, online archive log files can potentially become quite large. Thus, a N series product is an excellent location for these objects, as well as for the database itself because of its dynamic ability to increase volume sizes.

3.5.11 The recovery history file

The recovery history file contains certain historical information about major actions that were performed against a database. Information recorded in the recovery history file is used to assist with the recovery of the database in the event of a failure. The following is a list of the actions that generate an entry in the recovery history file:

- ▶ Backing up the database or a tablespace
- ▶ Restoring the database or a tablespace from a backup image
- ▶ Performing a roll-forward recovery operation on the database or a tablespace
- ▶ Loading a table
- ▶ Altering a tablespace's definition

- ▶ Quiescing a tablespace
- ▶ Reorganizing a table (REORG)
- ▶ Updating statistics for a table (RUNSTATS)
- ▶ Dropping a table

The recovery history file can be queried to obtain information such as when a database was last backed up and when a database was last restored. Taking a N series product Snapshot of this file is a low cost guarantee that this file will be available when needed.

Archived

Introduction to the Oracle Database

This chapter describes the following:

- ▶ The different physical and logical structures for Oracle
- ▶ The processes used to access and manipulate the databases
- ▶ Integration of Oracle with the IBM System Storage N series

The intended audience for this chapter is storage administrators and others who are unfamiliar with the Oracle database.

4.1 Oracle Database architecture

An Oracle server is a database management system that provides an open, comprehensive, integrated approach to information management. It consists of an Oracle instance and an Oracle Database. An Oracle instance is the combination of all of the memory structures and background processes that are allocated when an Oracle Database is started. Oracle users frequently confuse an Oracle instance with an Oracle Database.

An Oracle Database has both physical structures (data files) and logical structures (table, index). Oracle separates the physical from the logical structure

so that the physical storage of data can be manipulated without affecting user access to the logical structures. The physical structure of an Oracle Database is determined by the files created at the operating system level by the Oracle instance during database creation (controlfile, logfile) or by the DBA during normal operation (create table space, create control file). In later chapters we focus on the physical structure of Oracle on the IBM System Storage N series.

Oracle also uses several memory and process structures to access and manage the database. The memory structures hold executing program code and information shared and communicated among Oracle processes as well as the actual data from the underlying physical storage structures. The various Oracle processes perform such tasks as interfacing with user application programs and monitoring the database for availability and performance.

4.1.1 Memory structures

Oracle utilizes several different types of memory structures for storing and retrieving data in the system. These include the *System Global Area (SGA)* and *Program Global Areas (PGA)*. Figure 4-1 depicts these memory structures and the relationship between them.

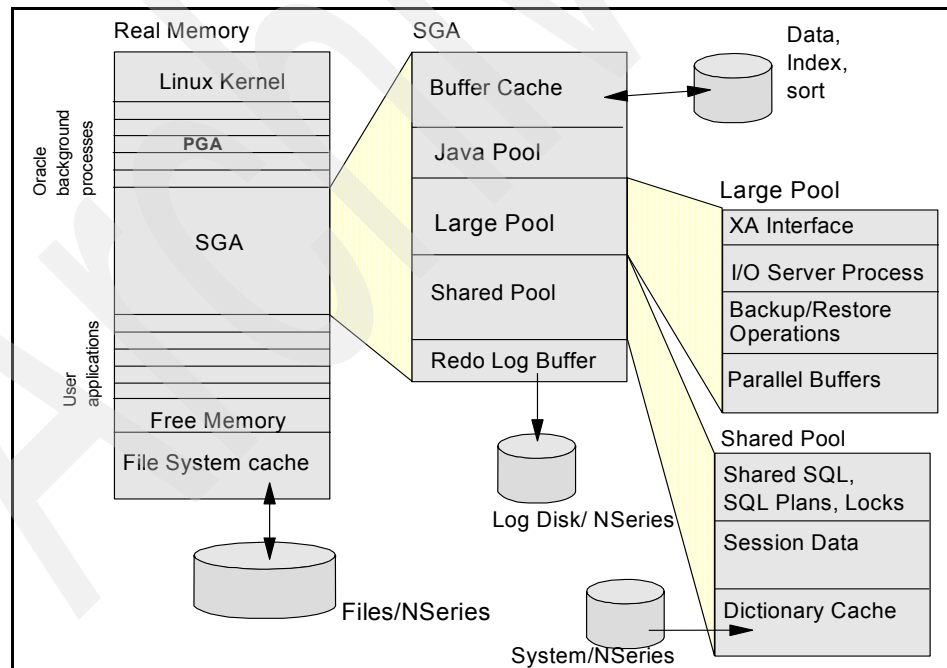


Figure 4-1 Oracle memory structures

The Oracle SGA is a shared memory region that holds data and internal control structures of the database. This shared memory region's details can be displayed in Linux with the `ipcs -ma` command. The Oracle SGA and its associated background processes, described in section 4.1.4, "Processes" on page 92, are known as an Oracle *instance*. The instance identity is called by the short name *SID*. This short name is used in all Oracle processes connected to this instance. To connect to a particular instance, set the shell variable `$ORACLE_SID` to the *SID* or specify it in the connect command at SQLPlus. The SGA memory region is allocated upon instance startup and de-allocated when the instance is shut down and is unique to each database instance. SGA is dynamically managed by Oracle, while the instance is up. The information contained in the SGA is logically separated into three-to-five different areas: The *database buffer cache*, the *redo log buffer*, the *shared pool*, the *Java pool* (optional) and the *large pool*.

The database buffer cache consists of Oracle Database data blocks or *buffers* that are read from disk and placed into memory. These buffers are classified as either *free*, *dirty*, or *pinned* and are organized in two lists: the *write list* and the *LRU list*. Free buffers are those that have not yet been modified and are available for use. Dirty buffers contain data that was modified but has not yet been written to disk and are held in the write list. Lastly, pinned buffers are buffers that are currently being accessed. We will discuss in later chapters how IBM System Storage N series works with these buffers.

Oracle uses a Least Recently Used (LRU) algorithm to age the dirty data blocks from memory to disk. An LRU list keeps track of all available buffers and their state (dirty, free, or pinned). Infrequently accessed data is moved to the end of the LRU list where it is eventually written to disk by the Oracle DBWn process (see section 4.1.4, "Processes" on page 92) should an additional free buffer be requested but not available.

The size of the database buffer cache is determined by a combination of some Oracle initialization parameters. It is possible to set different block sizes for a database by specifying different cache sizes for each of these nonstandard block sizes. The `DB_BLOCK_SIZE` parameter specifies the default or standard block size and the `DB_CACHE_SIZE` parameter sets the size of the cache using the blocks of `DB_BLOCK_SIZE`. To specify different block sizes, use the initialization parameter `DB_NK_CACHE_SIZE`, where *NK* ranges from 2 KB to 32 KB. Also, the DBA can change these parameters while the instance is up and running, except for the `DB_BLOCK_SIZE` parameter, which requires the database to be recreated.

The Java pool holds Java execution code and classes information if the Java option is turned on to a specific Oracle instance.

The redo log buffer stores information about changes made to data in the database. This information can be used to reapply or *redo* the changes made to the database should a database recovery become necessary. The entries in the redo log buffer are written to the online redo logs by the LGWR process (see section 4.1.4, “Processes” on page 92). The size of the redo log buffer is determined by the LOG_BUFFER parameter in the Oracle initialization file.

The shared pool area stores memory structures, such as the shared SQL areas, private SQL areas, and the data dictionary cache, and, if large pool is not turned on, the buffers for parallel execution messages. Shared SQL areas contain the parse tree and execution plan for SQL statements. Identical SQL statements share execution plans. One memory region can be shared for multiple identical Data Manipulation Language (DML) statements, thus saving memory. DML statements are SQL statements that are used to query and manipulate data stored in the database, such as SELECT, UPDATE, INSERT, and DELETE.

Private SQL areas contain Oracle bind information and run-time buffers. The bind information contains the actual data values of user variables contained in the SQL query.

The data dictionary cache holds information pertaining to the Oracle data dictionary. The Oracle data dictionary serves as a roadmap to the structure and layout of the database. The information contained in the data dictionary is used during Oracle’s parsing of SQL statements.

The buffers for parallel execution hold information needed to synchronize all the parallel operations in the database. This is allocated from SGA, if the large pool is not configured.

The large pool area is an optional memory that can be used to address shared memory contention. It holds information such as the Oracle XA interface (the interface that Oracle uses to enable distributed transactions), I/O server processes, backup and restore operations, and, if the initialization parameter PARALLEL_AUTOMATIC_TUNING is set to TRUE, the buffer for parallel execution. Large pool is enabled by setting the LARGE_POOL_SIZE to a number greater than 0.

Starting with Oracle 9i, a new feature called *Dynamic SGA* allows optimized memory usage by instance. Also, it allows increasing or decreasing of Oracle’s use of physical memory, with no downtime, because the database administrator may issue ALTER SYSTEM commands to change the SGA size, while the instance is running.

The Oracle PGA is a nonshared memory region that contains data and control information for a server process. This memory area is allocated by Oracle every time a server process is started, and access is allowed only to Oracle code

working on it. In general, PGA contains a private SQL area (storing bind information and run-time structures that are associated with a shared SQL area) and a session memory that holds session specific variables, such as logon information. If Oracle is using shared servers, this memory area is shared.

The run-time area of PGA can also be used as a work area for complex queries making use of memory-intensive operators like the following ones:

- ▶ Sort operations, like ORDER BY, GROUP BY, ROLLUP, and WINDOW
- ▶ HASH JOINS
- ▶ BITMAP MERGE
- ▶ BITMAP CREATE

If the session connects to a dedicated Oracle server, PGA is automatically managed. The database administrator just needs to set the PGA_AGGREGATE_TARGET initialization parameter to the maximum amount of memory he wants Oracle to use for client processes. Oracle ensures that the total PGA allocated to all processes never exceeds this value.

4.1.2 Logical storage structures

An Oracle Database is made up of several logical storage structures, including *data blocks*, *extents* and *segments*, *tablespaces*, and *schema objects*.

The actual physical storage space in the datafiles is logically allocated and deallocated in the form of Oracle data blocks. Data blocks are the smallest unit of I/O in an Oracle Database. Oracle reserves a portion of each block for maintaining information, such as the address of all the rows contained in the block and the type of information stored in the block. This overhead is normally in the range of 84 to 107 bytes.

An extent is a collection of contiguous data blocks. A table is comprised of one or more extents. The very first extent of a table is known as the *initial extent*. When the data blocks of the initial extent become full, Oracle allocates an *incremental extent*. The incremental extent does not have to be the same size (in bytes) as the initial extent.

A segment is the collection of extents that contain all of the data for a particular logical storage structure in a tablespace, such as a table or index. There are four different types of segments, each corresponding to a specific logical storage structure type:

- ▶ Data segments
- ▶ Index segments

- ▶ Rollback segments
- ▶ Temporary segments

Data segments store all the data contained in a table, partition, or cluster. Likewise, index segments store all the data contained in an index. For backward compatibility, rollback segments hold the previous contents of an Oracle data block prior to any change made by a particular transaction. If any part of the transaction should not complete successfully, the information contained in the rollback segments is used to restore the data to its previous state. From Oracle9i onwards, this is achieved by using *automatic undo* and *undo tablespaces*, which allow better control and use of the server resources.

Rollback segments are also used to provide *read-consistency*. There are two different types of read-consistency: *statement-level* and *transaction-level*. Statement-level read consistency ensures that all of the data returned by an individual query comes from a specific point in time: the point at which the query started. This guarantees that the query does not see changes to the data made by other transactions that have committed since the query began. This is the default level of read-consistency provided by Oracle.

In addition, Oracle offers the option of enforcing transaction-level read consistency. Transaction-level read consistency ensures that all queries made within the same transaction do not see changes made by queries outside of that transaction but can see changes made within the transaction itself. These are known as *serializable* transactions.

Temporary segments are used as temporary workspaces during intermediate stages of a query's execution. They are typically used for sort operations that cannot be performed in memory. The following types of queries may require a temporary segment:

- ▶ SELECT.....ORDER BY
- ▶ SELECT.....GROUP BY
- ▶ SELECT.....UNION
- ▶ SELECT.....INTERSECT
- ▶ SELECT.....MINUS
- ▶ SELECT DISTINCT.....
- ▶ CREATE INDEX....

Tablespaces group related logical entities or objects together in order to simplify physical management of the database. Tablespaces are the primary means of allocating and distributing database data at the physical disk level. Tablespaces are used to do the following:

- ▶ Control the physical disk space allocation for the database
- ▶ Control the availability of the data by taking the tablespaces online or off-line
- ▶ Distribute database objects across different physical storage devices to improve performance
- ▶ Regulate space for individual database users

Every Oracle Database contains at least one tablespace named SYSTEM. The SYSTEM tablespace contains the data dictionary tables for the database used to describe its structure.

Schema objects are the logical structures used to refer to the database's data. A few examples of schema objects are tables, indexes, views, and stored procedures. Schema objects, and the relationships between them, constitute the relational design of a database.

4.1.3 Physical storage structures

An Oracle Database is made up of three different types of physical database files: *datafiles*, *redo logs*, and *control files*.

An Oracle Database must have one or more datafiles in order to operate. Datafiles contain the actual database data logically represented in the form of tables or indexes. At the operating system level, datafiles can be implemented as either JFS files or raw devices. These JFS files or raw devices may be provided by the IBM System Storage N series. The data contained in the datafiles is read from disk into the memory regions, as described in section 4.1.1, "Memory structures" on page 86.

An Oracle tablespace is comprised of one or more datafiles. A datafile cannot be associated with more than one tablespace, nor can it be used by more than one database. At creation time, the physical disk space associated with a datafile is pre-formatted but does not contain any user data. As data is loaded into the system, Oracle reserves space for data or indexes in the datafile in the form of extents.

Redo logs are used by Oracle to record all changes made to the database. Every Oracle Database must have at least two redo logs in order to function. The redo log files are written to in a circular fashion. When the current online log fills up, Oracle begins writing to the next available online redo log. In the event of a failure, changes to the Oracle Database can be reconstructed using the

information contained in the redo logs. Due to their importance, Oracle provides a facility for mirroring or *multiplexing* the redo logs so that two (or more) copies of the log are available on disk.

Note: If you are performing a full, offline backup, SnapManager backs up the redo logs. SnapManager always backs up the archive logs and the control files.

The control file describes the physical structure of the database. It contains information, such as the database name, date, and time the database was created, as well as the names and locations of all the database data files and redo logs. Like the redo logs, Oracle can have multiple copies of the control file to protect against logical or physical corruption. In a later chapter we discuss the optimal configuration and layout of these files on an IBM System Storage N series. Snapshot can also be used to make copies of the control files.

4.1.4 Processes

A process is defined as a *thread of control* used in an operating system to execute a particular task or series of tasks. Oracle utilizes the following three different types of processes to accomplish these tasks:

- ▶ User or client processes
- ▶ Server processes
- ▶ Background processes

User processes are created to execute the code of a client application program. The user process is responsible for managing the communication with the Oracle server process using a *session*. A session is a specific connection of a user application program to an Oracle instance. The session lasts from the time that the user or application connects to the database until the time the user disconnects from the database.

Figure 4-2 on page 93 shows the various pieces and relationships of the Oracle process architecture.

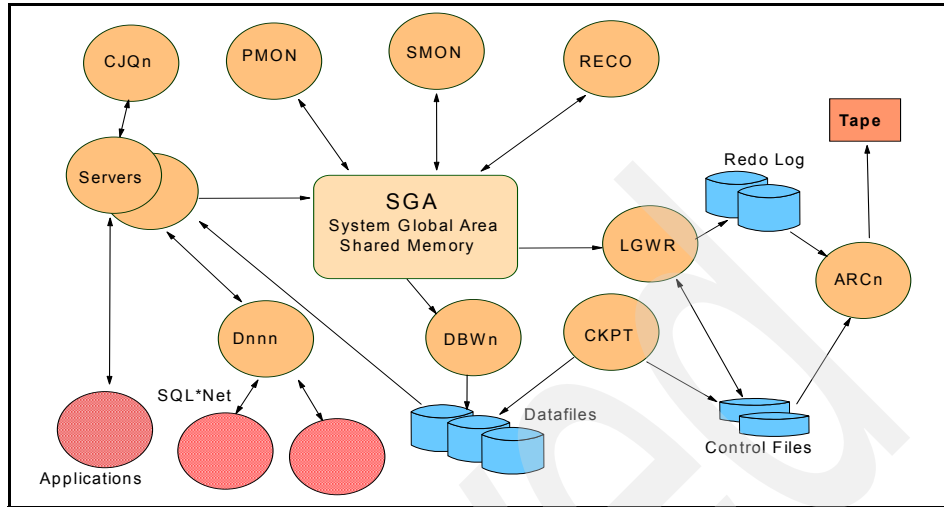


Figure 4-2 Oracle process architecture

Server processes are created by Oracle to service requests from connected user processes. They are responsible for interfacing with the database to carry out the requests of user processes. The number of user processes per server process is dependent on the configuration of Oracle. In a *dedicated server* configuration, one server process is spawned for each connected user process. In a *multi-threaded server* configuration, user processes are distributed among a predefined number of server processes.

Oracle background processes are created upon database startup or initialization. Some background processes are necessary for normal operation of the system, while others are only used to perform certain database maintenance or recovery related functions. The Oracle background processes includes the following:

- Database Writer (DBWn)

The database writer process writes modified or *dirty* database buffers from the database buffer cache to disk. It uses a least recently used (LRU) algorithm to ensure that the user processes always finds free buffers in the database buffer cache. Dirty buffers are written to disk using a single multi-block write. Additional database writer processes can be configured to improve write performance if necessary. On Linux, enabling asynchronous I/O should yield better performance. Refer to the following Web site for more information:

<http://www.redhat.com/magazine/013nov05/features/oracle>

Refer to 16.3, “Deploying databases with IBM System Storage N series and NFS” on page 321 for parameter setting examples. If higher write

performance is required, Fibre Channel drives should be used instead of SATA drives on the N series product.

Note: Multiple database writer processes provide no performance benefit on uniprocessor based systems.

► Log Writer (LGWR)

The log writer process writes modified entries from the redo log buffer to the online redo log files on disk. This occurs when one of the following conditions is met:

- Three seconds have elapsed since the last buffer write to disk.
- The redo log buffer is one-third full.
- The DBWn process has written modified buffers to disk.
- A transaction commits.

A *commit record* is placed in the redo log buffer when a user issues a COMMIT statement, at which point the buffer is immediately written to disk. The commit record serves as a reminder to the LGWR process that the redo entries associated with this particular transaction were already written to disk. The actual modified database data blocks are written to disk at a later time, which is a technique known as *fast commit*. The committed transaction is assigned a *system change number* (SCN), which is recorded in the redo log in order to uniquely identify the changes made within the transaction.

► Checkpoint Process (CKPT)

The checkpoint process notifies the DBWn process that the modified database blocks in the SGA need to be written to the physical datafiles. It is also responsible for updating the headers of all Oracle datafiles and control files to record the occurrence of the most recent checkpoint.

► Archiver (ARCn)

The archiver process copies the online redo log files to an alternate physical storage location after they become full. The ARCH process exists only when the database is configured for ARCHIVELOG mode and automatic archiving is enabled.

► System Monitor (SMON)

The system monitor process recovers the Oracle instance upon startup. It is also responsible for performing various other administrative functions, such as cleaning up temporary segments that are no longer in use and coalescing free extents in dictionary managed tablespaces.

- ▶ **Process Monitor (PMON)**

The process monitor cleans up after failed user processes. This includes such tasks as removing entries from the process list, releasing locks, and freeing up used blocks in the database buffer cache associated with the failed process. It also starts dispatcher or server processes that unexpectedly finished.

- ▶ **Recover (RECO)**

The recover process is responsible for recovering all in-doubt transactions that were initiated in a distributed database environment. RECO contacts all other databases involved in the transaction to remove any references associated with that particular transaction from the pending transaction table. The RECO process is not present at instance startup unless the `DISTRIBUTED_TRANSACTIONS` parameter is set to a value greater than zero and distributed transactions are allowed.

- ▶ **Dispatcher (Dnnn)**

Dispatcher processes are only present in a multi-threaded server configuration. They allow multiple user processes to share one or more server processes. A client connection request is received by a network listener process, which, in turn, passes the request to an available dispatcher process, who then routes the request to an available server process. If no dispatcher processes are available, the listener process starts a new dedicated server process and connects the user process directly to it.

- ▶ **LOCK (LCKn)**

The lock process is used in Oracle Parallel Server configurations to ensure inter-instance resource management. As of Oracle 8, up to ten lock processes can be started. In Oracle9i Real Application Cluster, the Lock Process is called Lock Manager Server (LMS) and there is only one.

- ▶ **Job coordinator (CJQn)**

The job coordinator process is the scheduler in a Oracle instance. It starts jobs processes that execute batch processing. These jobs are PL/SQL statements or procedures in the instance. When a job has executed, CJQn will spawn a job queue slave process named J000 to J999. Thus, the slave process then proceeds to execute job processing. If the `JOB_QUEUE_PROCESSES` is set to 0, the job coordinator will not start.

4.1.5 Administration tools

Oracle provides several unique tools for managing various aspects of the database. These include tools for exporting and importing data, performance and availability monitoring, and centralized database administration. These tools

include Datapump, Export and Import, SQLLoader, Oracle Enterprise Manager (OEM), and Oracle Tuning Pack.

CONNECT INTERNAL and Server Manager are no longer supported; therefore, use CONNECT / AS SYSDBA or CONNECT username/password AS SYSDBA instead of CONNECT INTERNAL.

Use SQL*Plus to startup and to shutdown Oracle from the command line instead of using Server Manager administration tasks such as startup and shutdown of the database, backup and recovery, and running dynamic SQL statements. There are both command-line and GUI versions of the tool available.

Export and Import utilities

Oracle provides utilities for the exporting and importing of data contained in the database. The primary tools for exporting and importing are the **exp** and **imp** commands. The **exp** command extracts the object definitions and table data from an Oracle Database and stores them in an Oracle proprietary binary format on disk or tape. Likewise, the **imp** command reads the object definitions and table data from the exported data file and inserts them into an Oracle Database.

While the **imp** command addresses the need for importing data that was exported from an existing Oracle Database, it does not provide a facility for importing user defined data. The Oracle utility SQL*Loader loads data from external data sources into an Oracle Database. SQL*Loader can be used to do the following:

- ▶ Load data from multiple input files of different file types
- ▶ Load data from disk, tape, or named pipes
- ▶ Load data directly into Oracle datafiles, thereby bypassing the Oracle buffers and increasing the speed of data import operations
- ▶ Load fixed format, delimited format, or variable length records
- ▶ Selectively filter data based on predefined filtering rules
- ▶ Load xml data into the database

Oracle Data Pump

Oracle Data Pump is a new feature of Oracle Database 10g that enables very fast bulk data and metadata movement between Oracle Databases. Oracle Data Pump provides new high-speed, parallel Export and Import utilities (expdp and impdp) as well as a Web-based Oracle Enterprise Manager interface.

- ▶ Data Pump Export and Import utilities are typically much faster than the original Export and Import Utilities. A single thread of Data Pump Export is about twice as fast as original Export, while Data Pump Import is 15-45 times faster than original Import.

- ▶ Data Pump jobs can be restarted without loss of data, whether or not the stoppage was voluntary or involuntary.
- ▶ Data Pump jobs support fine-grained object selection. Virtually any type of object can be included or excluded in a Data Pump job.
- ▶ Data Pump supports the ability to load one instance directly from another (network import) and unload a remote instance (network export).

SQL*Loader

SQL*Loader is a high-speed data loading utility that loads data from external files into tables in an Oracle Database. SQL*Loader accepts input data in a variety of formats, performs filtering, and loads data into multiple Oracle Database tables during the same load session.

SQL*Loader provides three methods for loading data: Conventional Path Load, Direct Path Load, and External Table Load.

Oracle Enterprise Manager (OEM)

Oracle Enterprise Manager (OEM) is an integrated system management tool for managing Oracle Databases. It includes a graphical console, intelligent system agents, and access to common database management tools. OEM is invoked using the **oemapp <application>** command.

The graphical console serves as a central management point for the database allowing the database administrator to complete the following tasks:

- ▶ Administer and tune multiple databases
- ▶ Distribute software to both client and servers
- ▶ Schedule jobs to run on different databases at different times
- ▶ Monitor and respond to predefined database events

Figure 4-3 on page 98 shows the Oracle Enterprise Manager console.

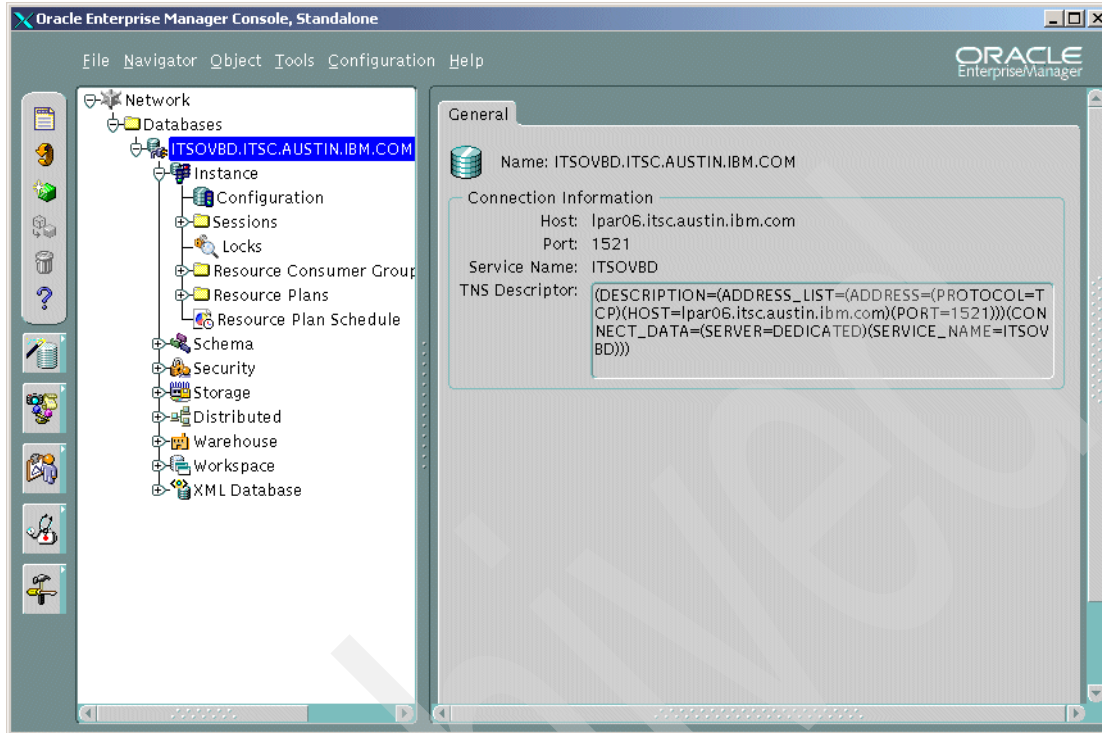


Figure 4-3 Oracle Enterprise Manager console

Oracle Tuning Pack

Oracle Tuning Pack is a set of tools that detect performance bottlenecks and tune an Oracle instance.

With these tools, database administrators can perform the following tasks:

- ▶ Identify and resolve performance problems
- ▶ Identify the cause of a reported problem, and get advice on the best way to fix it
- ▶ Maintain existing performance
- ▶ Avoid performance problems by practicing proper maintenance operations
- ▶ Identify potential tuning problems before they occur
- ▶ Provide tools and methodologies to establish and maintain database performance

Following is a list of some of the tools in the Oracle Tuning Pack:

- Oracle Expert

This tool is useful to see if the database configuration is optimal for performance based on its current workload. Based on this analysis, Oracle Expert will create some scripts that help to implement the tuning recommendations.

- Index Tuning Wizard

The Index Tuning is a *must use* tool for all developers and DBAs working on tuning SQL statements and logical database design. It helps them see if the chosen index strategy is the right one.

- SQL Analyzer

The SQL Analyzer allows a developer or DBA to check if the SQL statement that is sent to the Oracle instance is written for performance. If not, the tool provides some hints on how to improve it. This tool also helps the DBA to identify poorly written SQL and the most resource consuming SQL.

- Tablespace Map

Oracle Tablespace Map helps the database administrator improve I/O performance by mapping the used and free space as well as the location of the objects inside the tablespace. It also helps to determine if the tablespace needs to be reorganized.

- Reorganize Wizard

When storage problems are detected using Tablespace Map, reorganization may be a way to solve it. Unfortunately, reorganization is not an easy task. DBAs often get into trouble when they walk through all the necessary steps to reorganize Oracle Databases. They usually write shell scripts that run SQL scripts to make this reorg. These scripts may contain errors, which can leave the database in an inconsistent status. For example, the index generation script may contain a comma instead of a semicolon, and it will not work. Oracle knows about this problem and provides DBAs with this unique tool designed to help them to reorganize the database or just a schema object



Introduction to Microsoft SQL Server

This chapter provides a brief overview of Microsoft SQL (MS SQL) Server for the storage administrator or person unfamiliar with MS SQL. Microsoft SQL Server is a relational database management and analysis system for e-commerce, line-of-business, and data-warehouse solutions.

5.1 Database Architecture

Microsoft SQL Server data is stored in databases. The data in a database is organized into the logical components visible to users. A database is also physically implemented as two or more files on disk.

When using a database, you work primarily with the logical components such as tables, views, procedures, and users. The physical implementation of files is largely transparent. Typically, only the database administrator needs to work with the physical implementation.

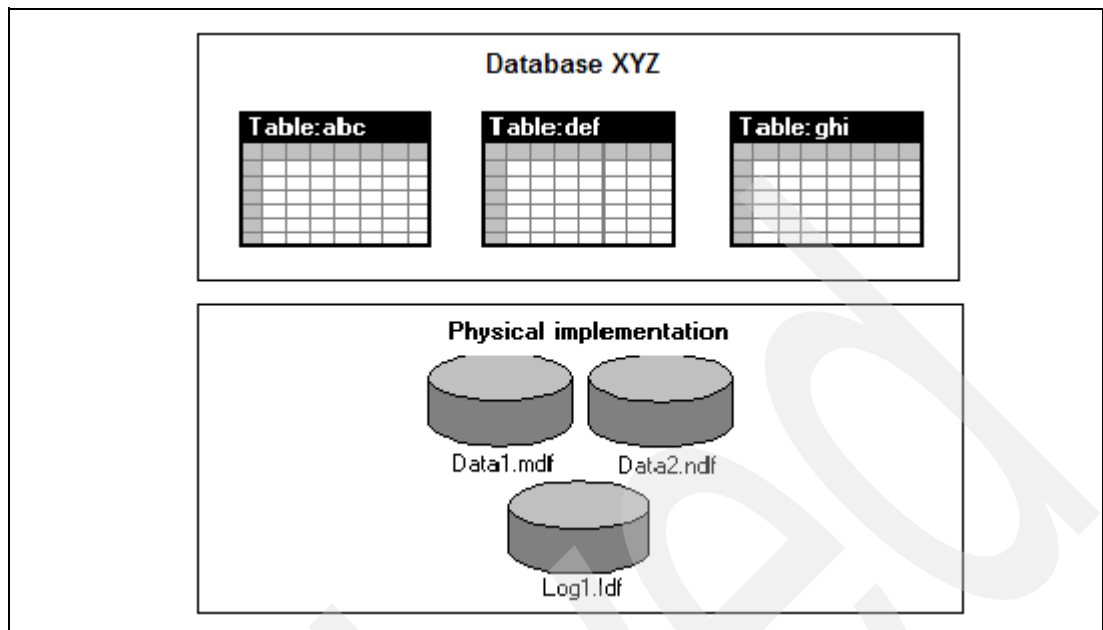


Figure 5-1 Database architecture

Each instance of SQL Server has four system databases (master, model, tempdb, and msdb) and one or more user databases. Some organizations have only one user database that contains all the data for their organization. Some organizations have different databases for each group in their organization and sometimes a database used by a single application. For example, an organization could have one database for sales, one database for payroll, one database for a document management application, and so on. Sometimes an application uses only one database; alternately, other applications may access several databases.

It is not necessary to run multiple copies of the SQL Server database engine to allow multiple users to access the databases on a server. An instance of the SQL Server Standard or Enterprise Edition can handle thousands of users working in multiple databases at the same time. Each instance of SQL Server makes all databases in the instance available to all users that connect to the instance—subject to the defined security permissions.

5.1.1 Logical database components

The data in a Microsoft SQL Server database is organized into several different objects. These objects are what a user can see when they connect to the database. In SQL Server, these components are defined as the following objects: Constraints, Tables, Defaults, Triggers, Indexes, User-defined data types, Keys, User-defined functions, Stored procedures, and Views.

System databases and data

Microsoft SQL Server systems have the following four system databases:

- ▶ master
- ▶ tempdb
- ▶ model
- ▶ msdb

master

The master database records all of the system level information for a SQL Server system. It records all login accounts and all system configuration settings. Master is the database that records the existence of all other databases, including the location of the database files. Master records the initialization information for SQL Server.

tempdb

The tempdb holds all temporary tables and temporary stored procedures. It also fills any other temporary storage needs such as work tables generated by SQL Server. Tempdb is a global resource. The temporary tables and stored procedures for all users connected to the system are stored there. Tempdb is re-created every time SQL Server is started, so the system starts with a clean copy of the database. Because temporary tables and stored procedures are dropped automatically on disconnect, and no connections are active when the system is shut down, there is never anything in tempdb to be saved from one session of SQL Server to another.

By default, tempdb autogrows as needed while SQL Server is running. Unlike other databases, however, it is reset to its initial size each time the database engine is started. If the size defined for tempdb is small, part of your system processing load may be taken up with autogrowing tempdb to the size needed to support your workload each time you restart SQL Server. You can avoid this overhead by using ALTER DATABASE to increase the size of tempdb.

model

The model database is used as the template for all databases created on a system. When a CREATE DATABASE statement is issued, the first part of the database is created by copying in the contents of the model database, and then the remainder of the new database is filled with empty pages. Because tempdb is created every time SQL Server is started, the model database must always exist on a SQL Server system.

msdb

The msdb database is used by SQL Server Agent for scheduling alerts, jobs, and recording operators.

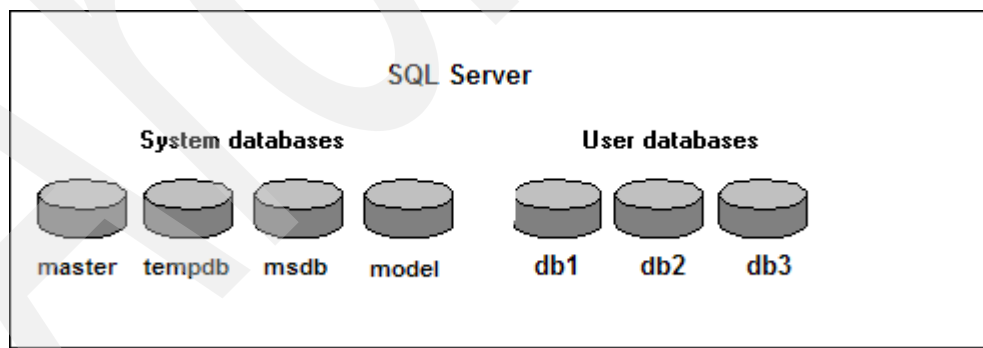


Figure 5-2 Database architecture

5.1.2 Physical database architecture

The topics in this section describe the way Microsoft SQL Server files and databases are organized.

► Pages and Extents

The fundamental unit of data storage in Microsoft SQL Server is the page. In SQL Server, the page size is 8 KB. This means SQL Server databases have 128 pages per megabyte. The start of each page is a 96-byte header used to store system information, such as the type of page, the amount of free space on the page, and the object ID of the object owning the page.

Table 5-1 shows eight types of pages in the data files of a SQL Server database:

Table 5-1 Page types

Page type	Contents
Data	Data rows with all data except text , ntext , and image data.
Index	Index entries.
Text/Image	Text , ntext , and image data.
Global Allocation Map, Secondary Global Allocation Map	Information about allocated extents.
Page Free Space	Information about free space available on pages.
Index Allocation Map	Information about free space available on pages.
Bulk Changed Map	Information about extents modified by bulk operations since the last BACKUP LOG statement.
Differential Changed Map	Information about extents that changed since the last BACKUP DATABASE statement.

There is an entry for each row on the page, and each entry records how far the first byte of the row is from the start of the page. The entries in the row offset table are in reverse sequence from the sequence of the rows on the page.

Rows cannot span pages in SQL Server. In SQL Server, the maximum amount of data contained in a single row is 8060 bytes, not including text, ntext, and image data.

Extents are the basic unit in which space is allocated to tables and indexes. An extent is eight contiguous pages, or 64 KB. This means SQL Server databases have 16 extents per megabyte.

To make its space allocation efficient, SQL Server does not allocate entire extents to tables with small amounts of data. SQL Server has the following two types of extents:

- Uniform extents are owned by a single object. All eight pages in the extent can only be used by the owning object.
- Mixed extents are shared by up to eight objects.

A new table or index is usually allocated pages from mixed extents. When the table or index grows to the point that it has eight pages, it is switched to uniform extents. If you create an index on an existing table that has enough rows to generate eight pages in the index, all allocations to the index are in uniform extents.

Physical database files and filegroups

Microsoft SQL Server maps a database over a set of operating-system files. Data and log information are never mixed on the same file, and individual files are used only by one database.

SQL Server databases have the following three types of files:

- ▶ **Primary data files**

The primary data file is the starting point of the database and points to the other files in the database. Every database has one primary data file. The recommended file name extension for primary data files is .mdf.

- ▶ **Secondary data files**

Secondary data files comprise all of the data files other than the primary data file. Some databases may not have any secondary data files, while others have multiple secondary data files. The recommended file name extension for secondary data files is .ndf.

- ▶ **Log files**

Log files hold all of the log information used to recover the database. There must be at least one log file for each database, although there can be more than one. The recommended file name extension for log files is .ldf.

Following are examples of the logical file names and physical file names of a database created on a default instance of SQL Server.

Example database XYZ:

- ▶ **MyDB_primary**

Primary data file => D:\Data\MyData1.mdf

- ▶ **MyDB_secondary1**

Secondary data file => D:\Data\MyData2.ndf

- ▶ **MyDB_secondary2**

Secondary data file => D:\Data\MyData3.ndf

- ▶ **MyDB_log1**

Log file => E:\Data\Mylog1.ldf

- ▶ **MyDB_log2**

Log file => E:\Data\Mylog2.ldf

Archived



Part 3

Using N series and MSSQL

In this part we discuss topics relevant to the IBM System Storage N series and MSSQL solution.

Archived

Preparing IBM System Storage N series for MS Windows & MSSQL

This chapter discusses setting up SnapDrive and CIFS shares for use with MSSQL or other databases in Windows environment.

6.1 Joining Active Directory

In this section we discuss the basics to joining a Windows Domain.

Data ONTAP

Data ONTAP is a proprietary operating system. It is not based on the Windows OS. Consequently, the current Data ONTAP operating system requires additional rights assigned to the user or to the pre created device object when an administrator or administrator equivalent account is not used. After the computer object successfully joins the Active Directory® domain, the user account credentials are no longer used and are not stored in any way in the OS. They are used only to allow the N series product to become an active member of Active Directory and to write standard properties to the object during the join process. The properties that are written are listed in the next section.

Machines need accounts too

Every computer running a Windows workstation (Windows NT® 4.0 or higher), Windows server operating system, or IBM System Storage N series has a computer account (Figure 6-1 on page 110). As for users who require a valid account before being allowed to access a networked resource, it is also requisite that workstations, servers, and other devices participating in an Active Directory domain have an account that provides a means for authenticating and auditing computer access to the network and access control, security, and management to domain resources.

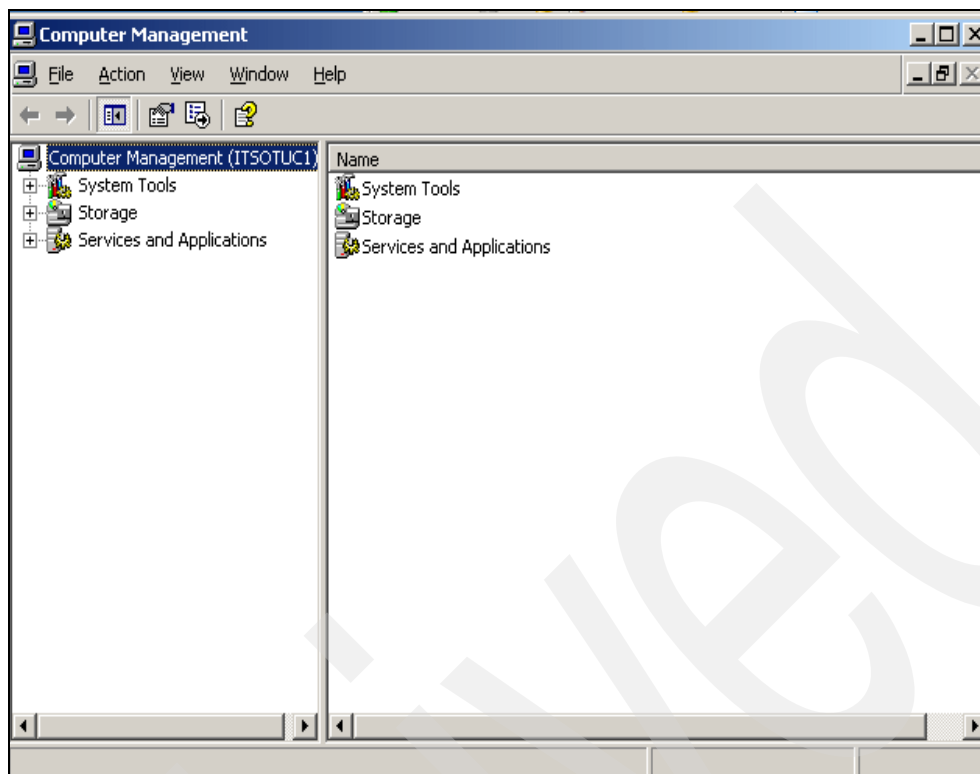


Figure 6-1 Using Microsoft Windows computer management on IBM System Storage N series

6.1.1 Preparing for CIFS shares

1. First determine the host name of the IBM System Storage N series. You can do this on the command line by issuing the **hostname** command.

```
Data ONTAP (itsotuc2)
login: root
Password:
itsotuc2> Sat Jan  3 14:34:40 MST [telnet_0:info]: root logged in from host:
192
.168.3.242

itsotuc2*> hostname
itsotuc2
itsotuc2*>
```

Figure 6-2 hostname command

2. Next determine the IP address of the IBM System Storage N series. This can be done with the **ifconfig -a** command.

```
e0a: flags=848043<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.252 netmask 0xffffffff broadcast 192.168.3.255
    ether 00:a0:98:01:ff:c2 (auto-100tx-fd-up) flowcontrol full
e0b: flags=848043<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.251 netmask 0xffffffff broadcast 192.168.3.255
    ether 00:a0:98:01:ff:c3 (auto-100tx-fd-up) flowcontrol full
lo: flags=1948049<UP,LOOPBACK,RUNNING,MULTICAST,TCPCSUM> mtu 8160
    inet 127.0.0.1 netmask 0xff000000 broadcast 127.0.0.1
    ether 28:bb:cf:37:04:00 (VIA Provider)
itsotuc2*>
```

Figure 6-3 `ifconfig -a` command

3. Make sure the IBM System Storage N series is licensed for CIFS.

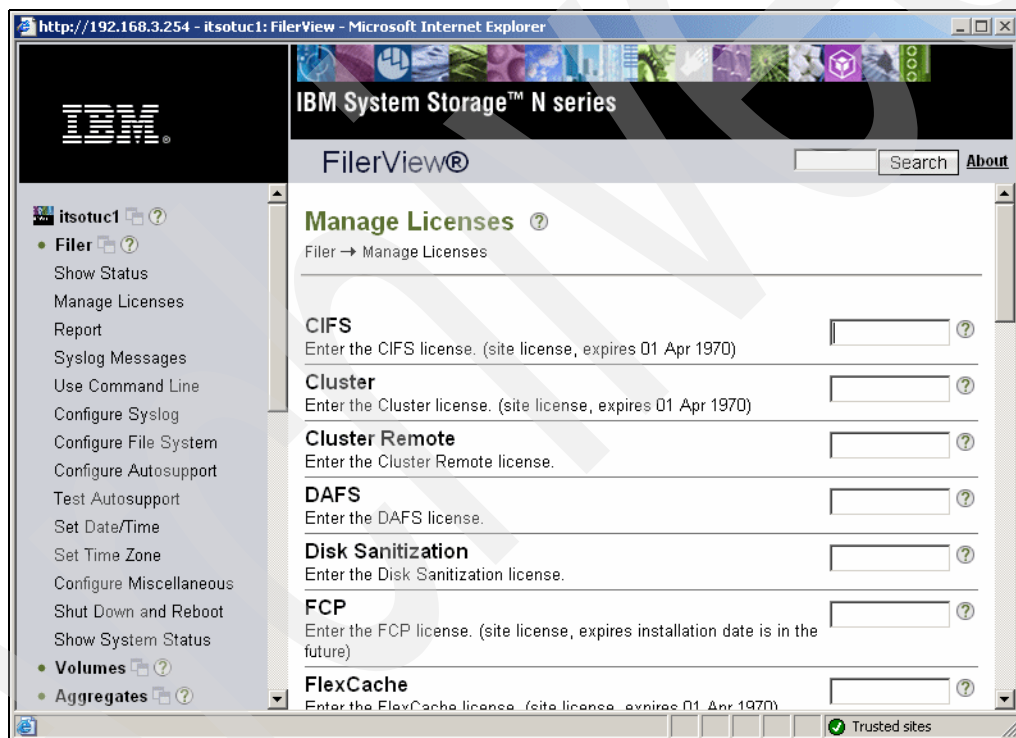


Figure 6-4 Licensing

6.1.2 Select a user account

Create or select a user account for use to pre create the IBM System Storage N series computer object (see Figure 6-5). Make sure the IBM System Storage N series acquires minimum rights.

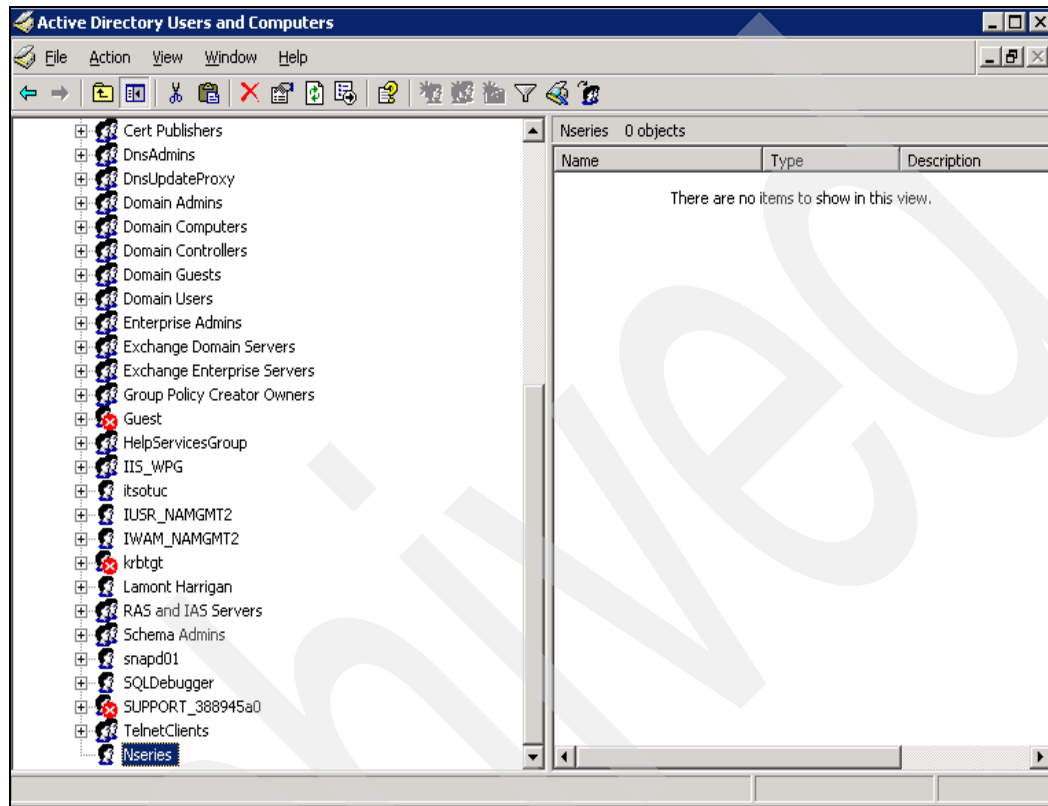


Figure 6-5 User to be used with creation of IBM System Storage N series computer object

1. Select View from the top of the panel, and make sure **Advanced View** is selected. See Figure 6-6.

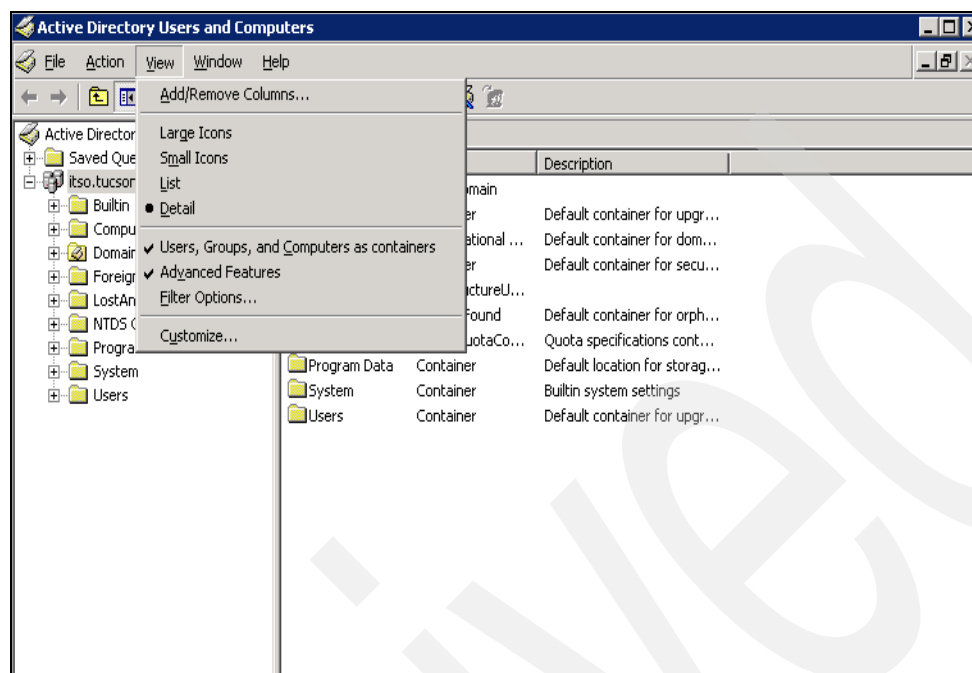


Figure 6-6 Advanced users

2. Right-click and select the security tab. See Figure 6-7.

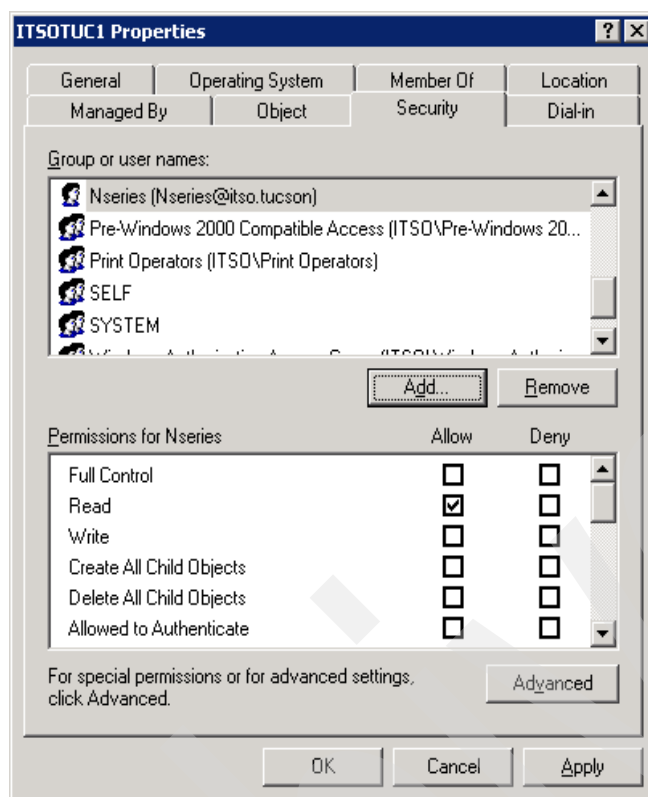


Figure 6-7 Permissions

3. On the Security tab in the permissions area, scroll down and select allow **Change Password** or **Reset Password**. See Figure 6-8.

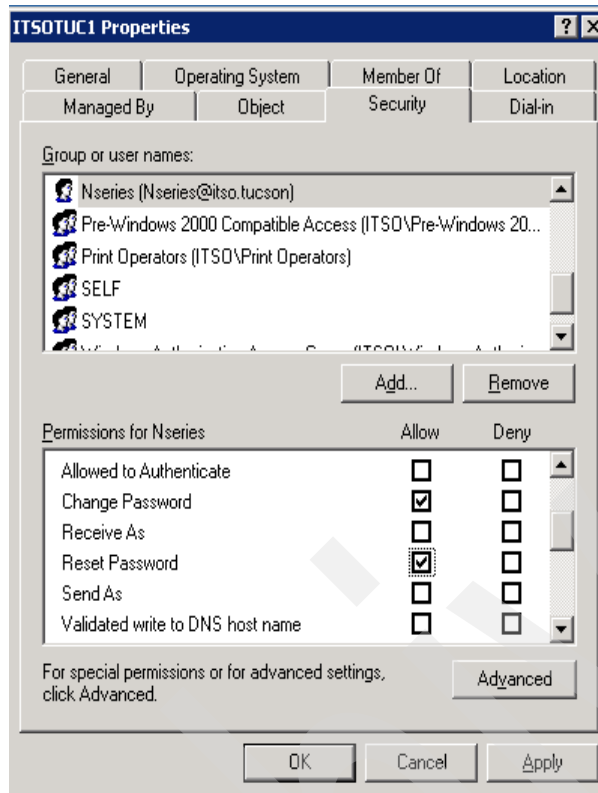


Figure 6-8 Password permissions

4. On the same Security tab in the permissions area, scroll down and select **Write Public Information**. See Figure 6-9.

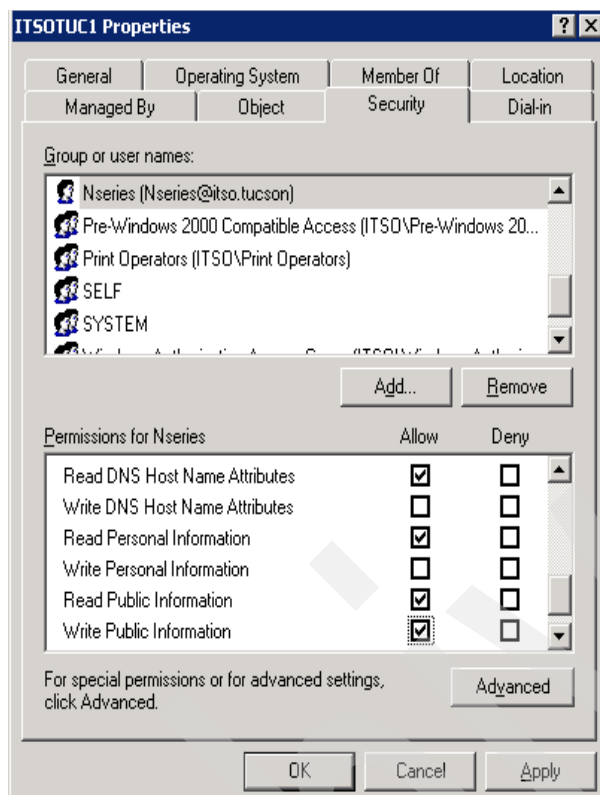


Figure 6-9 Write Public Information permission

6.2 Active Directory

This section covers subjects related to the IBM System Storage N series joining the Microsoft Windows Active directory.

6.2.1 Should Active Directory be in mixed or native Mode?

The terms mixed mode and native mode refer to functional levels in a Windows 2000 server. In a Windows 2003 server, the terms mixed and native have been superseded by Raise Function Level.

Domain Function Levels (mixed and native)

There are now four domain levels in which a Windows 2003 server can operate.

- ▶ Windows 2003 server. All Windows 2003 servers. No other domain controllers. However, even in this level, the whole range of clients (including the N series product) and member servers can still join the domain.
- ▶ Windows 2003 server interim. Windows NT 4.0 servers and Windows 2003 servers (no Windows 2000). This level arises when you upgrade a Windows NT 4.0 PDC to a Windows 2003 server. Interim mode is important when you have Windows NT 4.0 groups with more than 5,000 members. Windows 2000 does not allow you to create groups with more than 5,000 members.

- ▶ Windows 2000 native. Allows Windows 2000 and Windows 2003 servers (no Windows NT 4.0).
- ▶ Windows 2000 mixed. Allows Windows NT 4.0 BDCs and Windows 2000. Naturally Windows 2000 mixed is the default function level, because it supports all types of domain controllers.

6.2.2 IBM System Storage N series

An IBM System Storage N series may be joined to Active Directory whether in mixed, native, interim, or pure Windows 2003 server mode.

6.2.3 Precreating a computer object

Many Active Directory administrators employ a set of best practices that place strict controls over who can create computer objects. If the join is performed in the following manner, security risks are minimized because the need for Active Directory administrator rights at the device during the setup process is eliminated.

First pre create the computer object using an account with the required privileges, and later use an account with fewer privileges to log on to the computer and issue the appropriate command to complete the join process. Precreating a computer object is the recommended method of joining a the N series product to Active Directory.

6.2.4 Creating the computer object

In order for the N series product to join the Active Directory a computer object referencing it must be created.

1. Open the Active Directory MMC, select computer objects in your domain. Right-click and create a new computer object (See Figure 6-10 on page 118).
2. Add a new computer object referencing your IBM System Storage N series (See Figure 6-11 on page 118, and Figure 6-14 on page 120 and Figure 6-15 on page 120.) using the preselected account in section “Select a user account” on page 112.

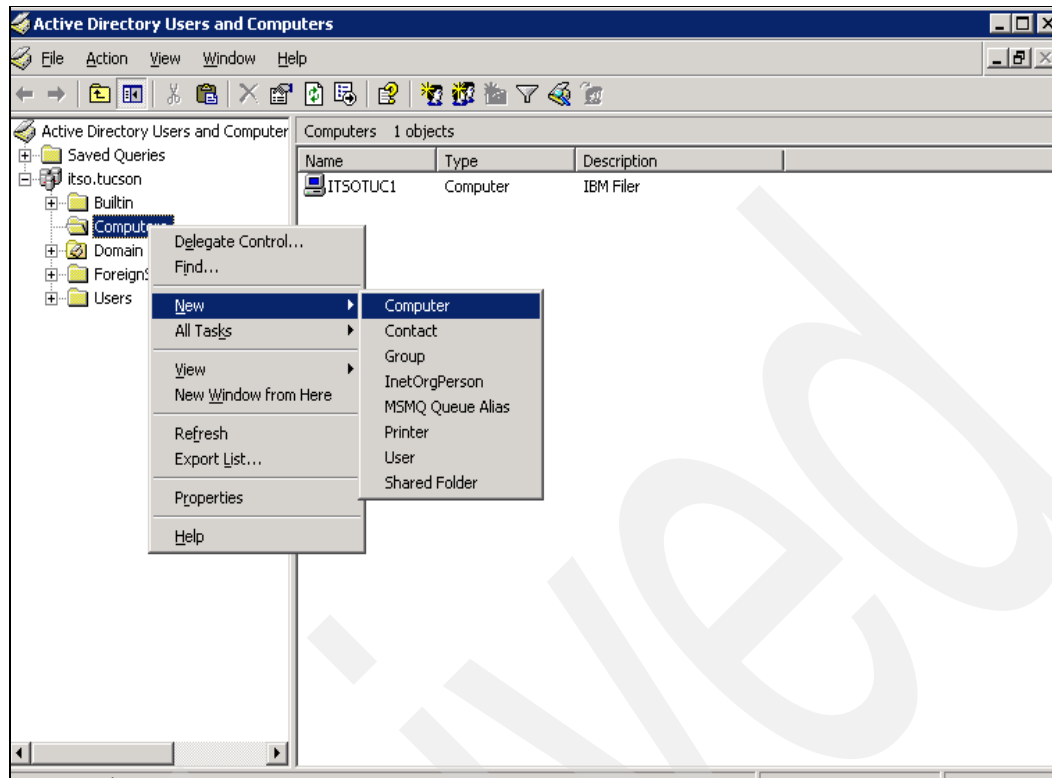


Figure 6-10 Creating a computer in Active Directory

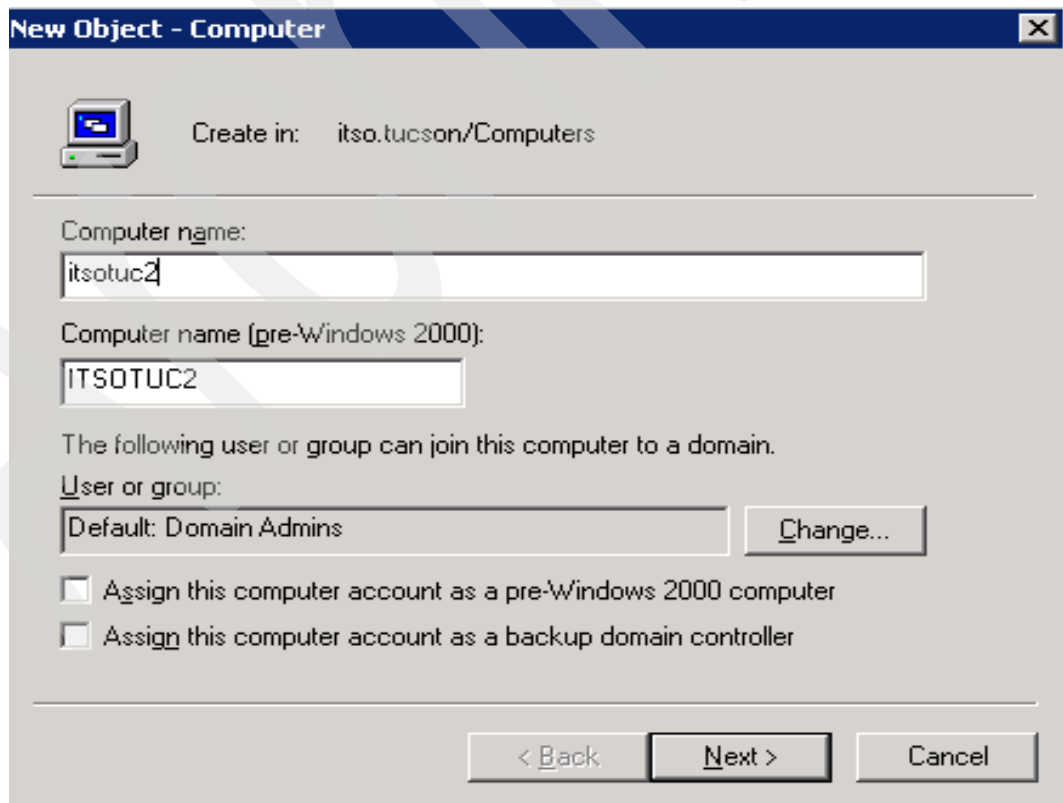


Figure 6-11 Adding a new computer object

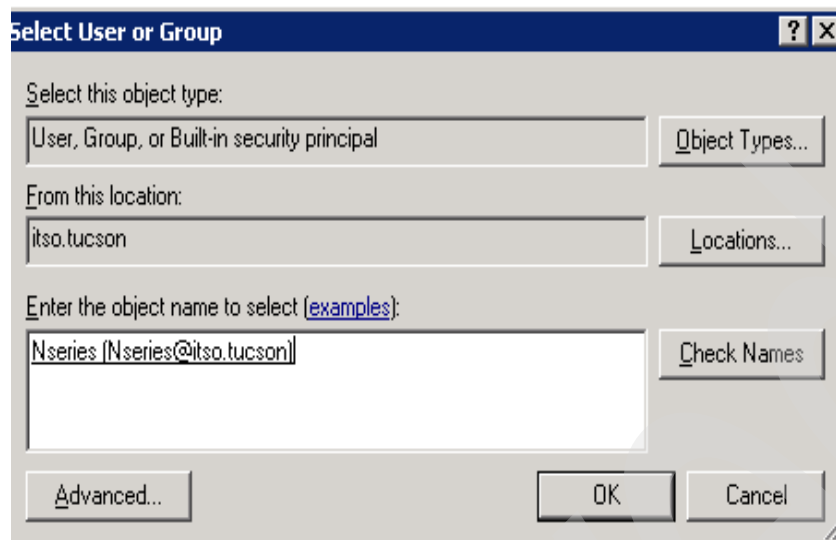


Figure 6-12 Selecting a user for the N series product computer object

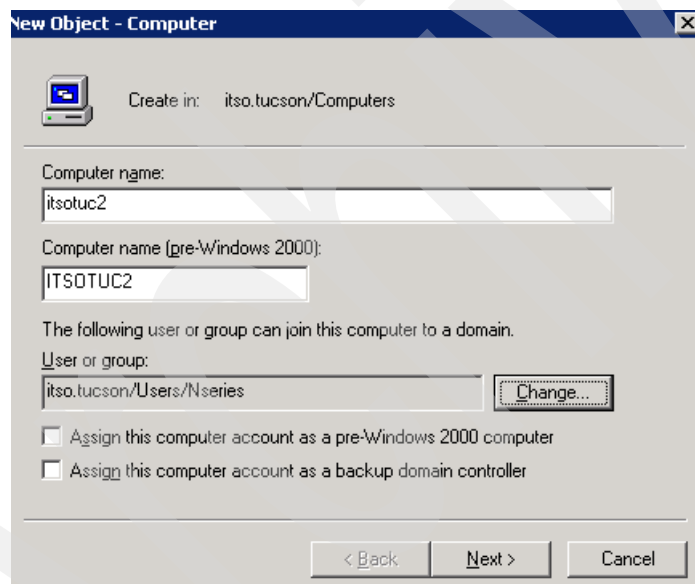


Figure 6-13 Results of specifying a user

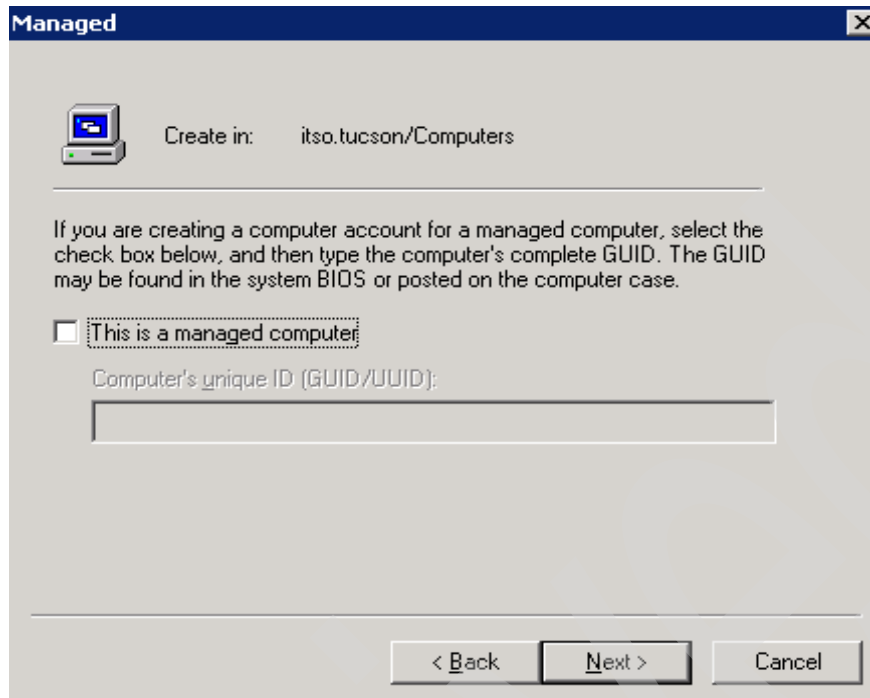


Figure 6-14 Creating a computer object

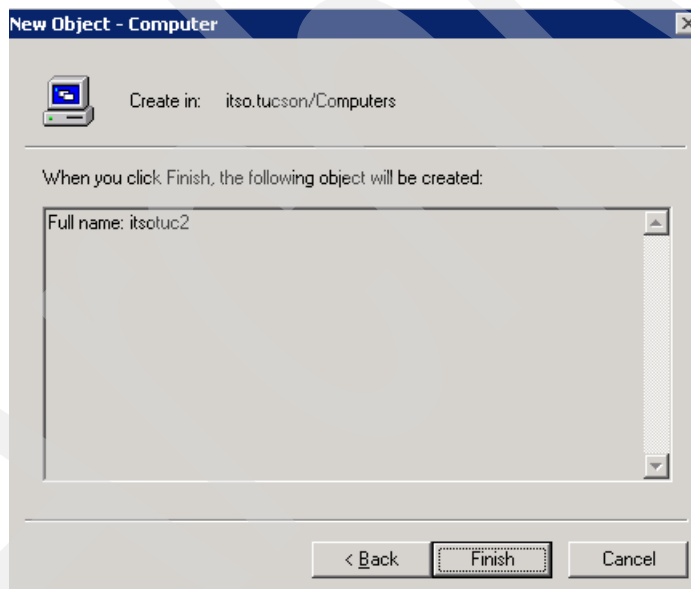


Figure 6-15 Last step in creating the computer object

3. You should see the computer object created for N series in the computer object container of your Active Directory.

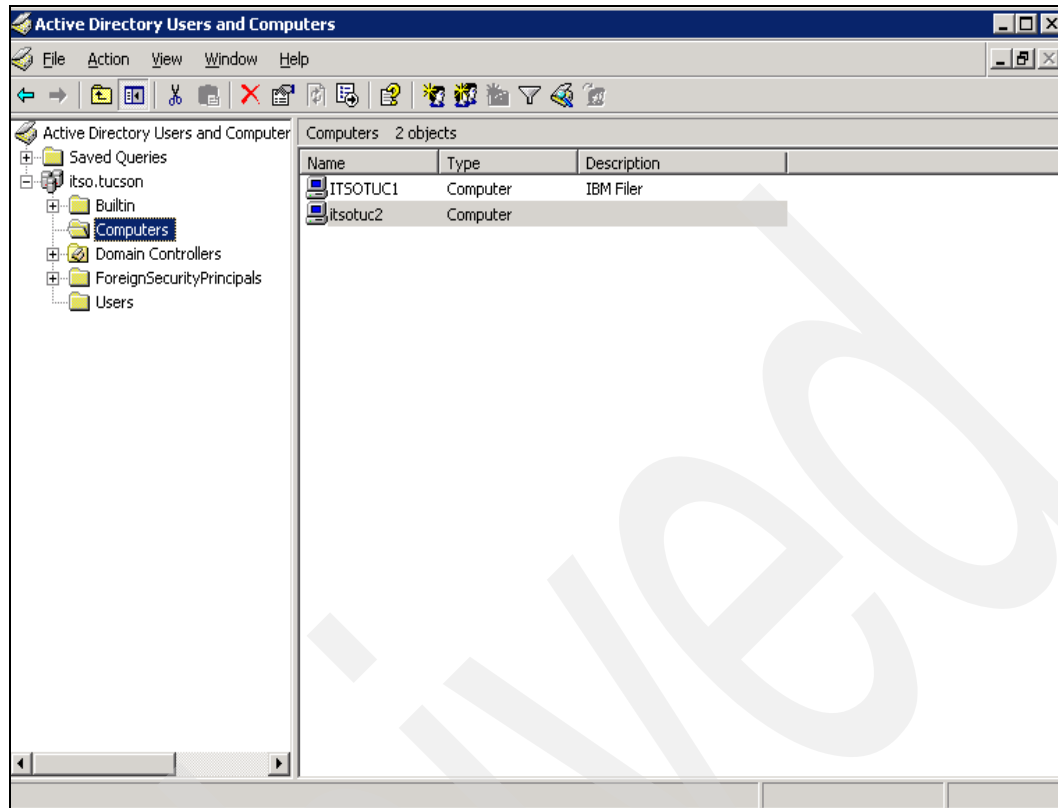


Figure 6-16 Verification of computer object creation

At the completion of the Active Directory join process, a number of properties are written to the computer account, including the following:

- ▶ DNS host name
- ▶ Several service principal names
- ▶ Object classes
- ▶ Operating system name and version
- ▶ A randomly generated password is set for this account via KPASSWD

Note: This is the only instance in which N series' join process differs from the Microsoft join process. Microsoft uses proprietary RPC calls to change the password, whereas N series uses the published KPASSWD APIs to accomplish this task.

6.2.5 Running CIFS setup with the N series product

In our example we used the CIFS setup wizard from the command line.

1. Answer yes to continue.

```
itsotuc2> cifs setup
This process will enable CIFS access to the filer from a Windows(R) system.
Use "?" for help at any prompt and Ctrl-C to exit without committing changes.

      This filer is currently a member of the Windows-style workgroup
      'WORKGROUP'.
Do you want to continue and change the current filer account information? [n]:
```

Figure 6-17 CIFS setup

2. Take the default no in answer to WINS. See Figure 6-18.

```
Do you want to continue and change the current filer account information? [n]:
y
      Your filer does not have WINS configured and is visible only to
      clients on the same subnet.
Do you want to make the system visible via WINS? [n]:
```

Figure 6-18 Filer account

3. Take the default to the NTFS question as you will want multiprotocol access.

```
Do you want to make the system visible via WINS? [n]:
      This filer is currently configured as a multiprotocol filer.
Would you like to reconfigure this filer to be an NTFS-only filer? [n]:
```

Figure 6-19 WINS setup

4. Keep the name assigned to IBM System Storage N series. During initial N series product set up take the default n. See Figure 6-20.

```
Would you like to reconfigure this filer to be an NTFS-only filer? [n]:
      The default name for this CIFS server is 'ITSOTUC2'.
Would you like to change this name? [n]:
```

Figure 6-20 Protocol setup

5. In our example we joined the Active Directory so we selected 1. See Figure Figure 6-22 on page 123.

```

Would you like to change this name? [n]:
    Data ONTAP CIFS services support four styles of user authentication.
    Choose the one from the list below that best suits your situation.

(1) Active Directory domain authentication (Active Directory domains only)
(2) Windows NT 4 domain authentication (Windows NT or Active Directory domains)
(3) Windows Workgroup authentication using the filer's local user accounts
(4) /etc/passwd and/or NIS/LDAP authentication

Selection (1-4)? [1]:

```

Figure 6-21 Name designation

```

Selection (1-4)? [1]: 1
What is the name of the Active Directory domain? []:

```

Figure 6-22 Active directory selection

6. Specify the Active Directory Domain.

```

Selection (1-4)? [1]: 1
What is the name of the Active Directory domain? []:itso.tucson

```

Figure 6-23 Active Directory Domain

7. Next specify the previously designated account in “Select a user account” on page 112.

```

In order to create an Active Directory machine account for the filer,
you must supply the name and password of a Windows account with
sufficient privileges to add computers to the ITSO.TUCSON domain.
Enter the name of the Windows user []: Nseries
Password for Nseries:

```

Figure 6-24 Enter the Previously selected user

8. Enter the password for the designate user.

```

Password for Nseries:
CIFS - Logged in as Nseries@ITSO.TUCSON.
An account that matches the name 'ITSOTUC2' already exists in Active
Directory: 'cn=itsotuc2,cn=computers,dc=itso,dc=tucson'. This is
normal if you are re-running CIFS Setup. You may continue by using
this account or changing the name of this CIFS server.
Do you want to re-use this machine account? [y]:

```

Figure 6-25 Password entry

9. Because we pre configured the computer object, the set up process will recognize this and ask if you want to reuse this object, answer yes. See Figure 6-25.

```
CIFS - Starting SMB protocol...
Currently the user "ITSOTUC2\administrator" and members of the group
"ITS0\Domain Admins" have permission to administer CIFS on this filer.
You may specify an additional user or group to be added to the filer's
"BUILTIN\Administrators" group, thus giving them administrative
privileges as well.
Would you like to specify a user or group that can administer CIFS? [n]:
```

Figure 6-26 account reuse

10. The SMB protocol will start and you are offered the chance to specify other users to administer N series. In our example we took the default no, after answering no you will see confirmation of joining the Active directory.

```
Welcome to the ITS0.TUCSON (ITS0) Active Directory(R) domain.

CIFS local server is running.
itsotuc2>
```

Figure 6-27 CIFS setup completion

11. To verify and get more info about the domain you just joined execute the CIFS DOMAININFO command. See Figure 6-28 on page 125.


```

IBM Storage System N3700
itsotuc1*> cifs domaininfo
NetBios Domain:          ITS0
Windows 2000 Domain Name: itso.tucson
Type:                    Windows 2000
Filer AD Site:           none

Current Connected DCs:   \\CHRISANTHY
Total DC addresses found: 4
Preferred Addresses:
                        None
Favored Addresses:
                        None
Other Addresses:
                        192.168.3.242   CHRISANTHY   PDC
                        192.168.88.1    PDC
                        9.11.218.250    PDC
                        192.168.110.1   PDC

Not currently connected to any AD LDAP server
Preferred Addresses:
                        None
Favored Addresses:
                        None
Other Addresses:
                        None
itsotuc1*>

```

Figure 6-28 cifs domaininfo

6.3 Why is it necessary to join an IBM System Storage N series to Active Directory

In order for resources on a network to be locatable, a mechanism must exist whereby the resources can easily be found. A directory service, in this case Active Directory, keeps track of all known resources and responds to requests with a list of currently available devices and services. But before you can be trusted to query for resources, you must be granted membership in the Active Directory domain.

Active Directory works on a container basis. A container can be a domain, organization unit (OU), or computer.

Following are the key benefits for IBM System Storage N series to join Active Directory:

- ▶ Controlled security and management through group management, for example, group policy objects (GPOs) and access control lists (ACLs) placed on objects and organization units (OUs).
- ▶ Single-sign-on and pass-through authentication for users.
- ▶ Interoperability by extending control beyond the native Windows environment through the Microsoft management interface by providing a read-only computer management view of:
 - Shared folders, shares, sessions, and open files
 - Local users and groups to the IBM System Storage N series

6.4 Troubleshooting the domain-joining process

This section covers the more common problems related to the N series product joining the Windows Microsoft Domain.

6.4.1 DNS

In order to determine whether N series' device is joining a Windows NT 4.0 domain or Active Directory and to locate domain controllers, a key distribution center (KDC used for Kerberos), and other necessary services, CIFS, rely on DNS. If DNS is not enabled or is configured incorrectly, the domain-joining phase will either fail, or if a Microsoft Windows Internet-naming server (WINS) is running, assume that the domain being joined is a Windows NT 4.0 domain.

6.4.2 Time synchronization

If time synchronization is not enabled, and the IBM System Storage N series' time drifts by more than five minutes from the domain's time, client authentication attempts to the IBM System Storage N series will fail until corrected.

6.4.3 Active Directory replication

Based on the size of the Active Directory domain, to propagate a change for a small organization with one site, the replication will usually take less than 15 minutes. For a global company with many sites, the replication may take up to several hours to complete.

6.5 Device Discovery

The IBM System Storage N series performs an intelligent discovery process to locate the most appropriate domain controller (DC) in the network with which to communicate. For its first connection, Data ONTAP attempts to use servers that appear in the CIFS prefdc list (in list order), if configured (see Figure 6-29 and Figure 6-30). If none of these preferred servers are available or configured, all server addresses are discovered at once and then categorized, prioritized, and cached.

```
itsotuc2> cifs prefdc print
No preferred Domain Controllers configured.
DCs will be automatically discovered.
```

Figure 6-29 prefdc list with nothing configured

```
itsotuc2> cifs prefdc print
Preferred DC ordering per domain:

ITS0:
    1. 192.168.3.242
```

Figure 6-30 cifs prefdc print with prefdc configured

Preferred addresses are ordered as specified using the `cifs prefdc` command. "Favored" and "other" categories are sorted according to the fastest response. Data ONTAP simultaneously pings all addresses listed in both categories and waits one second for responses.

The **`cifs prefdc`** command allows control over the order in which Data ONTAP attempts to contact a server. The list is consulted for all Windows service connections, not just domain controllers.

Note: When configuring CIFS on an N series device in a Windows 2000/2003 domain, an LDAP query to Active Directory checks to ensure that a computer object with the same name does not already exist. If the name does exist, the set up process makes sure it is not a domain controller. These are precautionary measures used to guarantee that no computer object names are duplicated in error.

6.6 SnapDrive set up

This section covers setting up SnapDrive with MSSQL.

6.6.1 Creating disk with iSCSI management

Use the following steps to create a disk with iSCSI management.

1. Open MMC Computer Management.
2. Select SnapDrive.
3. Right-click Disks, and then click in Create Disk.

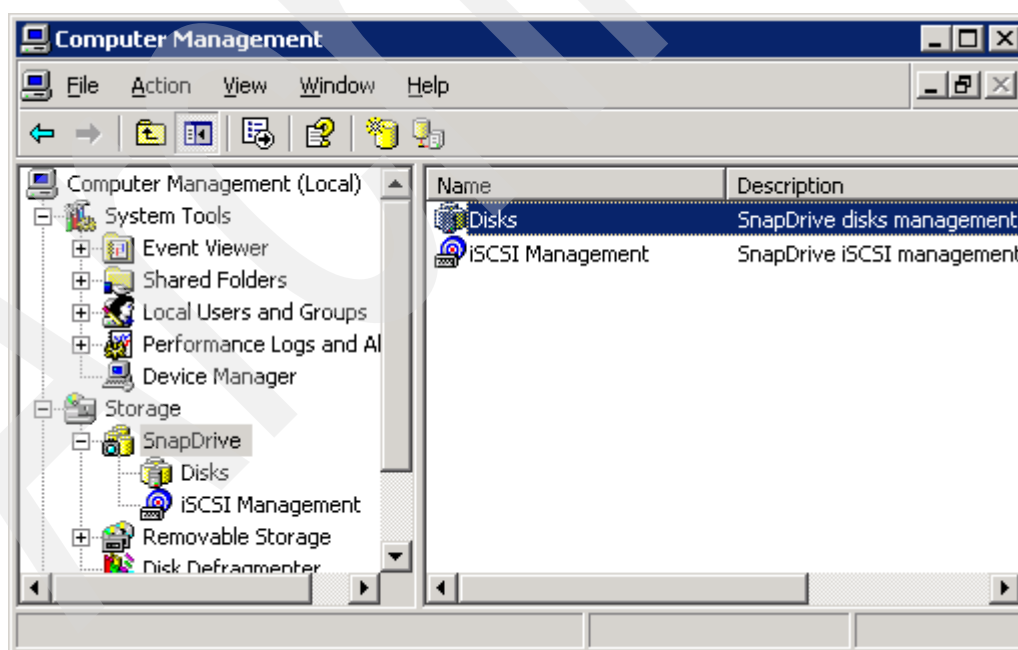


Figure 6-31 Create a disk using the sequence

4. Click Next. As shown in Figure 6-32.

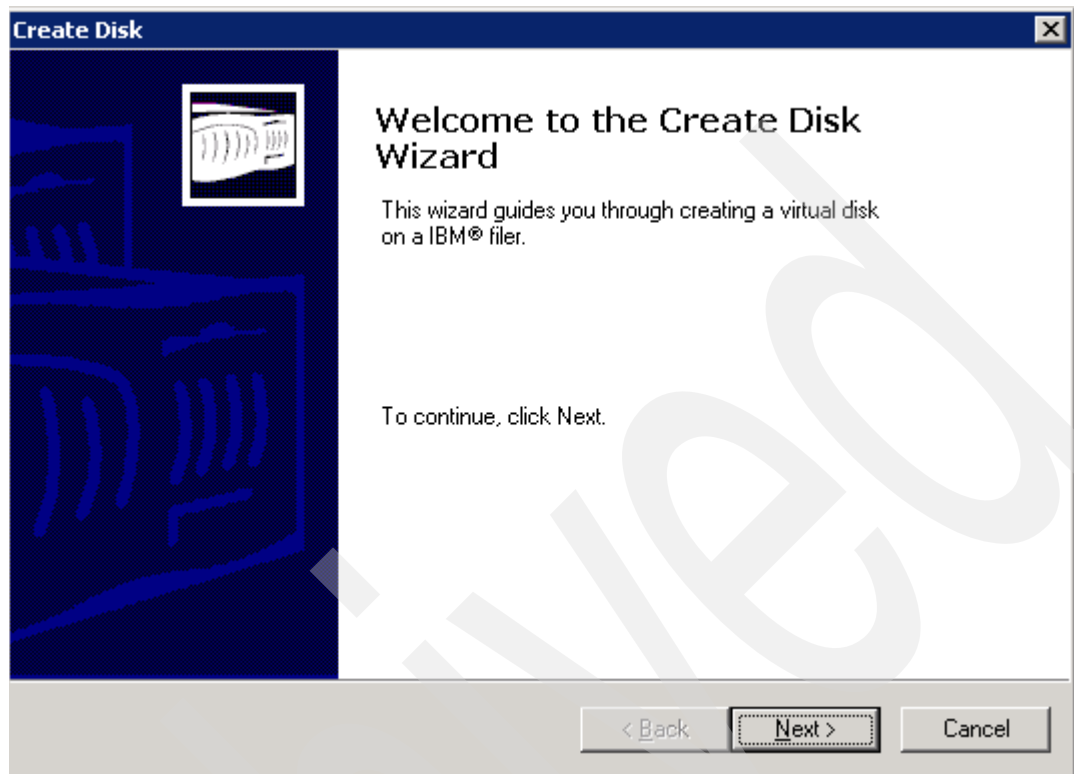


Figure 6-32 Welcome to Create Disk Wizard

5. Enter a virtual disk UNC path to file volume or qtree. In our example we used IBM System Storage N series itsotuc2 and FlexVol sqldata.
6. Enter a name for new virtual disk. In the example we used sqldata_2. As shown in Figure 6-33 on page 129.

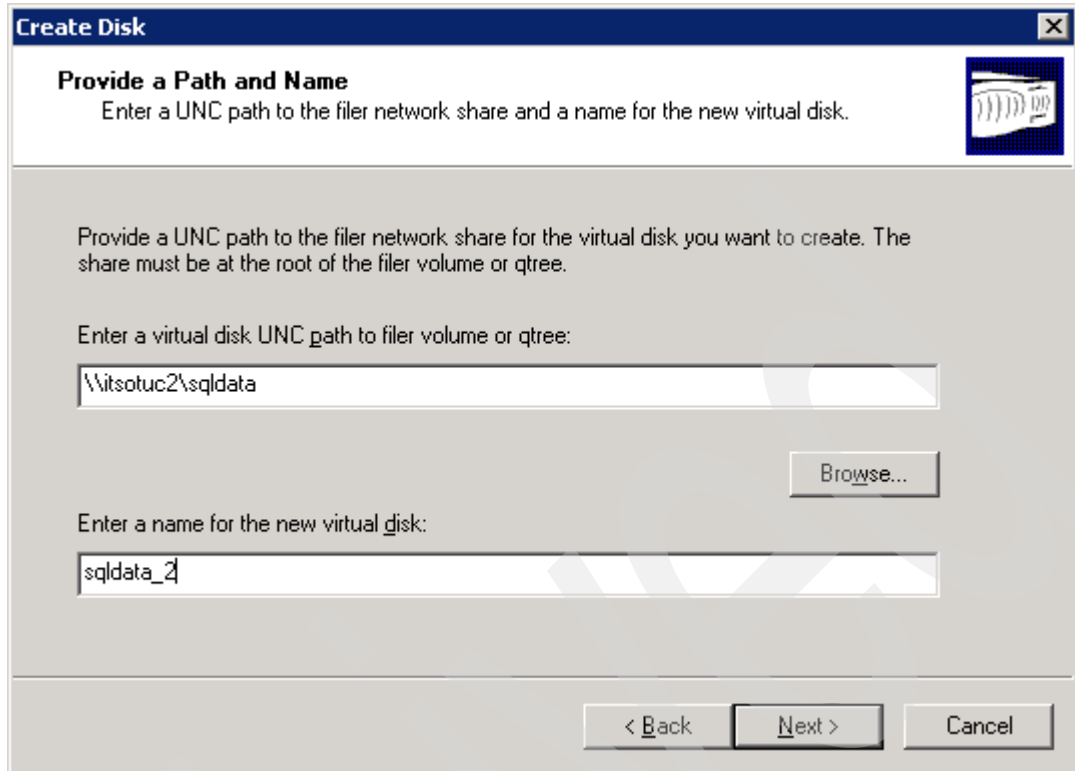


Figure 6-33 Informations volume

7. Select a virtual type disk, as shown in Figure 6-34.

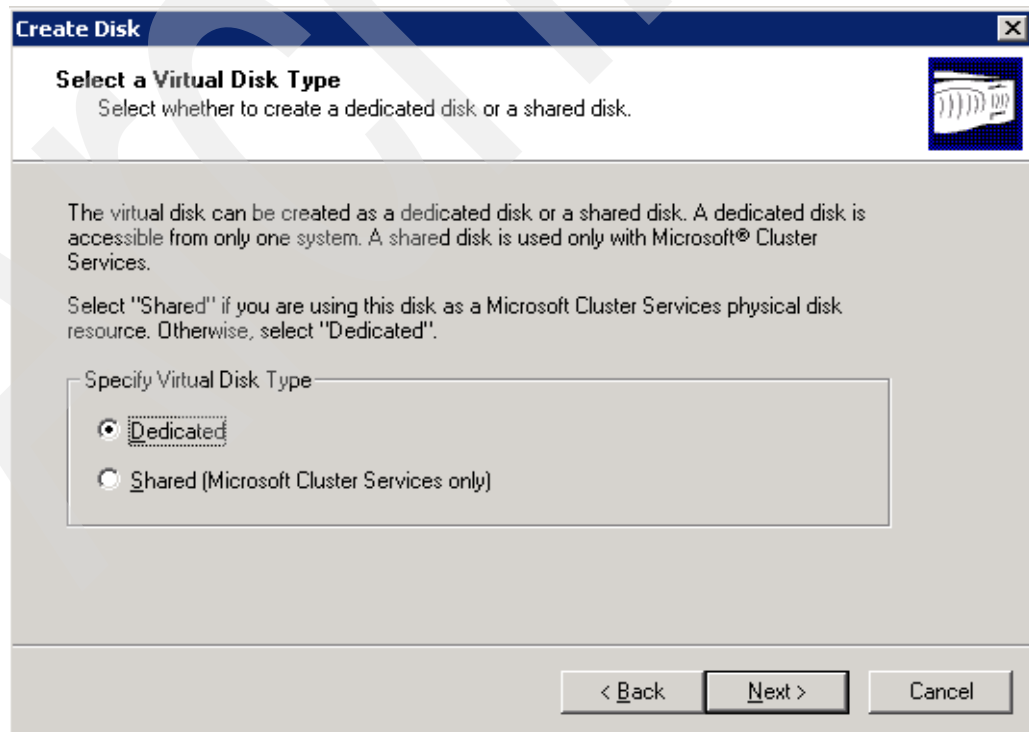
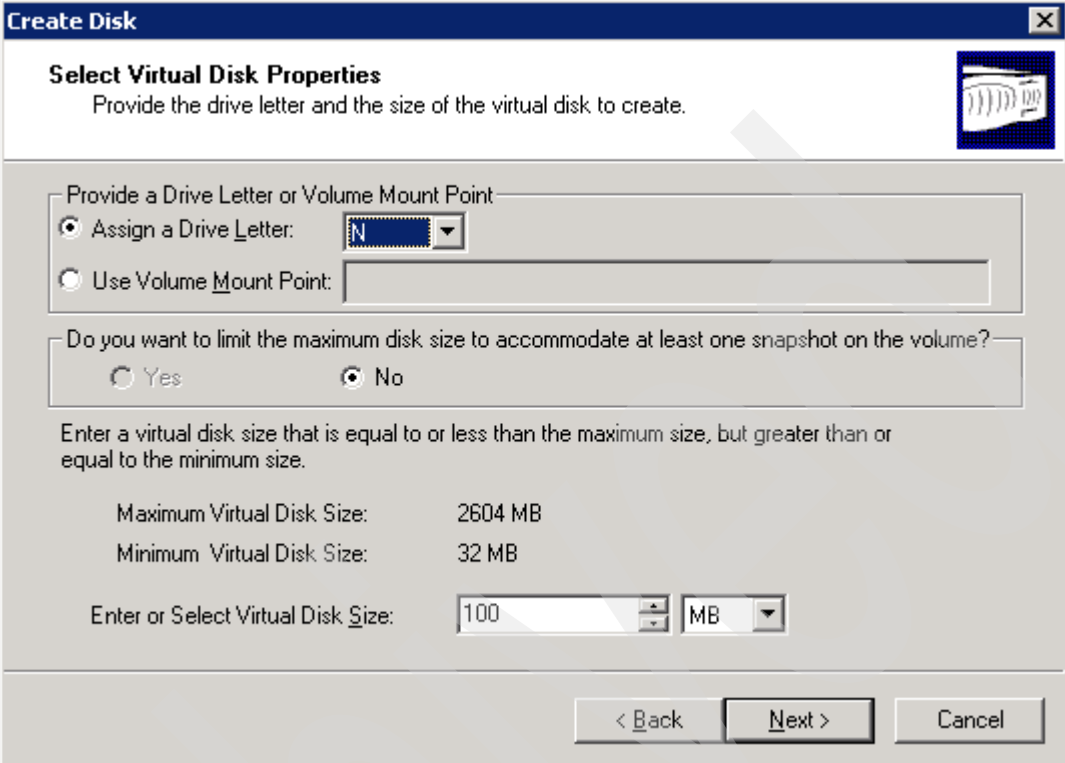


Figure 6-34 Specify Virtual Disk Type

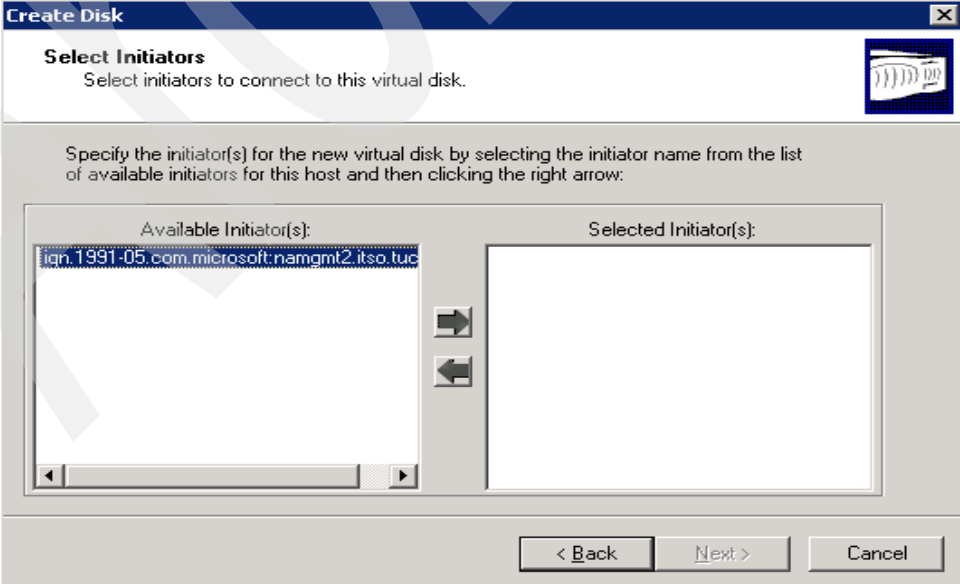
8. Select the Drive Letter and the size of the virtual disk to create. As shown in Figure 6-35.



The 'Create Disk' dialog box, titled 'Select Virtual Disk Properties', prompts the user to 'Provide the drive letter and the size of the virtual disk to create.' It features two radio buttons: 'Assign a Drive Letter' (selected) and 'Use Volume Mount Point'. The 'Assign a Drive Letter' option has a dropdown menu showing 'N'. Below this, there is a question: 'Do you want to limit the maximum disk size to accommodate at least one snapshot on the volume?' with 'Yes' and 'No' radio buttons, where 'No' is selected. Further down, it specifies size limits: 'Maximum Virtual Disk Size: 2604 MB' and 'Minimum Virtual Disk Size: 32 MB'. At the bottom, there is a field 'Enter or Select Virtual Disk Size:' with a value of '100' and a unit dropdown set to 'MB'. Navigation buttons at the bottom right include '< Back', 'Next >', and 'Cancel'.

Figure 6-35 Select drive letter and virtual disk size

9. On the Select Initiators panel, select the appropriate initiator from the Available Initiator section. As shown in Figure 6-36.
10. Use the left-pointing arrow to move the selected initiator to the Selected Initiator section.



The 'Create Disk' dialog box, titled 'Select Initiators', prompts the user to 'Select initiators to connect to this virtual disk.' It instructs the user to 'Specify the initiator(s) for the new virtual disk by selecting the initiator name from the list of available initiators for this host and then clicking the right arrow:'. The dialog is divided into two main sections: 'Available Initiator(s):' on the left and 'Selected Initiator(s):' on the right. In the 'Available Initiator(s)' list, the text 'iqn.1991-05.com.microsoft:namgmt2.itso.tuc' is selected. Between the two lists are two arrows: a right-pointing arrow and a left-pointing arrow. At the bottom right, there are navigation buttons: '< Back', 'Next >', and 'Cancel'.

Figure 6-36 Select Initiators

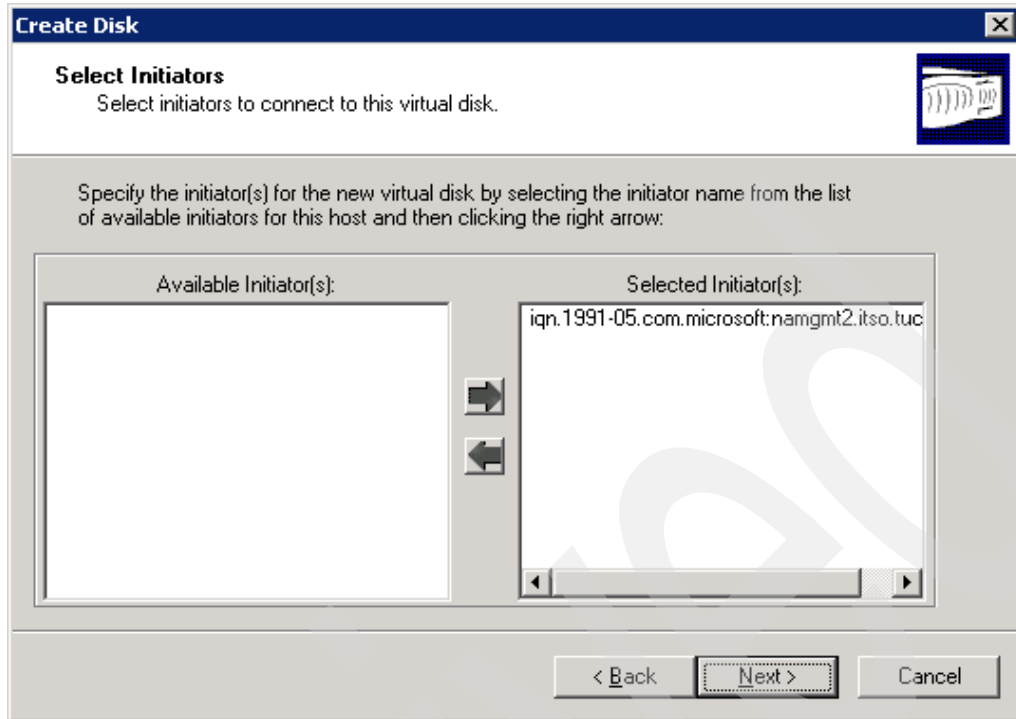


Figure 6-37 Select Initiator

11. Click Finish.

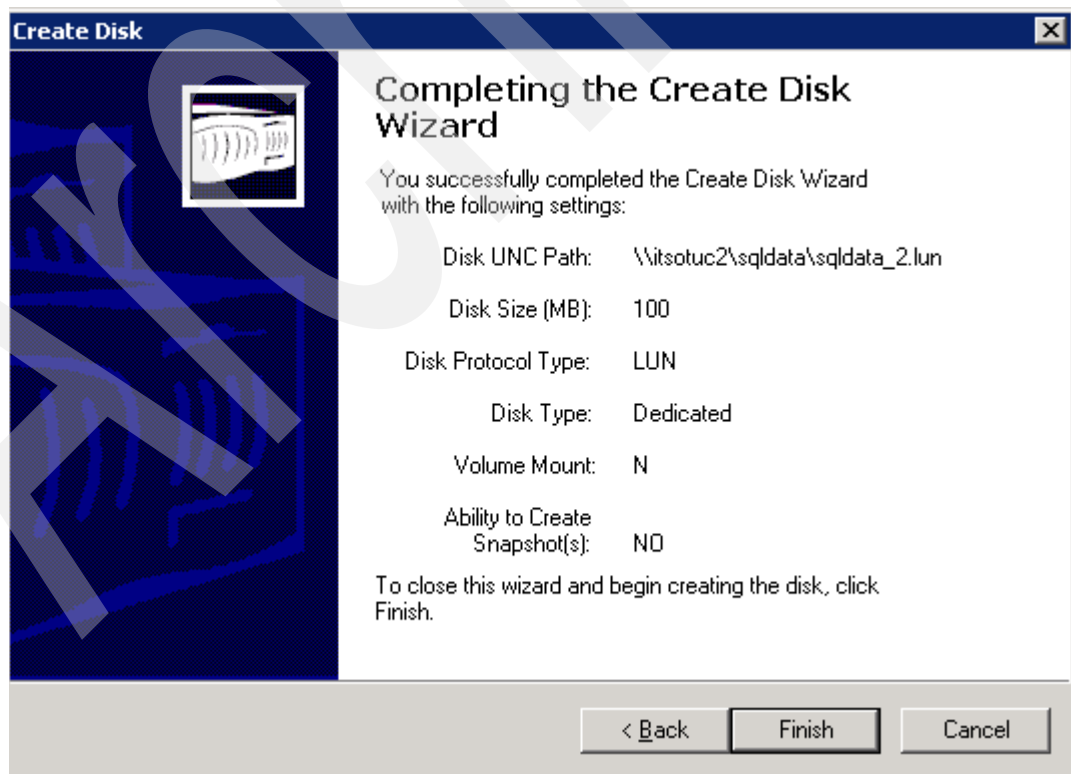


Figure 6-38 Completing the Create Disk Wizard

12. You can check in Explorer to make sure that the disk was created.

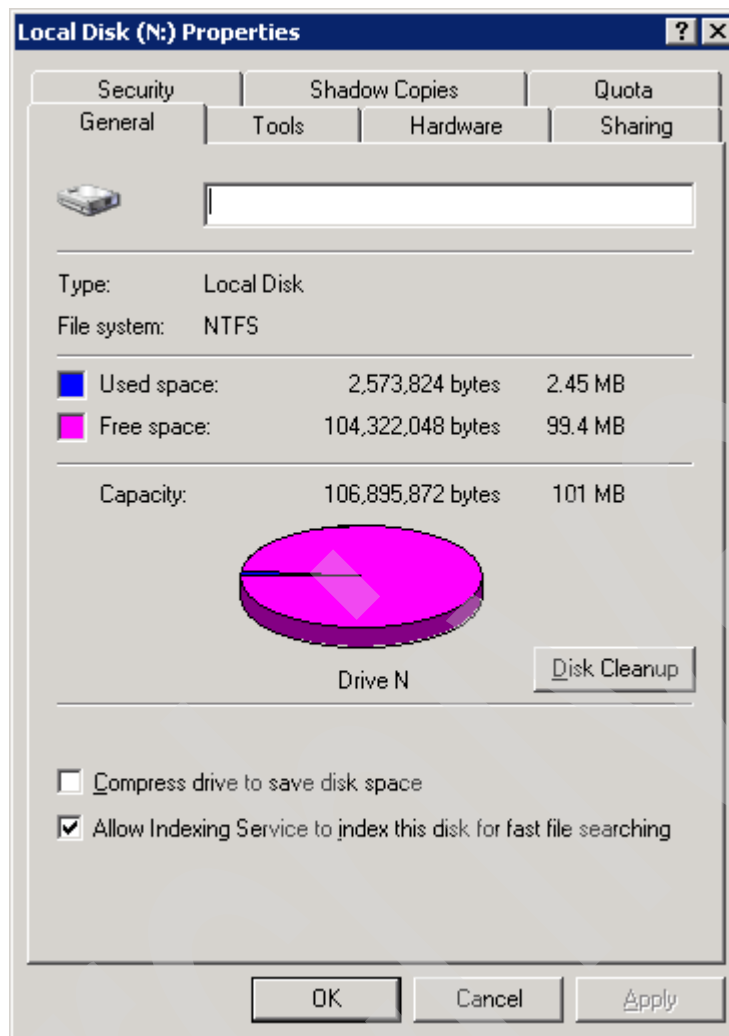


Figure 6-39 Disk created

6.7 Conclusion

To locate resources on a network, a mechanism must exist where the resources can easily be found. A directory service in this case, Active Directory, keeps track of all known resources and responds to requests with a list of currently available devices and services. But before you can be trusted to query for resources, you must be granted membership in the domain. Joining a domain accomplishes two tasks. First, for an IBM System Storage N series, it grants the required rights to query Active Directory should it need to find other resources. Second, it provides a single management interface through MMC for administration of security and users' access levels to the IBM System Storage N series.

Managing Microsoft SQL Server with IBM System Storage N series

7.1 Managing MSSQL with the N series product

For management of SQL Server environments, we recommend that you use the following comprehensive set of solutions:

- ▶ Windows 2000 Server/Advanced Server SP2
- ▶ Windows 2003 Server Standard Edition/Enterprise Edition
- ▶ Microsoft SQL Server 2000 Standard Edition/Enterprise Edition SP2
- ▶ Microsoft SQL Server 2005 Standard Edition/Enterprise Edition
- ▶ Fibre Channel or iSCSI
- ▶ SnapManager for SQL Server 2.0
- ▶ SnapDrive 3.1 or higher
- ▶ SnapMirror
- ▶ FlexClone

7.2 SnapManager

SnapManager for Microsoft SQL Server 2000 (SMSQL) relies on services by SQL Server 2000 and depends on SnapDrive servicing I/O-related requests to the N series product's storage devices. SMSQL easily and rapidly backs up multiple databases in a short time and is an excellent tool for migrating databases to virtual disks using the N series product.

Lightening fast backup and restore with SMSQL 2.0

SnapManager for Microsoft SQL Server 2.0 is an integral part of the complete N series product's data management solution that integrates Microsoft SQL Server with renowned features intrinsic to the N series product's storage technology.

SMSQL benefits

- ▶ Near instantaneous backup and fast restore of entire SQL Server databases and full text indexes (SQL Server 2005) using Snapshot technology.
- ▶ Easy migration wizards to move databases to SAN and IP SAN environments.
- ▶ Integration with SnapMirror for wide area data replication.

SnapManager for SQL Server 2.0 enhancements

- ▶ Full support and integration with Microsoft SQL Server 2005
- ▶ Backup and recovery of Microsoft SQL Server 2005 full text indexes
- ▶ Ability to restore clustered databases without taking the virtual server offline (SnapDrive 3.2 and above only)

New options:

- ▶ Option to skip transaction log backup prior to a database restore operation
- ▶ Option to limit AutoSupport notifications to SnapManager failure events
- ▶ Option to run "UPDATE STATISTICS" on tables before detaching the databases

For a complete overview of SnapManager for Microsoft SQL Server 2.0, see the SnapManager for SQL Server (SMSQL) Installation and Administration Guide located on the following IBM Web support site.

<http://www-304.ibm.com/jct01004c/systems/support/supportsite.wss/supportresources?brandind=5000029&familyind=5329815&taskind=1>

7.3 SnapDrive

SnapDrive software provides storage virtualization of filer volumes via the iSCSI or Fibre Channel file access protocol. The software allows administrators to define virtual disks that are presented to the Windows Server® 2003 or Windows Server 2000 operating system as basic logical disks.

After created, the virtual disks can be completely managed by the SnapDrive MMC plug-in and the Microsoft Windows Disk Administrator MMC plug-in.

In the event of a low disk space situation, SnapDrive provides the ability to expand virtual disks on demand and instantly presents the new disk space to Windows. This capability assists in storage planning by allowing the administrators to add storage as needed without negatively affecting the Exchange services.

Finally, SnapDrive integrates with Microsoft Cluster Services to provide highly available storage solutions in clustered Microsoft Windows environments. Virtual disks stored on an IBM System Storage N Series can host the quorum drive in a Microsoft cluster as well as other cluster resources.

7.4 SMSQL, SnapDrive, and SQL Server 2000

SMSQL is dependent on services provided by SnapDrive and SQL Server. SMSQL executes user requests that have to be coordinated with SnapDrive and SQL Server 2000. See figure 12-1.

1. Full back-up and truncation of transaction logs are synchronized between SMSQL and SQL Server. SQL Server will quiesce database(s) and dump metadata to SnapInfo when a Snapshot copy is created. Backup and truncation of transaction logs are requested by SMSQL but performed by SQL Server, and are dumped to SnapInfo by SQL Server. Database(s) are verified by SQL Server after the virtual disk containing the database(s) is restored as a writable Snapshot file by SnapDrive.
2. SMSQL requests that SnapDrive create a Snapshot backup when a full backup of a database(s) is done. A copy of backed up transaction logs is created if that option is turned on. SMSQL dumps metadata describing backup sets into the SnapInfo directory.
3. If a volume is mirrored, SnapDrive updates the mirror every time a Snapshot copy is created. SnapDrive will restore virtual disks in Snapshot backups as required by database restore operations, or it will restore a virtual disk as a writable copy for database verification.

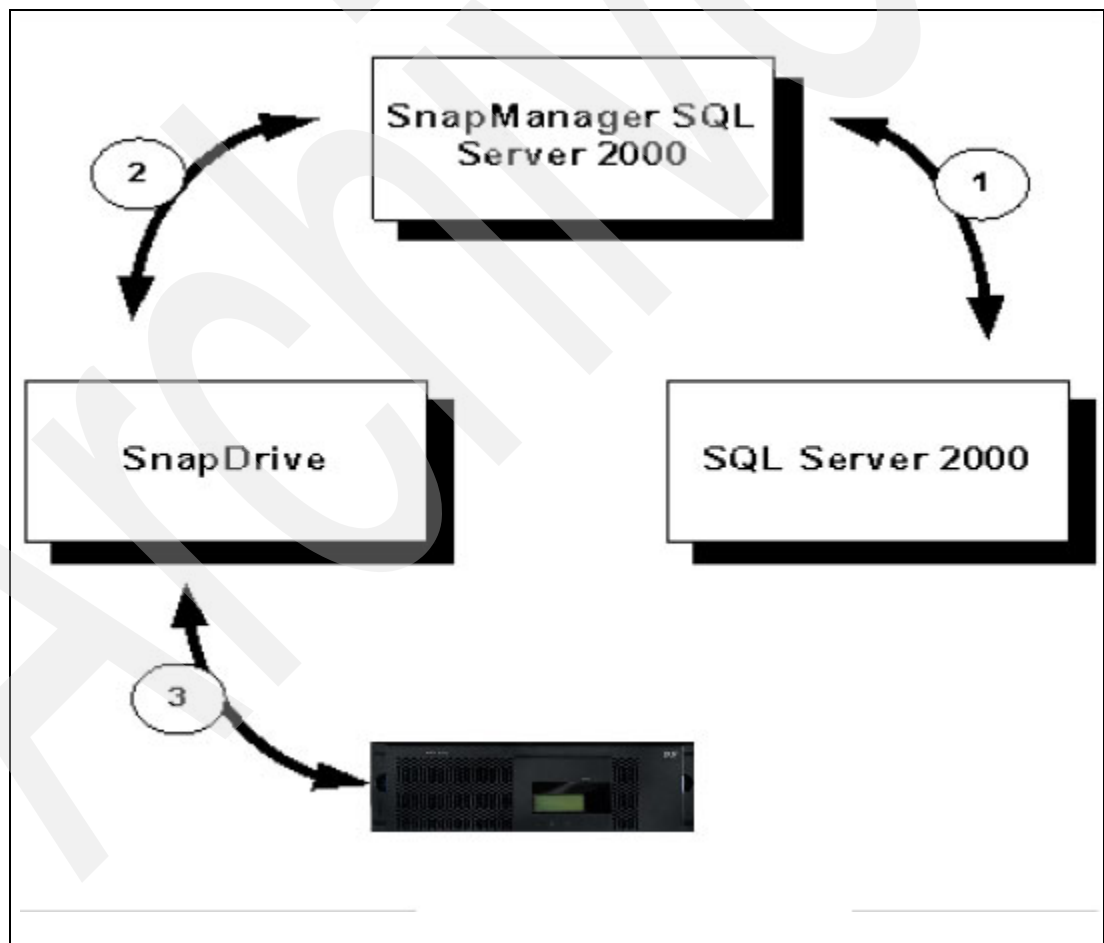


Figure 7-1 SMSQL overview

7.5 Accelerated test and development with rapid cloning

A Snapshot copy of a database in a FlexVol volume can easily be cloned for testing or development purposes (Figure 7-2).

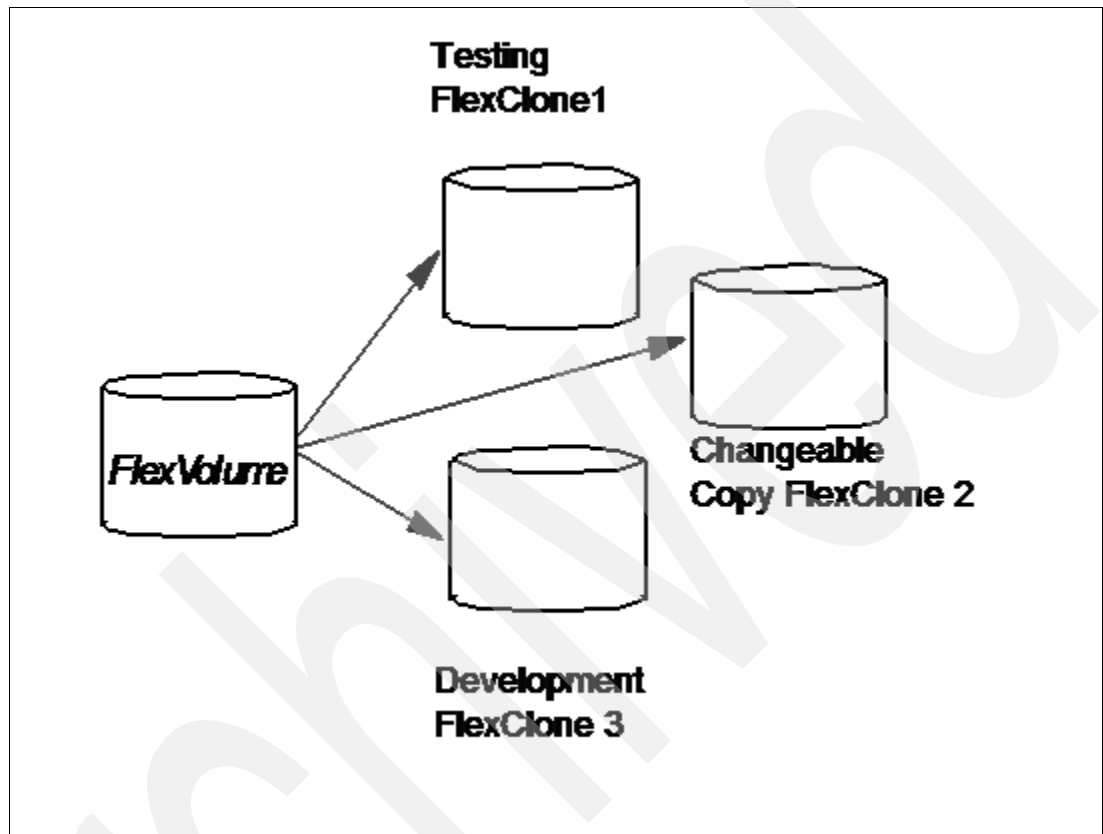


Figure 7-2 Testing with FlexClone

High Availability with MSSQL and N series

In this chapter we introduce basic information about high availability with MSSQL Server and IBM System Storage N series.

Organizations today depend on digital data to function, and any event that suspends the flow of information can effectively halt business operations. Minutes of downtime are costly, and hours of downtime can be catastrophic.

N series enables disaster recovery plan with simple, yet powerful, highly-dependable solutions that deliver data security, high availability, and, in case of disruption, rapid and complete data recovery.

High availability design

The IBM System Storage N series is uniquely scalable and highly available with the ability to scale from 2TB all the way up to 100TB. The N series product offers a rich set of features with inherent redundancy of robust hardware components. Clustered IBM System Storage N series is available for even higher levels of availability. Innovations like RAID Double Parity (RAID-DP) enhance availability by protecting against secondary drive failures that may occur during RAID reconstructs. For more information about RAID-DP, please refer to Redbooks publication *IBM System Storage N series*, SG24-7129. The SnapDrive family of products offers Mutli-Path I/O (MPIO) solutions for both iSCSI and Fibre Channel protocols, providing redundant paths from hosts to the N series product. For more information about MPIO, please see the Redpapers publication *IBM System Storage N series MPIO Support Frequently Asked Questions*, Redp4213.

8.1 Backup and restore databases

Backup is an important safeguard for protecting critical data stored in SQL Server databases. Backing up and restoring databases is useful for other purposes, such as copying a database from one server to another. By backing up a database from one computer and restoring the database to another, a copy of a database can be made quickly and easily.

8.1.1 Backup databases

The methodology used to back up databases is indirectly decided when the databases' physical layouts are designed and implemented. Full backup of user-defined databases through SMSQL is implemented using volume-level Snapshot copies, which means that all databases utilizing virtual disks located in the same volume are backed up together. Databases sharing a virtual disk are backed up together with all other databases sharing the same volume, except if the virtual disk contains a system database. In this case, full backup is streaming-based and is done per database, independent of other databases using the virtual disk or volume. This is not recommended since streaming-based full backup is slower than Snapshot technology-based. Full backup and streaming based backup take much more space in the related SnapInfo directory.

Backup can be done with the help of the backup wizard or the SnapManager Backup tab. The crucial panel is the same with both tools and is shown in Figure 9-1 on page 158. This panel has the following interesting points:

- ▶ Only databases located on virtual disks can be backed up by SMSQL.
- ▶ Databases using virtual disks located in the same filer volume can only be backed up together (pubs, Northwind, and test in Figure 9-1 on page 158). If you check or uncheck one of the databases, the action affects all databases utilizing the same volume.
- ▶ A transaction log is logically truncated when it is backed up and not when a full backup is created.
- ▶ Full backup can be followed by a backup of the transaction log.
- ▶ Full backups use Snapshot and are quick.
- ▶ Transaction log backup is streaming-based and is done by SQL Server.
- ▶ Backup of system databases is streaming-based. See Figure 9-2 on page 159.
- ▶ Streaming-based backup is used when a used database is sharing a virtual disk with a system database.
- ▶ Backups can be automatically scheduled (set up through the Schedule button). The SQL Server Agent has to be running if it is scheduling backup jobs.

All backup sets take space in their SnapInfo directory for metadata, backed up transaction logs, and streaming based backed up databases:

- ▶ Metadata per backup set is only a few kB.
- ▶ The size of a backed up transaction log depends mostly on the number of new transactions since the last transaction log backup.
- ▶ The size of a streaming-based full backup is roughly the same size as the online database.

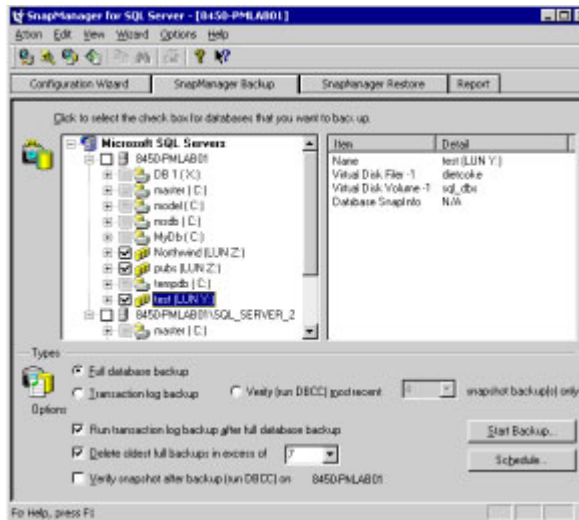


Figure 8-1 Main backup panel

Correct backups are critical for recover of a corrupt database; therefore, it is strongly recommended to daily check new backup reports for possible errors. An example of a report is shown in Figure 9-2 on page 159.

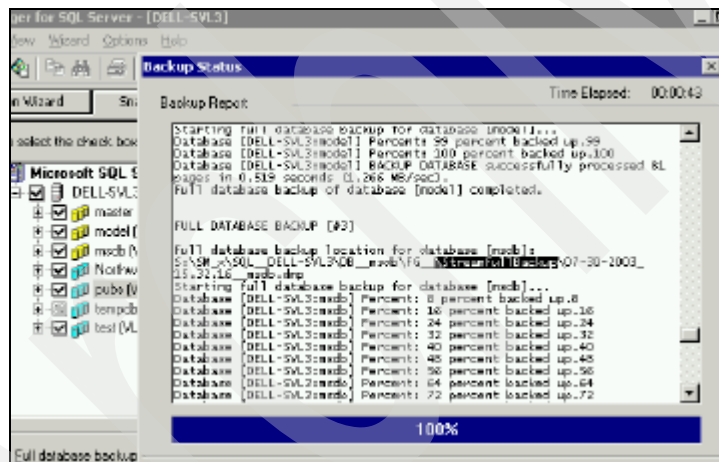


Figure 8-2 Report of System Database backup

8.1.2 Restore databases

It is critical to be able to restore a corrupt database quickly and efficiently when needed. It is never possible to plan when a disaster will occur; therefore, we strongly recommend that you test the implemented backup and restore processes occasionally. The process is indirectly decided when the database's physical layout is implemented. Databases that are not sharing any virtual disks with another database can be restored without any effect on any other database. This part discusses strategies for restoring a single database stored in a virtual disk and how to restore all databases stored in a virtual disk.

The SnapManager Restore Wizard or SnapManager Restore tab can be used for restoring databases. Both tools use the same central panel, as shown in Figure 9-3 on page 159.

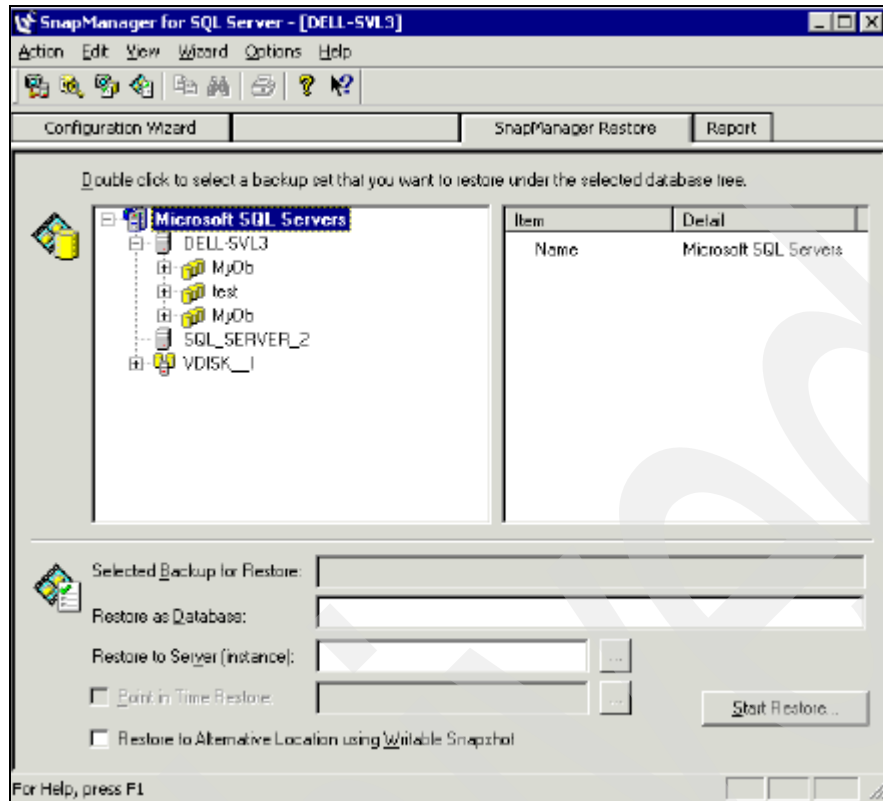


Figure 8-3 Main Restore panel

The structure of SnapInfo is recognizable in Figure 8-3. The top level is either a SQL Server instance or a virtual disk:

- ▶ SQL Server instance: DELL-SVL3 with a number of backed up databases
- ▶ SQL Server instance: SQL Server_2 with no backed up databases
- ▶ A virtual disk: VDISK_I

Expanding VDISK_I (Figure 8-4 on page 141) shows an overview of databases that can be restored. Restoring all databases in a virtual disk is straightforward and is probably quicker in many cases than restoring a single database.

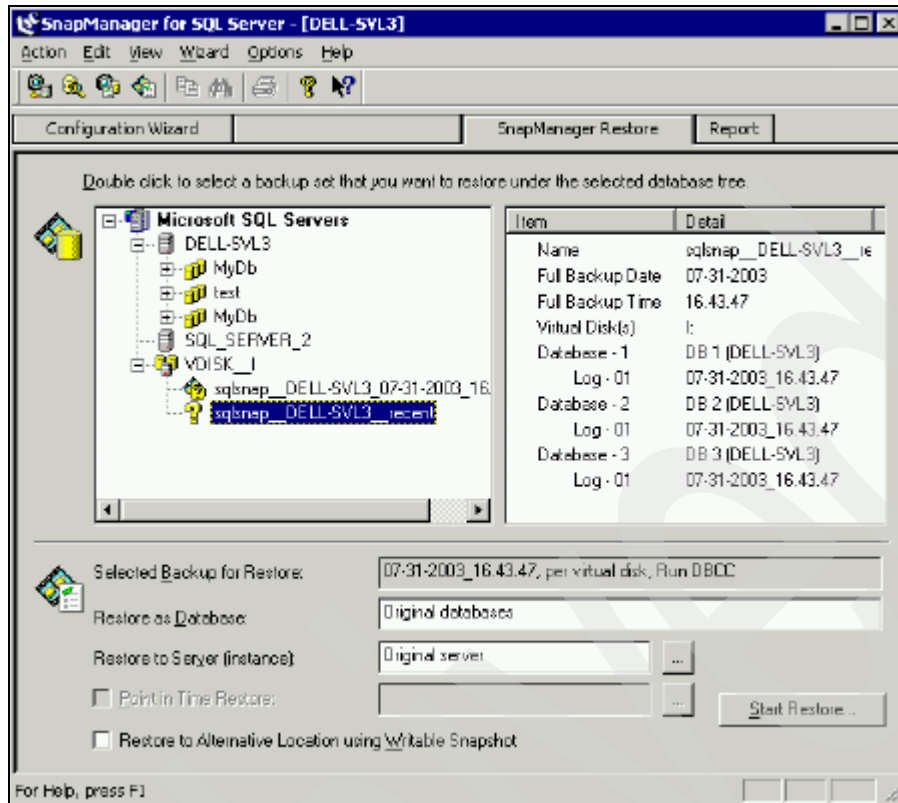


Figure 8-4 Main Restore panel

Restoring a single database in a virtual disk requires some manual steps. The process begins with the same screens as shown in Figure 8-3 on page 140:

1. Select the virtual disk with the database to be restored.
2. Select the option Restore to Alternative Location using Writable Snapshot.
3. Click StartRestore. The panel in Figure 8-5 on page 142 makes it possible to select only the database to restore.

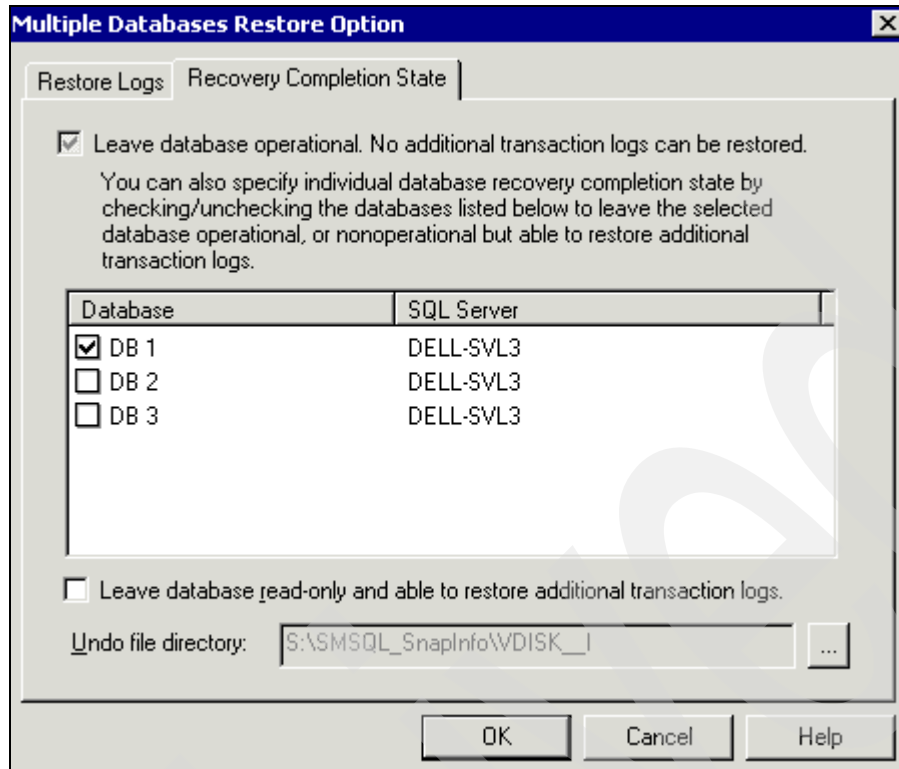


Figure 8-5 Restoring multiple databases

4. Click OK.
5. Start restore.
6. Wait for Restore Complete message.

The result of the restore is as follows:

- ▶ The Snapshot is restored and mounted on an unused drive letter.
- ▶ The selected database is online but renamed DB 1__Mount.
- ▶ The deselected databases are in loading state.

The following four additional steps have to be executed before the restored database is ready for use:

1. Start Enterprise Manager, and browse to the database(s).
2. Delete databases that are not to be restored:
DB 2__Mount and DB 3 __Mount (Figure 8-6 on page 143).
3. Close Enterprise Manager.

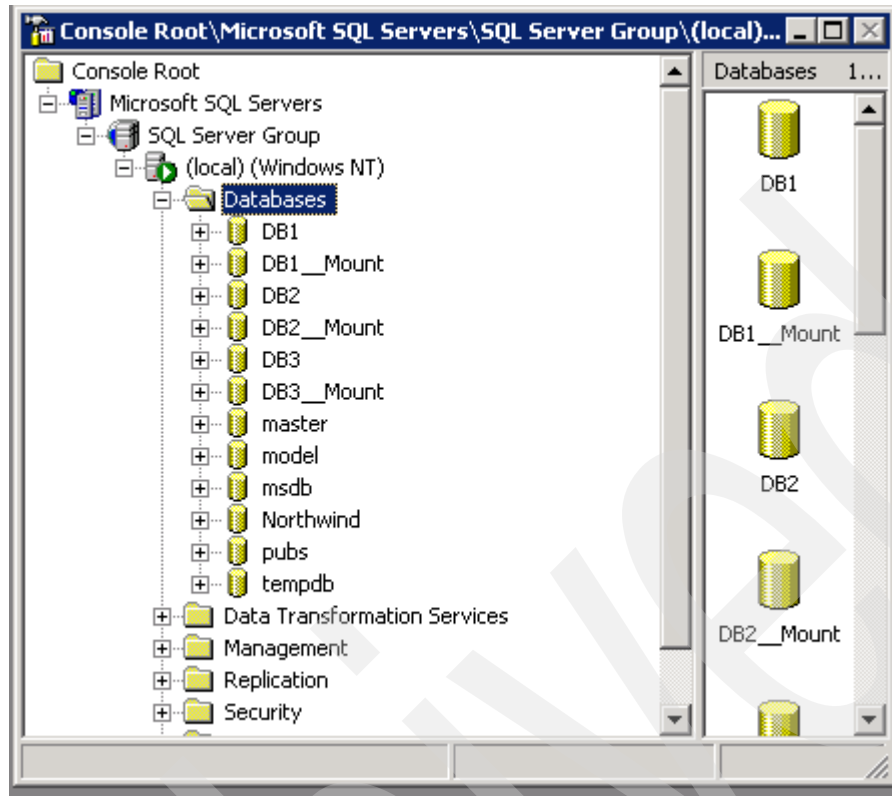


Figure 8-6 Enterprise Manager

4. Start Query Analyzer and rename the database. See Figure 8-7 on page 144.
5. Close Query Analyzer.

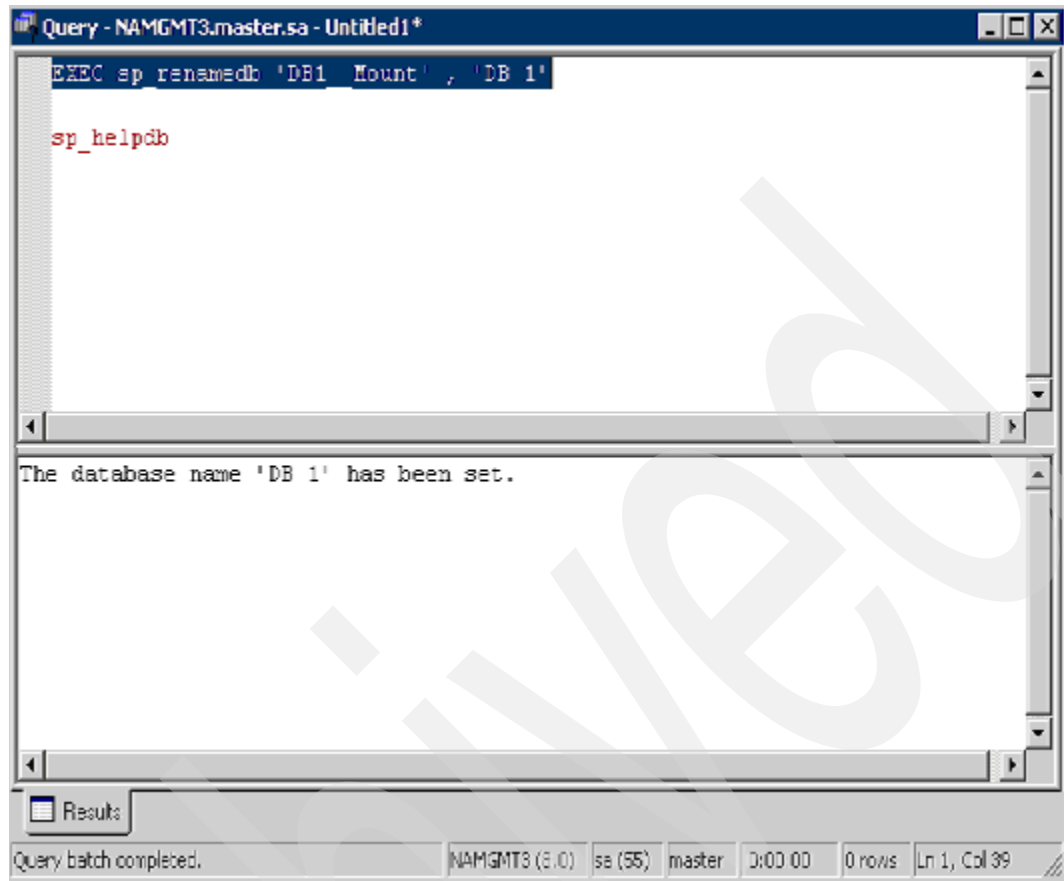


Figure 8-7 Rename database

6. Migrate the restored database back to the original location with SMSQL. See Figure 8-8 on page 145.

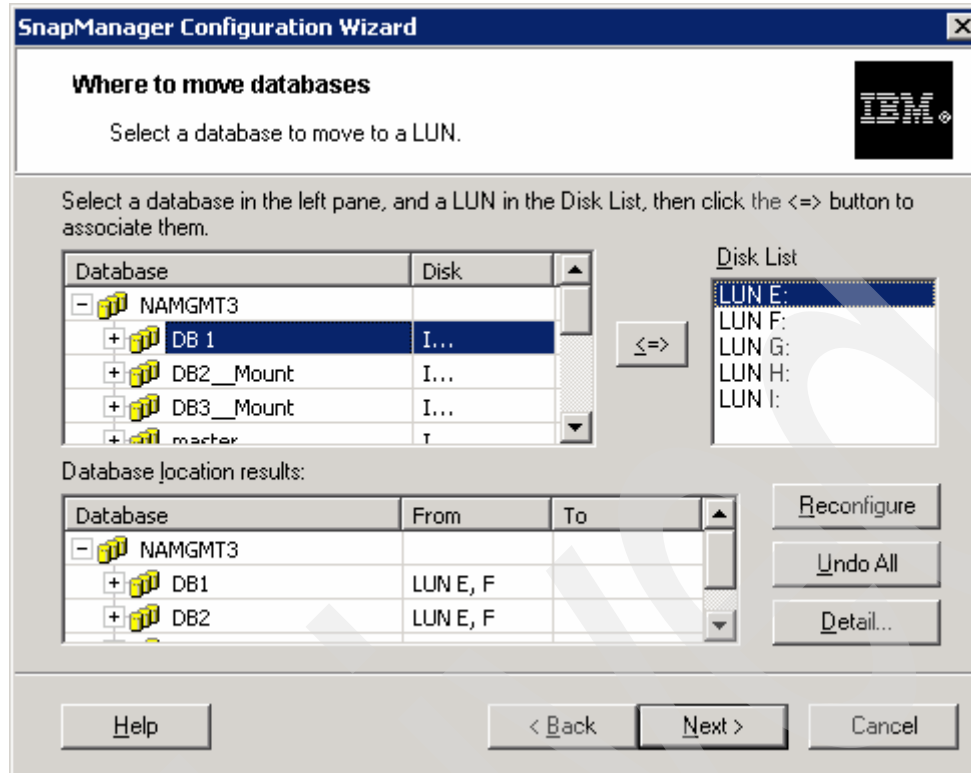


Figure 8-8 Migrate restored database

- Open Computer Manager, and disconnect the writable Snapshot copy. See Figure 8-9. The writable Snapshot copy was mounted on X in this case.

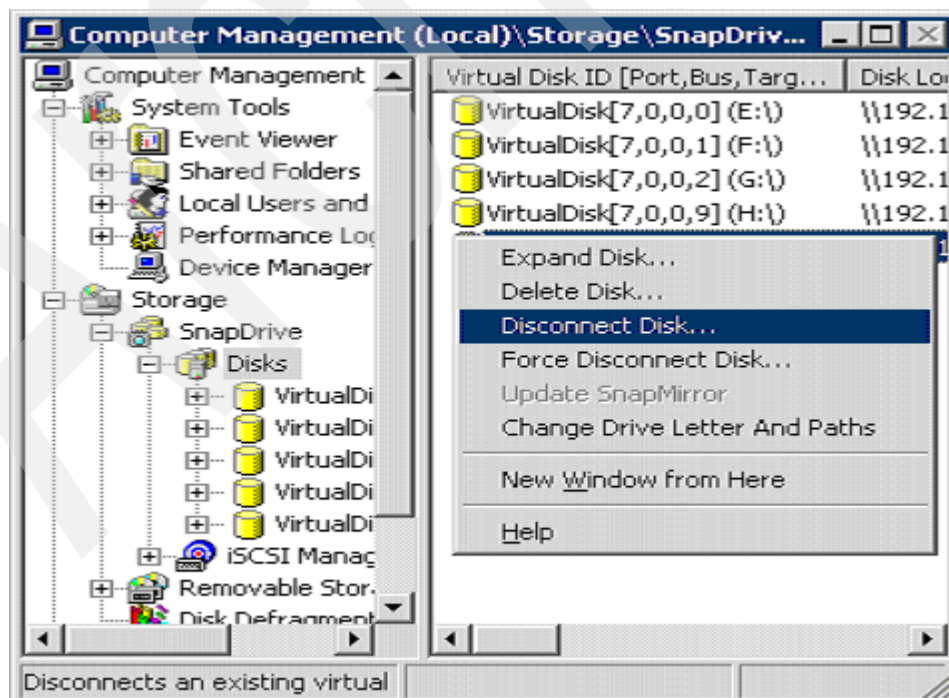


Figure 8-9 Disconnect Writable Snapshot Backup

8.1.3 Backup and restore best practices

In this section, we reiterate specific areas that have to be well understood.

SMSQL will back up

- ▶ Snapshot copies are created of all volumes used by the databases being backed up.
- ▶ System databases are backed up using conventional streaming-based backup.
- ▶ User databases placed in a LUN that also contain system databases are backed up using streaming-based backup.
- ▶ Backup of transaction logs are always use streaming-based backup to provide point-in-time restores.

SMSQL will restore database(s)

SMSQL restores databases by doing the following:

- ▶ SnapDrive will request a Single_File_SnapRestore to restore all LUNs used by the database(s).
- ▶ Restoring selected databases out of many sharing one or two LUNs.
- ▶ Requesting SQL Server to restore the databases.
- ▶ Requesting SQL Server to apply selected backed-up transaction logs. This is configurable and can conduct either point-in-time or up-to-the-minute restores.
- ▶ The most efficient configuration is one user database per LUN.

Backup recommendations

The SMSQL full backup using Snapshot is quick and does not adversely influence database performance, compared to traditional backup with SQL Server BACKUP statements. Transaction log backup is functionally the same using SMSQL as the transaction log backup done using BACKUP Log statements and is streamed out to the SnapInfo directory for safekeeping. The length of time required to back up a transaction log depends mainly on the active section of the transaction log that has to be extracted and copied to the dump directory. When planning the backup and restore process, the methodology or considerations are different from considerations when backup is done by Enterprise Manager or SQL Server Management Studio. Following are the two main considerations:

1. Recovery point objective (RPO) - To what point in time does the data need to be recovered?
2. Recovery time objective (RTO) - How long will it take to the get database back online and rolled forward or backward to the RPO?

Consider the following hypothetical customer scenario:

- ▶ Customer: ZYX Electronics
- ▶ Database: CommerceDB
- ▶ Failure Type: Database corruption or deletion
- ▶ RTO: It cannot take longer than 30 minutes to complete a restore
- ▶ RPO: Data should be brought back to a time no later than 15 minutes of failure

ZYX Electronics's IT group has shown that a full backup every 30 minutes, with a transaction log backup every 15 minutes, on the hour makes it possible to restore the database in 20 minutes (without the verification process) to the minimum recovery point objective of 15 minutes. Backup every 30 minutes during a 24-hour time frame will create 48 Snapshot

copies and as many as 98 transaction log archives in the volume containing SnapInfo. An IBM System Storage N series volume can host up to a maximum of 255 Snapshot copies, so the proposed configuration could retain over five days of online Snapshot data.

Thus, it is generally a good idea to categorize all your organization's systems that use SQL Server in terms of RTO and RPO requirements. After these requirements are established, SMSQL can be tailored to deliver the appropriate level of protection. Proper testing before deployment can also help take the guesswork out of what can be realistically achieved in production environments.

In situations where business requirements are not entirely clear, refer to the recommended default database backup frequency below.

Minimum backup frequency

All backup and recovery processes should be created according to each environment's unique requirements, but when in doubt, the general recommendation is to back up the database every hour and the transaction log every 30 minutes during production time and possibly less often during off-peak hours.

Always archive or mirror Snapshot copies

We strongly recommend archiving or mirroring backup sets as soon as possible. Disasters do occur, and if the storage device containing the databases and backup Snapshot images is adversely affected, it will not be possible to restore the databases from the primary storage system. Archive media can be tape, another IBM System Storage N series, or even local SATA drives available on N series. The archive process is described in further detail in the SMSQL Installation and Administration Guide.

8.2 Mirroring

Many organizations have requirements for mirroring SQL Server databases to geographically dispersed locations for data protection and business continuance purposes. Like Snapshot, SnapMirror works at the volume level, so proper grouping of databases by the high-availability index can be particularly useful. Locating one database per FlexVol volume provides the most granular control over SnapMirror frequency, as well as the most efficient use of bandwidth between SnapMirror source and destination. This is because each database is sent as an independent unit without the additional baggage of other files such as tempdb or databases that may not share the same requirements. While sharing of LUNs and volumes across databases is sometimes required because of drive letter limitations, every effort should be made to group databases of like requirements when specific mirroring and backup policies are implemented.

Using IBM System Storage N series SnapMirror Technology with MSSQL

The SnapMirror technology provides advantages for the SQL Server DBA seeking to support disaster recovery of a mission-critical.

Network Prerequisites

A network connection is required between the SQL Server machine and the storage system. 100BaseT, FDDI, and Gigabit Ethernet are all viable options. The faster the network, the better the performance. A robust network connection between the source filer and the target filer is required. It must have sufficient bandwidth to accommodate the anticipated data

change rate and SnapMirror software overhead. The network connection type should be based on the following parameters:

- ▶ Data transmission costs between the source and target filers.
- ▶ The source volume size.
- ▶ The data change rate.
- ▶ The SnapMirror software update schedule in addition to the data change block transfers. Each replication event requires an updated WAFL block map defined as 0.1% of the source volume size. The block map file is transferred for each mirror iteration regardless of the number of changed blocks. For example, a 250 GB volume mirror will transfer a 250MB block map file plus all changed data blocks since the last Snapshot.

Setting Up a SnapMirror Process for SQL Server 2000 Data File Storage

We will use Figure 8-10 in the following discussions about setup and modifications.

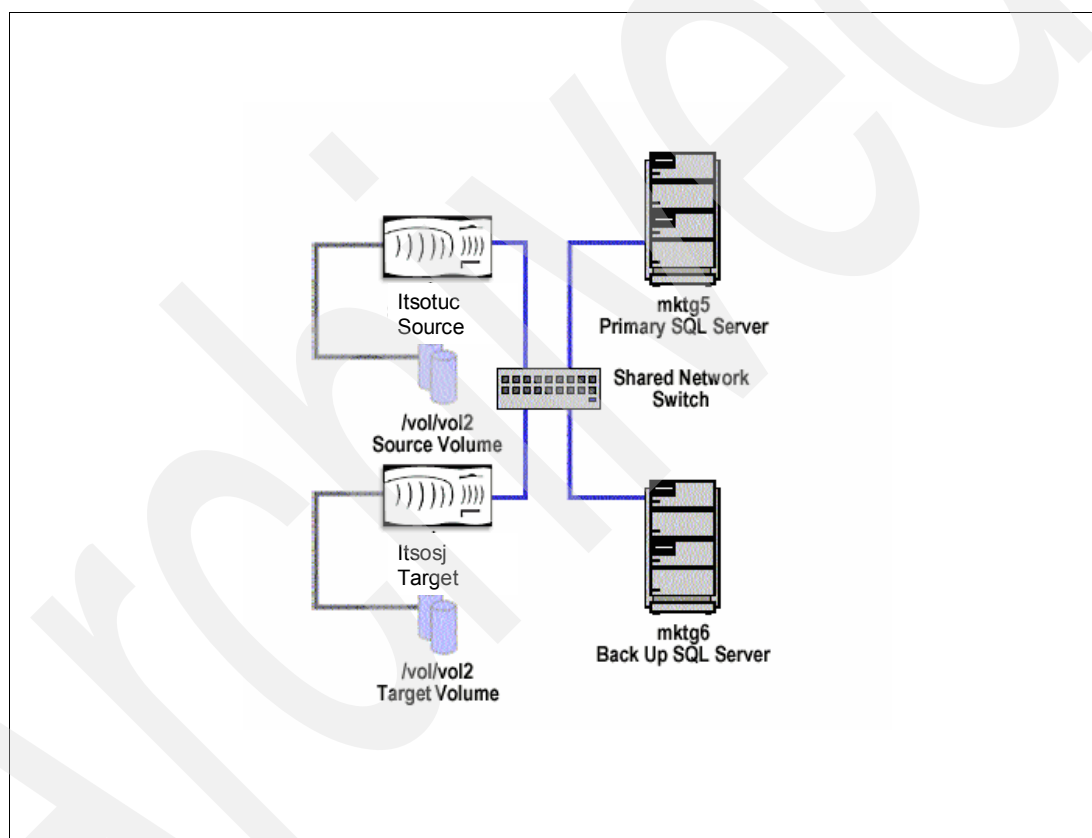


Figure 8-10 Configuration example

Mirrors are as simple to set up and manage as filer volumes. After all pre-requisites are met, the following must be done to initiate the process.

1. (Implemented on the Source storage system): Add the host name of the destination filer to the /etc/snapmirror.allow file on the source storage system. For example, itsosj is the name of the destination storage system, so the /etc/snapmirror.allow file on the source system (itsotuc1) appears as follows (this file can be viewed by mapping a drive to the root volume of the filer and looking in the /etc directory):

Note: The storage system is not shipped with a default /etc/snapmirror.allow file. Rather, it must be created using a text editor.

2. (Implemented on the Destination storage system itsosj): Specify the following in the /etc/snapmirror.conf file on the destination filer:
- The source storage system and volume
 - The target filer and volume
 - The maximum network bandwidth usage throttle
 - A list of incremental update times in minutes, hours, days, and months
- For example, the /etc/snapmirror.conf file on the destination storage system itsosj appears similar to the following example.

Example 8-1 snapmirror.conf file

```
itsotuc:vol2 lg:vol2 kbs=5000 * * * *
```

See step one for how to view this file.

The text in Example 8-1 sets up the following SnapMirror software parameters:

- ▶ The vol2 volume on bg will be replicated onto the vol2 volume on itsosj.
- ▶ A 5,000 KB per second maximum throttle is set on the mirror.
- ▶ The SnapMirror interval is set to occur every minute of every hour of every day of every week of every month.

▶ **Note:** The filer is not shipped with a default /etc/snapmirror.conf file. Rather, it must be created using a text editor.

3. (Implemented on Both the Source and Destination storage systems): At the command line of both the source and destination filers, enter the following command:

vol snapmirror on

It is important to note that the vol snapmirror on command does not persist across storage system reboots. Rather, the command must also be put in the /etc/rc files of both source and destination filers.

Note: This command can be placed anywhere after the network interfaces are defined.

If placing the vol snapmirror on command in the /etc/rc file, Example 8-2 shows how this file may appear (see Step One for how to view this file):

Example 8-2 /etc/rc file

```
#Regenerated by registry Wed Apr 21 10:10:28 PDT 1999
#Auto-generated by setup Mon Apr 19 18:22:22 GMT 1999
hostname itsotuc
ifconfig e0 `hostname` -e0 mediatype auto netmask 255.255.252.0
route add default 10.153.4.1 1
routed on
options dns.domainname 2700-1.ibm.com
options dns.enable on
```

vol snapmirror on

With SnapMirror turned on, the destination filer will read the `/etc/snapmirror.conf` file and, at the next scheduled mirror update, establish a connection with the source filer. If a baseline version of the mirror does not exist, the filer takes a Snapshot of the source volume and transfers all the data in the Snapshot from the source volume to the mirror. If a `vol status` command is issued at this point, the following result will appear (noting that, in this example, `vol2` is our mirror volume):

Example 8-3 vol status

```
itsotuc>vol status
Volume State Status Options
vol0   online normal root, nosnap=on
vol1   online normal
vol2   online snapmirrored
```

8.2.1 SnapMirror software operation

SnapMirror has two distinct phases: initialization and incremental update. The initialization phase consists of a level 0 replication event in which a Snapshot is created on the source volume and then sent in its entirety to the target volume. The amount of time required to replicate the entire source volume over to the target volume depends on many factors, including the network connection. For example, a 250 GB mirror initialization that would take approximately 7.5 hours to complete across a 100 BaseT full-duplex link may take 1.5 hours across a gigabit link. The level 0 event serves to initialize or "seed" the mirror volume since it contains every block in the source volume as of the time the Snapshot was created.

After mirror initialization is complete, the target filer(s) examines its `/etc/snapmirror.conf` file every minute to see if there are any scheduled updates. This allows for modification of the mirror's configuration without disrupting the mirror. When an incremental update schedule time is due, a new Snapshot is taken and compared to the previous Snapshot. The different blocks and block map file are sent to the mirror target. In contrast to the level 0 initialization, the data mirrored is typically much smaller. Note that at all times the mirror target file system is in a consistent state.

Several special cases should be noted

- ▶ If an initialization (level 0) replication event is interrupted for more than nine minutes (for example, a network outage), it will abort. Partial level 0 events are not recoverable, so the process must be restarted.
- ▶ The KB per second maximum throttle parameter in `/etc/snapmirror.conf` can be changed at any time, and the edited values take effect within two minutes. The exception to this is mirror initialization where a bandwidth throttle is already in effect. In this case, the bandwidth throttle cannot be modified until the initialization phase is complete or the process is interrupted.
- ▶ Incremental updates do not start until the level 0 initialization is complete.
- ▶ Any incremental updates that are missed are simply skipped. Subsequent incremental updates transmit any skipped data, so no data is lost.

- Incremental updates in progress run to completion according to the configuration settings and available network bandwidth. If a new incremental update is scheduled to start while an existing update is in progress, it is considered a schedule miss and this is skipped.
- The SnapMirror process can be stopped and restarted at any time by issuing the command sequence in Example 8-4 on either storage system.

Example 8-4 snapmirror commands

```
vol snapmirror off
[arbitrary time interval]
vol snapmirror on
```

As long as the target volume remains read-only during the time between turning the SnapMirror process off and on, the mirror remains intact.

- To break the mirror, SnapMirror must be turned off, at which time the target volume is placed into read/write mode. In order to turn SnapMirror off, the command in Example 8-5 should be executed on the destination storage system.

Example 8-5 Turning SnapMirror off

```
vol options vol_name snapmirrored off
```

8.2.2 Disaster recovery breaking the mirror

After creating a SnapMirror relationship between two storage systems we implemented the following test:

1. The following table was created on the database test_db. The database's data and log files were located in the mssql directory on vol2 on itsotuc:

Example 8-6 Table creation

```
DROP TABLE hammer_tab;
CREATE TABLE hammer_tab
( value INT NOT NULL );
```

2. We then created the following stored procedure in the table hammer_tab:

Example 8-7 Creating stored procedure

```
USE test_db
GO;
CREATE PROCEDURE add_data
AS
DECLARE @counter INT
SELECT @counter =
WHILE ( @counter < 100000000000 )
BEGIN
INSERT INTO hammer_tab
( value )
VALUES
( @counter );
PRINT 'Value is: ' + convert ( varchar(6), @counter1 )
SELECT @counter = @counter + 1
END
```

3. We then executed `add_data` from the SQL Server's Query Analyzer. With the procedure running, we allowed several SnapMirror events to occur.
4. Toward the end of the ten-minute procedure run, we changed the SnapMirror interval in the `/etc/snapmirror.conf` file to a less frequent mirror. We did this because we were about ready to simulate a filer/SQL Server failure followed by a recovery of both servers. We did not want a SnapMirror event to initiate once the filer recovery completed. If an event was triggered, we would not be able to break the mirror.
5. Next we simulated the above-mentioned disaster with a goal of mimicking a complete loss of service of both the filer and the database server. We did this in the following manner:
 - Failed the source filer (bg) by powering it down
 - Failed the SQL Server by turning off the MSSQLServer service while the query was still running

The failure of both the storage and the SQL Server resulted in the following activity (Example 8-8) from within the Query Analyzer (note that the last row inserted has a key value of 150942):

Example 8-8 Query Analyzer activity

```
(1 row(s) affected)
Value is: 1 (1 row(s) affected)
Value is: 2 (1 row(s) affected)
Value is: 3 (1 row(s) affected)
Value is: 4
... Many more like this ...
(1 row(s) affected)
Value is: 150939
(1 row(s) affected)
Value is: 150940
(1 row(s) affected)
Value is: 150941
(1 row(s) affected)
Value is: 150942
[Microsoft] [ODBC SQL Server Driver] [Named Pipes] ConnectionRead
(GetOverLappedResult()).
[Microsoft] [ODBC SQL Server Driver] [Named Pipes] Connection broken.
Connection Broken
```

6. After changing the mirror interval, the mirror on vol2 was broken (making itsosj's vol2 read/write) by issuing the command in Example 8-1 on page 149 on itsosj:

Example 8-9 vol options command

```
vol options vol2 snapmirrored off
```

7. We then created a CIFS share (sqldb) pointing to the directory (mssql) on the SnapMirror volume (vol2 on bg) in which the data resides.
8. After creating the share, we created a new SQL Server database (test_db_new) that pointed to the data and log files that were on itsosj's now read/write volume vol2. We did this by executing the following statement from an ISQL prompt:

Example 8-10 Creating SQL database

```
dbcc traceon ( 1807 )
go
sp_attach_db 'test_db_new', '\\lg\sqldb\test_db_data.mdf', '\\bg\sqldb\test_db_log.ldf'
```

```
go
dbcc traceoff ( 1807 )
```

9. Then hammer_tab table in both test_db and test_db_new was queried to ascertain if the data in the table on each database matched. Following are the query scripts and their respective result sets:

Query on test_db:

Example 8-11 Query on test db

```
USE test_db
SELECT * FROM hammer_tab
```

Result set from query on test_db:

Example 8-12 Query results

```
value time
1
2
3
4
...Many more rows incremented by one ...
150939
150940
150941
150942
(150942 row(s) affected)
```

Query on test_db_new:

Example 8-13 Test db command

```
USE test_db_new
SELECT * FROM hammer_tab
```

Result set from query on test_db_new:

Example 8-14 Query results

```
value time
1
2
3
4
...Many more rows incremented by one ...
130941
130942
130943
130944
(13944 row(s) affected)
```

Note that the key value for the source database matches precisely to that which was reported to the client. When an IBM System Storage N series experiences a dirty shutdown, no transactions are lost, although the SQL Server may apply a recovery to the database on the source filer (itsotuc). However, the target database did lose approximately 20,000 rows over

the ten minutes that the test was run. This is to be expected. The mirror interval was set to 60 seconds, and the mirror relies on consistency points. Thus, the mirror can be as much as two minutes out-of-date with the interval set in this way. (A longer interval would result in more lost transactions, but also lower overhead.) In our example, a total 150,942 rows were processed on the source database table. This is a rate of 15,094.2 rows per minute. Given that the mirror can be out-of-date by minutes, the number of rows that did not mirror over to the target database table could have been as high as 30,188.4 (20% of the table). The bottom line is that the target database can be created as a mirror of the source database, with a loss of only a few very recent transactions.

8.2.3 Resyncing the Mirror

At the time of this publication the ability to restart a broken mirror is not available. Presently, the method for reestablishing a mirror is to reinitialize the mirror. If the target volume was brought online and then written to, this may need to be done twice.

- Once to replicate the data back to the source volume. (In this case the mirror relationship between the two storage systems is reversed.)
- Once to reinitialize with the mirror relationship back to normal.

An alternative is to back up the target volume's data to tape, and then restore that data on to the source storage system. This avoids the need to reinitialize the mirror twice; however, you currently cannot "seed" a mirror from tape.

8.2.4 Conclusions

The N series's SnapMirror technology provides compelling advantages for the SQL Server DBA seeking to support disaster recovery of a mission-critical SQL Server database. Specifically the following:

- The network traffic generated by the mirror process can be throttled to allow support for WAN connections.
- The mirror interval is user-configurable and can be changed on-the-fly.
- In the event of a disaster on the source side of the mirror, the target side can be easily brought online, and the SQL Server database can return to normal use after a brief recovery process.

The near real-time nature of SnapMirror software means that the source storage system will still operate normally when the network link to the target filer is broken. This is not true of synchronous mirroring, where the mirror link is essential to continued operation of the source machine.

Using SnapMirror technology, the DBA can assure the customer that the only transactions that will be lost are those that occur between the mirror events.

8.3 About Snapshots and SQL Server

Snapshot technology is a patented space-efficient, high performance method of creating point-in-time replicas of data on disk. They provide a means of creating a backup image of data at a particular point-in-time. Snapshot backups occur in a matter of seconds and typically each copy consumes only the amount of data that changed since the last copy was created. Thus Snapshots consume minimal disk space while providing up to 255 online point-in-time images. A related capability, SnapRestore, can recover a previous point-in-time

image almost instantaneously. These features, together with two host-side software plug-ins can dramatically simplify and automate the backup and recovery of SQL Server databases. Figure 8-10 on page 148 illustrates a typical configuration.

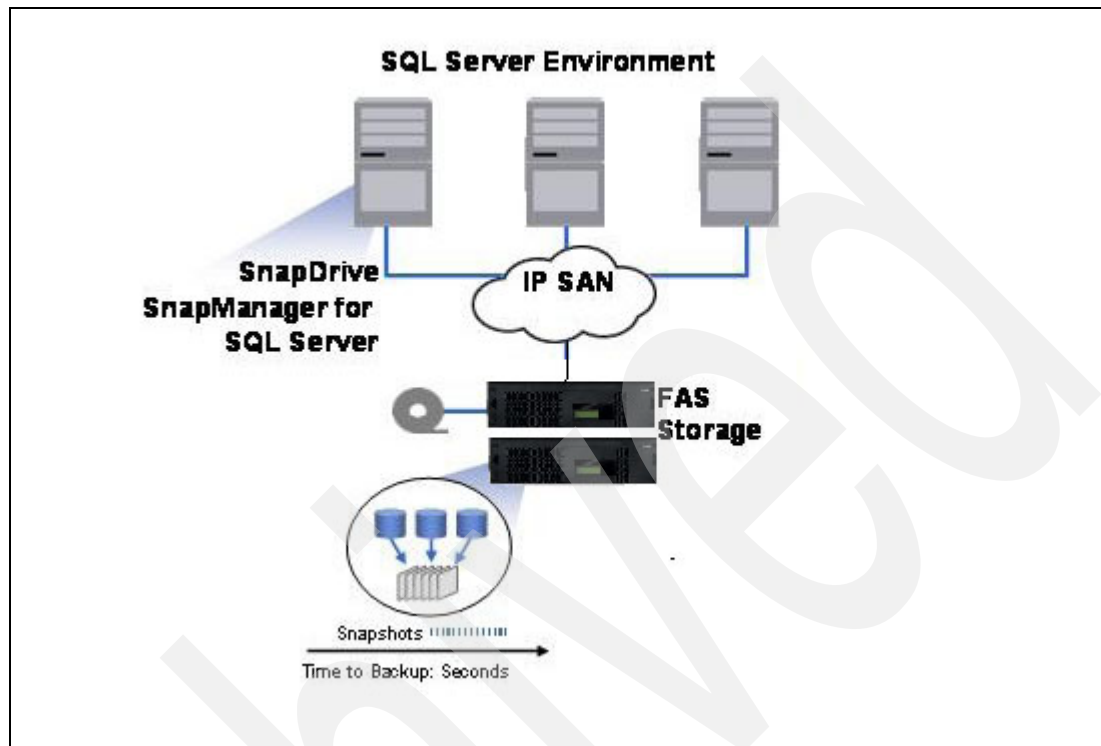


Figure 8-11 Simplifying and automating SQL Server backup and recovery

Best practices

Because SnapManager leverages volume-level Snapshot copies, all user databases residing in a given volume are backed up by the Snapshot copy. This becomes apparent in the SMSQL GUI when attempting to back up a single database sharing a volume with other databases. All databases residing in the volume get selected for backup regardless of unique backup requirements and cannot be overridden. The only exception to this is the system databases, which are handled by streaming backup, but even they are included in the Snapshot backup when sharing a volume with user databases.

Transaction log backup settings are also specific to the entire volume, so if you just want to dump the log, for say one or two databases in a volume hosting numerous databases, you will be forced to back up all transaction logs for all databases. Please review the following best practice guidelines.

Use FlexVol volumes when more granular control is required

When backup requirements are unique to each database, there are two possible paths that can help optimize snapshots and transaction log backups:

- ▶ **Most Granular Control:** Create separate FlexVol volumes for each database for more granular control of backup schedules and policies. The only drawback to this approach is a noticeable increase in administration and set up.
- ▶ **Second Most Granular:** Try to group databases with similar backup and recovery requirements on the same FlexVol volume. This can often provide similar benefits with less administration overhead.

Avoid sharing volumes between hosts

With exception to MSCS members, we do not recommend letting more than a single Windows server create and manage LUNs on the same volume. The main reason is that Snapshot copies are based at the volume level and issues can arise with Snapshot consistency between servers. Sharing volumes also increases chances for busy Snapshot copies.

Microsoft SQL Server diagnostics and monitoring with IBM System Storage N series

In this chapter we introduce basic information about diagnostics and monitoring in environments MS SQL Server with N series.

The goal of database diagnostics and monitoring is to identify problems in your server's resources improving the availability.

At times, it may become necessary to analyze the performance of such a database system in order to locate and resolve one or more problem areas that are having an adverse affect.

Unfortunately, there is no single tool that can be used to help locate and diagnose problems across all components used in such a system. However, there are tools available for monitoring each individual component of the system. We discuss some of these tools and show you how and when they can or should be used.

9.1 Determining whether you have a problem

In this section, we explain how to determine if a problem really exist.

Questions to ask yourself

To clarify the problem, you must ask yourself the following questions:

- ▶ How do you know that there is a problem?
- ▶ Where is the problem occurring?
- ▶ Did you ever have a good working situation?

How do you know that there is a problem?

This question is very basic, yet vital. Identifying how you know whether there is a problem determines the steps to take to analyze the problem.

Where is the problem occurring?

Another important step in analyzing an environment is to identify where the problem is occurring.

Did you ever have a good working situation?

A good working situation is when the system or an application is running without any problems. Knowing if you had a good working situation involves understanding the history of an application or the system. Some problems are caused by the following:

- ▶ The introduction of a new application
- ▶ The application of new program temporary fixes (PTFs)
- ▶ An upgrade to a newer release
- ▶ Changes to system values

9.2 Checklist of services that must be active

Following is a list of the services that must be active:

- ▶ MSSQLServer
- ▶ SnapDriver
- ▶ Microsoft iSCSI Initiator service

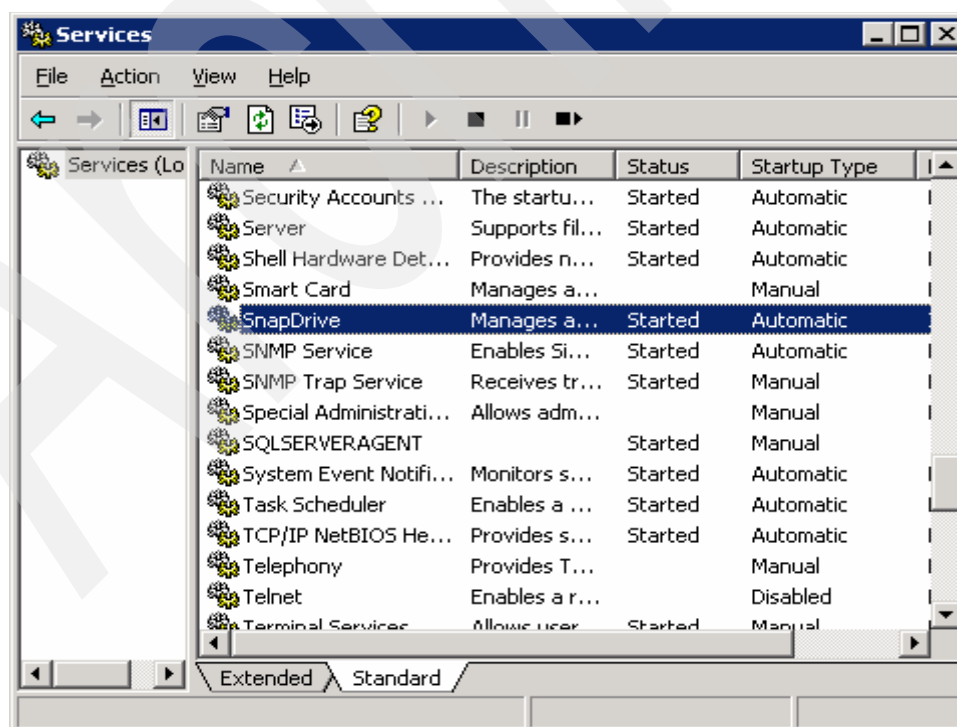


Figure 9-1 Service SnapDrive

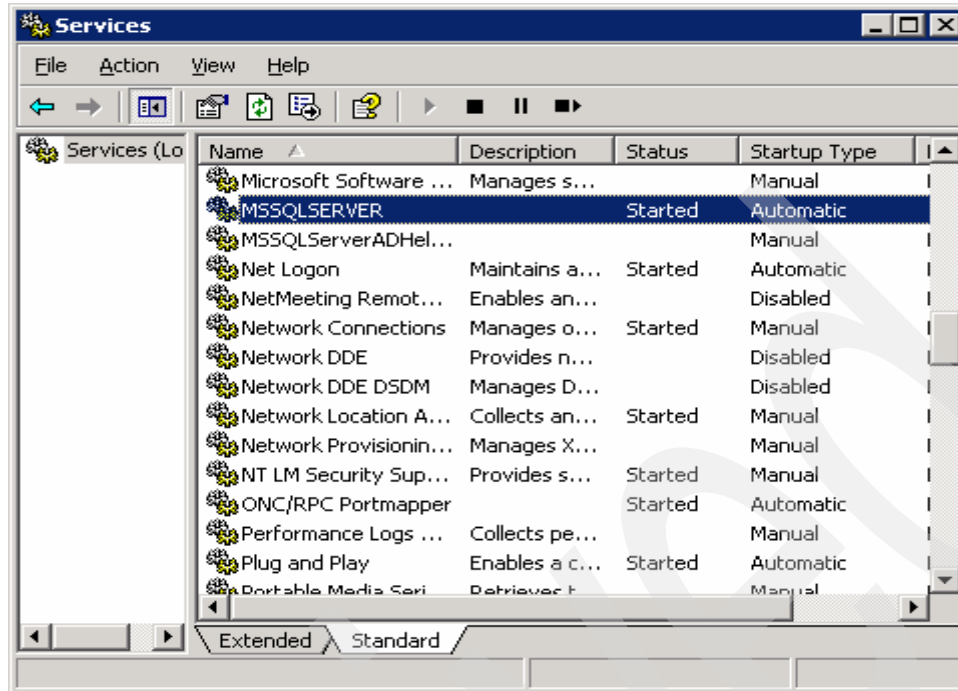


Figure 9-2 Service MSSQLSERVER

9.2.1 MSSQLServer error log

View the Microsoft SQL Server error log to ensure that processes have completed successfully (for example, backup and restore operations, batch commands, or other scripts and processes). This can be helpful to detect any current or potential problem areas, including automatic recovery messages (particularly if an instance of SQL Server has been stopped and restarted), kernel messages, and so on.

View the SQL Server error log by using SQL Server Enterprise Manager or any text editor. By default, the error log is located at Program Files\Microsoft SQL Server\Mssql\Log\Errorlog.

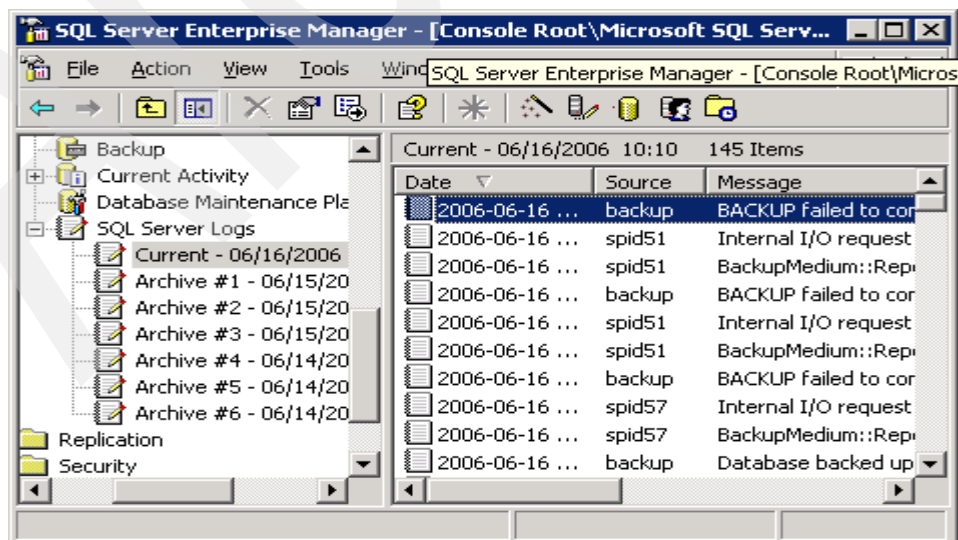


Figure 9-3 SQL Server Logs

9.2.2 SnapManager

View the SnapManager Reports by using SnapManager for SQL Server:

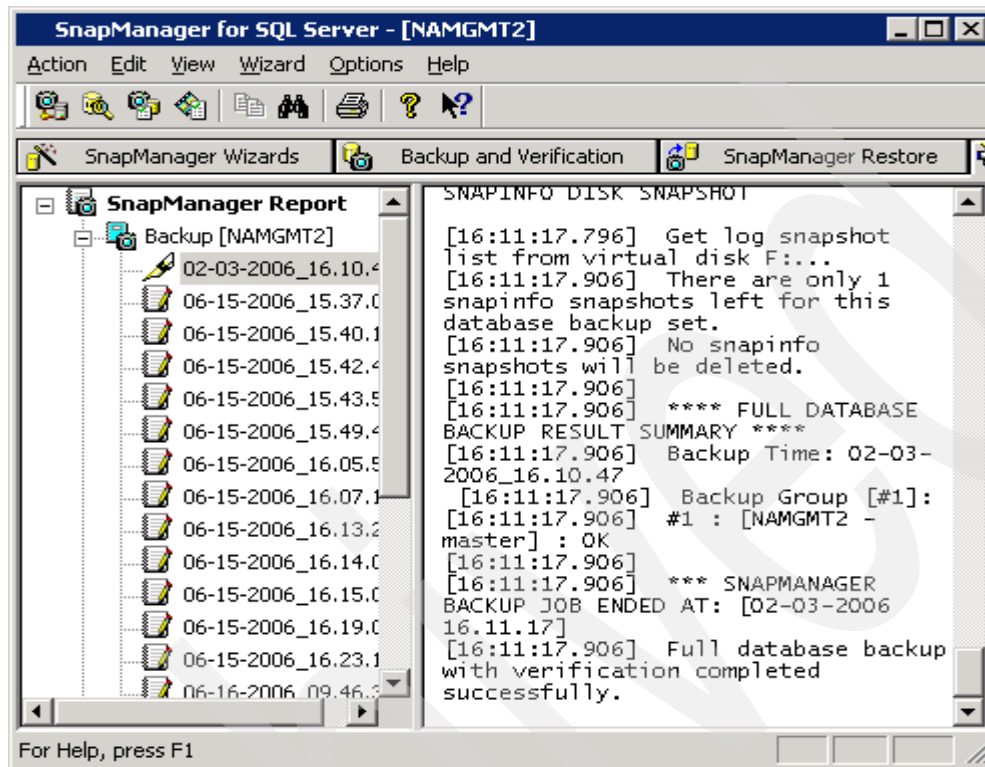


Figure 9-4 SnapManager Report

9.2.3 Event Viewer

View the Event Viewer Reports by using Administrative Tools.

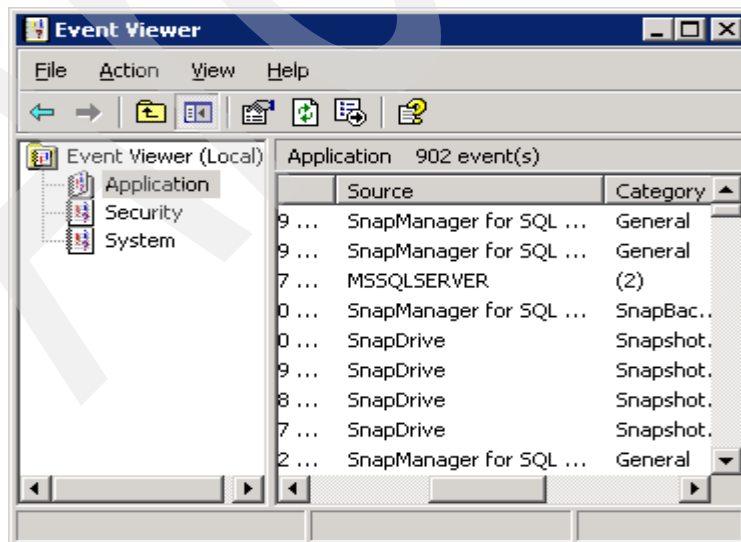


Figure 9-5 Event Viewer

9.3 MSSQLServer Monitoring Server Performance and Activity¹

Microsoft SQL Server provides a variety of tools that can be used to monitor the performance of an instance of SQL Server and the user activity that occurs in databases. Monitoring allows you to determine whether your database application is working efficiently and as expected, even as your application, database, and environment change. For example, as more concurrent users use a database application, the load on SQL Server can increase. By monitoring, you can determine whether the current instance of SQL Server or system configuration must be changed to handle the increased workload or whether the increased load is having no significant effect on performance.

Following are some tips to remember to monitor an application, an instance of SQL Server, or the operating system environment (hardware and software):

- ▶ Determine your monitoring goals.
- ▶ Choose the appropriate tool for the type of monitoring you will perform.
- ▶ Use the tool to monitor SQL Server or the system environment and analyze the captured data.
- ▶ Identify the events to monitor.

The events determine which activities are monitored and captured. Your selection of events to monitor will depend on what is being monitored and why. For example, when monitoring disk activity, it is not necessary to monitor SQL Server locks.

Determine the event data to capture

The event data describes each instance of an event as it occurs. For example, when monitoring lock events, you can capture data describing the tables, users, and connections affected by the lock event. The following explains the process involved in capturing event data and putting it to use.

Apply filters to limit the event data collected

Limiting the event data allows the system to focus on the events pertinent to the monitoring scenario. For example, if you want to monitor slow queries, you can use a filter to monitor only those queries issued by the application that take more than 30 seconds to execute against a particular database.

Monitor (capture) events

After enabled, active monitoring captures data from the specified application, instance of SQL Server, or operating system. For example, when disk activity is monitored using Performance, monitoring captures event data such as disk reads and writes and displays it to the panel.

Save captured event data

Saving captured data allows you to analyze it at a later time or even replay it using SQL Profiler. Captured event data is saved to a file that can be loaded back into the tool that originally created the file for analysis. SQL Profiler allows event data to be saved to a SQL Server table. Saving captured event data is vital when creating a performance baseline. The performance baseline data is saved and used when comparing recently captured event data to determine whether performance is optimal.

¹ From MSSQLServer Books Online

Create definition files containing settings specified to capture events

Definition files include specifications about the events themselves, event data, and filters that are used to capture data. These files can be used to monitor a specific set of events at a later time without redefining the events, event data, and filters.

9.3.1 Choosing a Monitoring Tool

Microsoft SQL Server provides some tools for monitoring events in SQL Server. Your choice of tool depends on the type of monitoring and the events to be monitored.

System Monitor

System Monitor enables you to monitor server performance and activity using predefined objects and counters or user-defined counters to monitor events. Performance collects counts rather than data about the events (for example, Average Disk Queue Length, memory usage, or CPU activity, Figure 9-6). You can set thresholds on specific counters to generate alerts that notify operators. System Monitor primarily tracks resource usage, such as the number of buffer manager page requests in use.

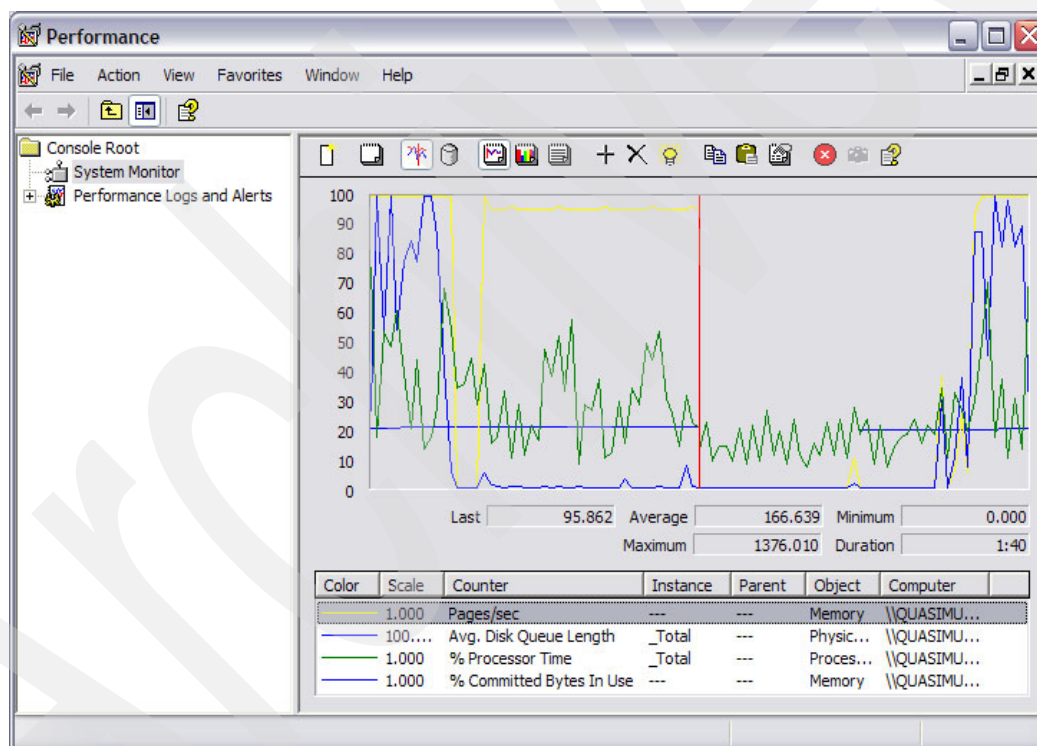


Figure 9-6 Performance monitor

System Monitor can monitor (remotely or locally) an instance of SQL Server.

SQL Server Enterprise Manager - Current activity window

SQL Server Enterprise monitor graphically displays information about processes running currently on an instance of SQL Server, blocked processes, locks, and user activity. This is useful for ad hoc views of current activity.

Error logs

Error logs contain additional information about events in SQL Server that are available elsewhere. You can use the information in the error log to troubleshoot SQL Server-related

problems. The Windows application event log provides an overall picture of events occurring on the Windows system as a whole, as well as events in SQL Server, SQL Server Agent, and full-text search.

9.4 IBM System Storage N series Management Tools

The IBM System Storage N series has several Management Tools available. In this section we cover those that pertain to MSSQL.

Performance Monitoring and Tuning—Set of Basic Counters

Performance monitoring is something every administrator has to do and should do; likewise, performance tuning is something everybody has to do periodically and is a central requirement for implementing capacity planning guidelines correctly. Capacity planning includes recommended counters to collect.

In this section we introduce and provide information about the tools for monitoring.

Snapshot Reserve Usage Monitoring

The task of monitoring the Snapshot reserve is automatically configured at the time of LUN creation. Simply monitor the application event log for warning events generated in the SnapDrive source and Snapshot Monitor event categories. Figure 9-7 demonstrates that due to Snapshot consumption, the reserve must be expanded to 63%, and there is not enough disk space to do so. In order to rectify this situation, either add more disks to the volume or remove some of the older SnapManager for Microsoft SQL Server 2000 Snapshot copies.

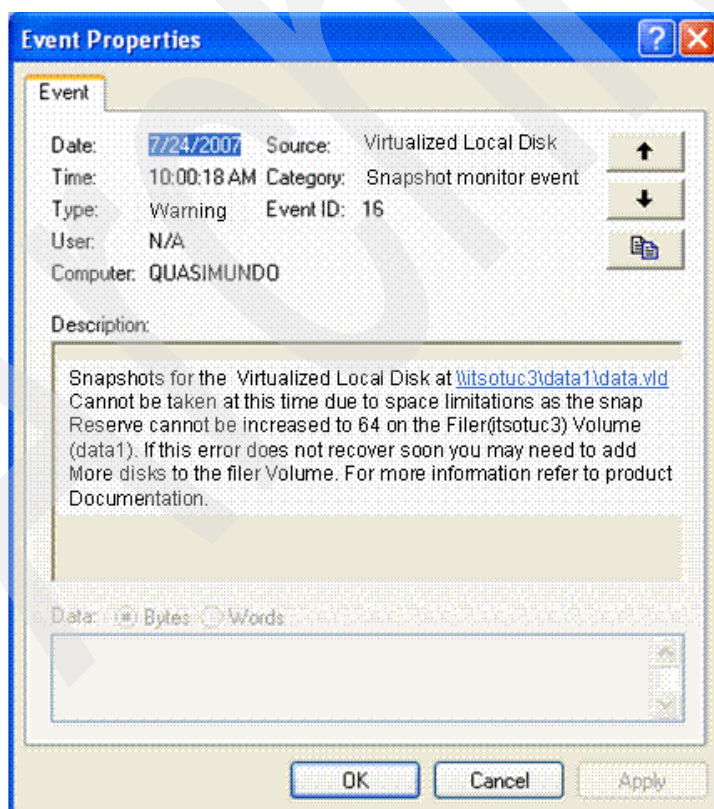


Figure 9-7 LUN error due to space limitations.

Disk Space Usage Monitoring

The amount of disk space used by each database managed by SQL Server 2000 should be monitored to ensure that the logical drives (LUNs) do not run out of space. SQL Server 2000 will autoextend its preallocated file space when there is not enough free file space to extend a table (if the table has been defined with autoextend on). SQL Server stops processing transactions when it cannot extend its file space. Monitoring free space has to be done in two levels. The first level is to free preallocated space. This space is available for table extents. If Perfmon or a similar monitoring tool is already used for monitoring the system performance, then it is easy to add counters to monitor database space. If the database size is only increasing slowly, then it is maybe enough for the DBA to check a database's available space once a quarter. A database's properties will show a database's current size and how much space is free. Figure 9-8 is an example with master properties. It shows a database size of 12.19 MB with 1.27 MB available for new table extends. When free space gets below or close to 64 kB, then the preallocated file space is extended. The second level to monitor is free LUN space, which can be done with Windows Explore. See Figure 9-9.

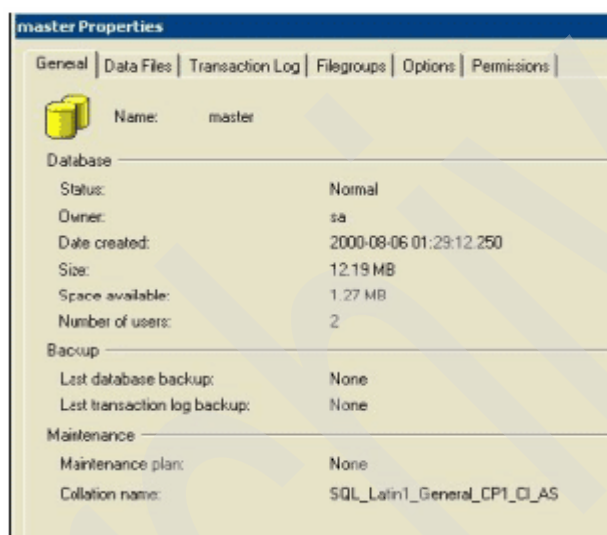


Figure 9-8 Database's available space

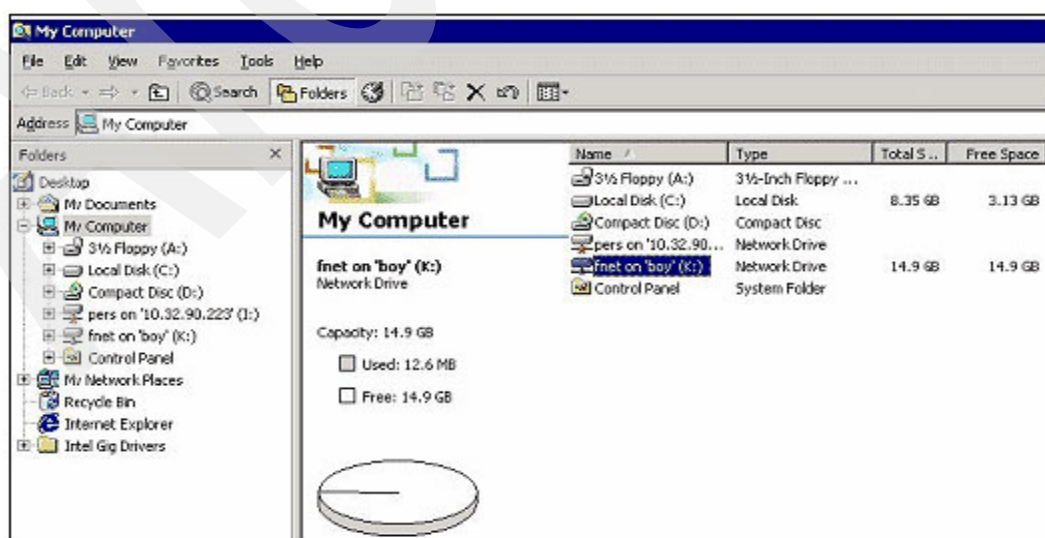


Figure 9-9 Available LUN space

Filer Event Monitoring

Monitor the filer event logs to ensure proper operation of the filer. Issues may be discovered in the event logs that require administrative action. One such action may be to replace a disk in a volume if spare disks are not available. This task can be completed by using the FilerView utility built into Data ONTAP.

This utility can be started by using the following steps:

1. Type the following address into your Web browser
http://filename/na_admin
2. Click the FilerView link. FilerView will start and will ask for the credentials of a filer administrator.
3. After you are logged on to FilerView, click the Filer menu option on the left side of the panel.
4. Choose the Syslog Messages menu option.
5. Review the log on the right side of the panel for any issues that may require administrative action.

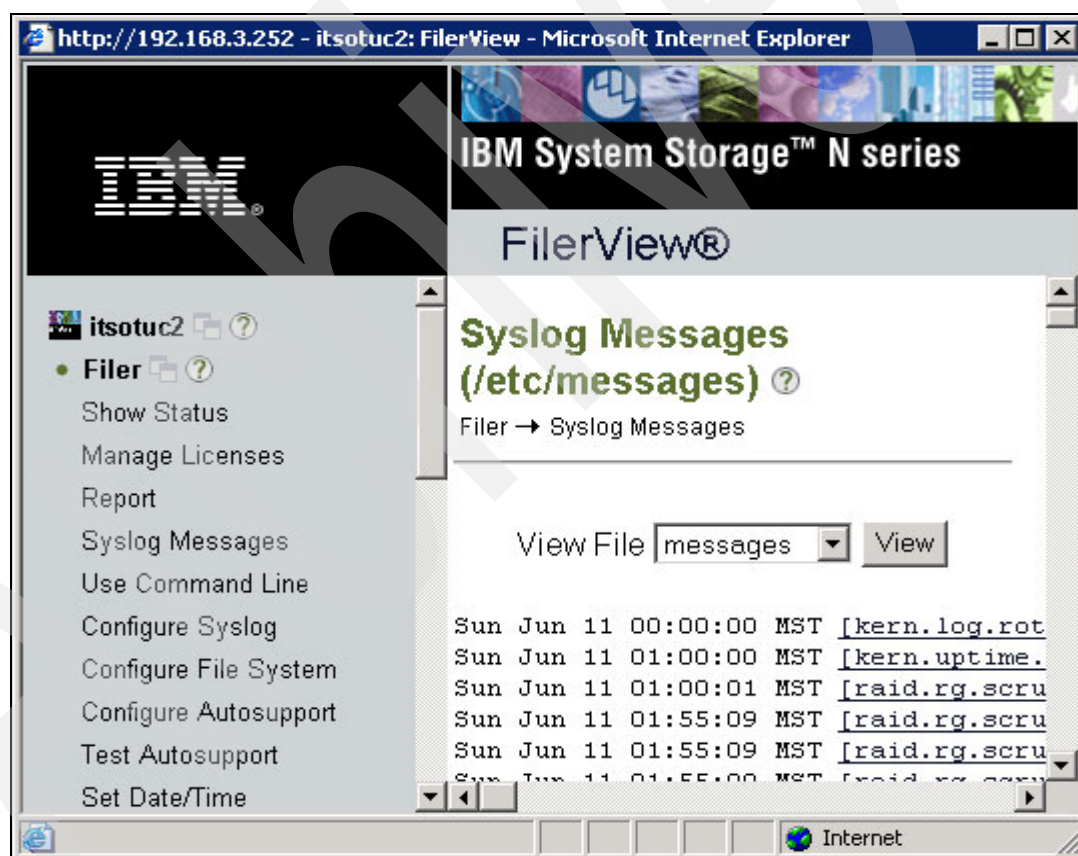


Figure 9-10 Syslog messages in FilerView

Terminal Server or Remote Desktop

Microsoft Terminal Server is not recommended as a way of administrating SnapDrive or SnapManager for SQL Server. Creating LUNs from a terminal server session, the drives can appear as though they were not created or were disconnected when in fact they are operating without errors. The only way to fully resolve problems when using Terminal Server with

SnapDrive is to log out of the session (do not disconnect). We recommend avoiding Terminal Server for server management when possible. SMSQL will, at times, temporarily connect to LUNs in a Snapshot copy when backed-up database(s) have to be verified; however, this operation will fail if the action was activated through Terminal Server!

Use alternative means for remote management

We recommend avoiding Terminal Server for server management when possible. When using Terminal Server with Windows 2003 (not Windows 2000), you may use a remote desktop to connect if you use a command line parameter/console, as shown below:

```
%SystemRoot%\System32\mstsc.exe /console
```

When using Windows 2000, we advise you to use remote control software that makes it appear as though you are directly in front of the system console.



Part 4

Using the N series with IBM DB2

In part four we introduce the basics of the IBM System Storage N series product line and its basic features and capabilities.

Archived

Design considerations and set up with DB2 and IBM System Storage N series

Now that you have an understanding of DB2 and the IBM System Storage N series technology and concepts, we will examine how a DB2 UDB database system can be configured to take advantage of an IBM System Storage N series. In this chapter we first discuss some design considerations to think about prior to setting up. After this, we discuss the steps needed to create a test environment using DB2 UDB Enterprise Server Edition Version 8.2 for AIX 5.2 maintenance level 7 and an IBM System Storage N series N3700.

Note: This Redbooks publication does not discuss the steps needed to install the network or the steps needed to add an IBM System Storage N series to the network. It should be noted, however, that we recommend using a dedicated Gigabit Ethernet network or SAN fabric, enabling flow control, and using full-duplex on both the database server and the IBM System Storage N series.

10.1 DB2 and the N series product design considerations

This section discusses DB2 design considerations when integrating the IBM System Storage N series.

10.1.1 Designing a database with recovery in mind

When designing a new database, it is important to think about recovery needs from the start. As part of the database design process, you should identify any relationships that exist between the various database objects. These relationships can be at an application level, where transactions can work with one or more tables, as well as at a database level, where referential integrity constraints can exist between tables, or where events against one table can activate triggers that affect other tables. With these relationships in mind, you can group related database objects together on the same logical IBM System Storage N series volume or group of volumes. Placing database objects with similar backup requirements or functions on the same logical volume(s) will simplify the use of Snapshot copies and typically make recovery that much easier. In addition, it goes without saying that the archive logs should be stored on a separate volume from that on which the data is stored. A documented model of your database design can be a tremendous benefit when making these kinds of decisions.

10.1.2 Tablespace management

As you may know by now, a DB2 tablespace can be either an system managed storage (SMS) tablespace or a database managed storage (DMS) tablespace. SMS table spaces are managed by the file system and DMS table spaces are managed by DB2.

SMS table spaces typically consist of a number of individual files—representing data objects such as tables and indexes—that are stored on a file system. SMS table spaces have the benefit of being able to grow and shrink on demand. DMS table spaces also have the capability to automatically grow on demand. However, this capability is not turned on by default. The default behavior of a DMS tablespace is to require pre-allocation of storage when the tablespace is created, and the DBA is required to manually resize the DMS tablespace when it gets full.

As you can imagine, this is more work for the DBA. As of DB2 UDB Version 8.2 Fixpak 2 (or Version 8.1 Fixpak 9), DB2 introduced the capability to automatically resize a DMS tablespace with a tablespace option called *AUTORESIZE*. When a DMS tablespace is defined with *AUTORESIZE* set to YES, DB2 will attempt to increase the size of the corresponding containers when they fill up. This new capability gives you the performance benefits of DMS table spaces—which generally perform better than SMS table spaces—as well as the same ease of management that was once exclusive to SMS table spaces.

Specifying the AUTORESIZE option for new DMS table spaces

To specify the *AUTORESIZE* option when creating a DMS new tablespace, use the following command:

```
db2 "create tablespace tsp_name managed by database using (FILE 'filename' size)  
autoresize yes"
```

where *tsp_name* is the name you want to give the tablespace, *filename* is a fully qualified (filename and full path) filename for the FILE container, and *size* is the size of the container. Note the specification of **autoresize yes** in the command. For example, to create a DMS tablespace called *myspace* with one FILE container called */mnt/db2data/myspacefile* that is 100 MB in size, issue the following command:

db2 “create tablespace myspace managed by database using (FILE ‘/mnt/db2data/myspacefile’ 100 M) autoresize yes”

To verify your work, check the directory where you specified your FILE container and look for it. Example 10-1 shows the output of a `ls -l` against the `/mnt/db2data` directory.

Example 10-1 Output of `ls -l` against `/mnt/db2data`

```
$ ls -l /mnt/db2data
total 204816
drwxrwxr-x  3 db2inst2 db2adm          512 Jun 19 17:23 db2inst1
drwxrwx---  2 db2inst2 db2adm          512 Jun. 19 17:15 lost+found
-rw-----  1 db2inst2 db2adm    104857600 Jun 19 18:19 myspacefile
$
```

You can verify that the tablespace was created by performing the following steps:

1. Connect to the sample database.
2. Issue the **db2 list tablespaces** command to verify that your tablespace was created. The output of this command will also show you the corresponding tablespace ID.
3. Issue the **db2 list tablespace containers for ID** where ID is the corresponding tablespace ID.

Example 10-2 shows the output of these commands. You can see from the output of the **db2 list tablespaces** command, that the myspace tablespace is a DMS (Database managed space) tablespace, and that the corresponding tablespace ID is 3. From the **db2 list tablespace containers for 3** command, we can see that there is one container of type FILE, and it is located in `/mnt/db2data/myspacefile`.

Example 10-2 Output of steps to verify tablespace creation

```
$ db2 connect to sample

      Database Connection Information

Database server      = DB2/6000 8.2.5
SQL authorization ID = DB2INST1
Local database alias = SAMPLE

$ db2 list tablespaces

      Tablespaces for Current Database

Tablespace ID      = 0
Name                = SYSCATSPACE
Type                = System managed space
Contents            = Any data
State               = 0x0000
  Detailed explanation:
    Normal

Tablespace ID      = 1
Name                = TEMPSPACE1
Type                = System managed space
Contents            = System Temporary data
State               = 0x0000
```

Detailed explanation:

Normal

Tablespace ID	= 2
Name	= USERSPACE1
Type	= System managed space
Contents	= Any data
State	= 0x0000

Detailed explanation:

Normal

Tablespace ID	= 3
Name	= MYSPACE
Type	= Database managed space
Contents	= Any data
State	= 0x0000

Detailed explanation:

Normal

```
$ db2 list tablespace containers for 3
```

Tablespace Containers for Tablespace 3

Container ID	= 0
Name	= /mnt/db2data/myspacefile
Type	= File

```
$
```

Enabling the AUTORESIZE option on existing DMS table spaces

To enable the AUTORESIZE option for an existing DMS tablespace, issue the following command:

```
db2 "alter tablespace tsp_name autoresize yes"
```

where *tsp_name* is the name of the existing tablespace. For example, to modify an existing DMS tablespace myspace to enable AUTORESIZE, issue the following command:

```
db2 "alter tablespace myspace autoresize yes"
```

10.2 Creating a sample database on the N series product in an NAS environment

In this section, we create a sample database where the data and logs reside on IBM System Storage N series FlexVol volumes in an NAS environment. First, we set up the N series product with the aggregates and volumes needed to store the data and logs for the sample database. Next, we set up the database server to access the volumes we created on the N series product. Lastly, we create the sample database such that the data and logs reside on the storage system's volumes on the N series product for NFS access.

10.2.1 Configuring the IBM System Storage N series

Use the following steps to configure the IBM System Storage N series:

1. Activate the appropriate license keys.

On the IBM System Storage N series, the file access protocols FCP, iSCSI, NFS, and CIFS are licensed services. You need to enable the appropriate service by activating license keys for the protocol you intend to use. The license keys can be activated by executing the following command from the IBM System Storage N series:

```
license add <license code>
```

For example, to activate the license code for NFS, you would execute the following command on the IBM System Storage N series:

```
license add 123XYZABCD
```

You can verify your action by issuing the **license** command, as shown in Figure 10-1.

```
itsotuc2> license
      cifs site TNXZHUO
      cluster not licensed
      cluster_remote not licensed
      disk_sanitization not licensed
      fcp site RGZOV1L
      flex_cache not licensed
      flex_clone site IPMK2CN
      gateway not licensed
      gateway_hitachi not licensed
      http not licensed
      iscsi site RGF3RGF
      multistore site DF4XLWH
      nfs site R5IEGMK
      rapid_restore not licensed
      smdomino not licensed
      smsql site 6BAEGMK
      snaplock not licensed
      snaplock_enterprise site FITZKF7
      snapmanagerexchange site NGQKA8N
      snapmirror site TTOX9WH
      snapmirror_sync site NYJABTI
      snapmover not licensed
      snaprestore site XJBGCKE
      snapvalidator site ZCUZHUB
      sv_linux_pri not licensed
      sv_ontap_pri not licensed
      sv_ontap_sec site DSQMWZG
      sv_unix_pri not licensed
      sv_windows_ofm_pri not licensed
      sv_windows_pri not licensed
      syncmirror_local not licensed
      vld site DTWKACE
```

Figure 10-1 License command

2. Update the /etc/host file on the IBM System Storage N series used.

The IBM System Storage N series must be able to communicate with the database server and vice versa. The IBM System Storage N series can communicate to the database server if there is an entry in its /etc/hosts file for the database server, or alternatively if it uses some other host name resolution techniques, such as NIS or DNS. By default, the /etc/hosts file is checked first for host name resolution. The easiest way to update the /etc/hosts file on the storage system is by using FilerView (see Figure 10-2). Entries made in the /etc/hosts file should look similar to the following:

<db server IP address> <host name>

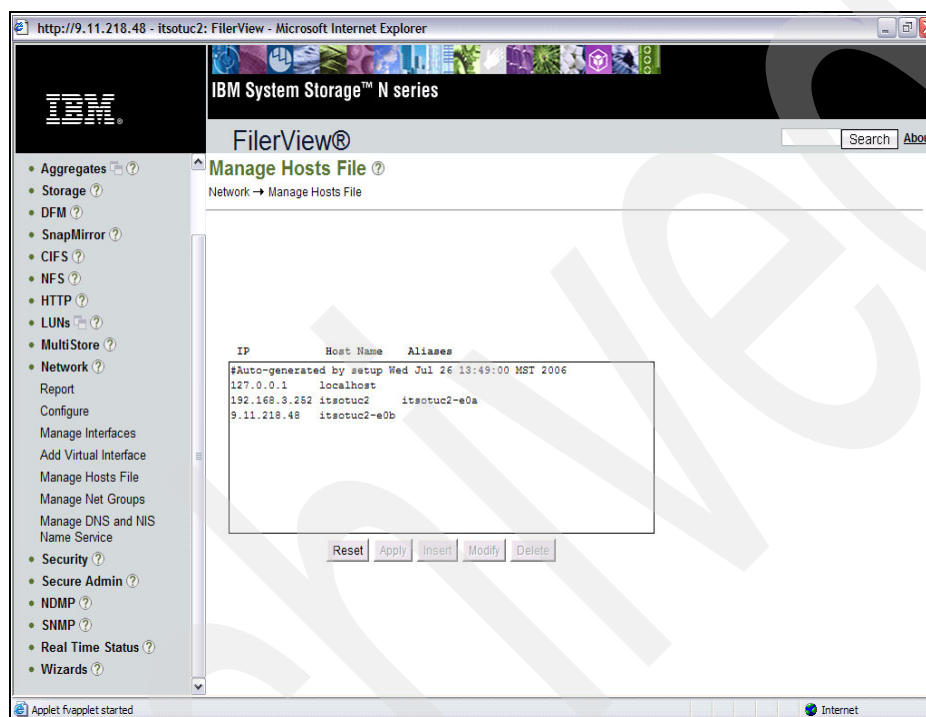


Figure 10-2 FilerView

For example, to add an entry for a database server named db2srv1 that has an IP address of 172.17.32.112, you would add the following line to the /etc/hosts file on the IBM System Storage N series:

172.17.32.112 db2srvr1

3. Enable rsh access for the database server.

In order to use the rsh (remote shell) command from the database server, you need to perform two steps.

- First, enable the rsh option on the IBM N series storage system by executing the following command:
- Then, add the database host and user name entry to the /etc/hosts.equiv file found on the IBM System Storage N series. The entry to this file looks somewhat similar to the following one:

< hostname > < username >

For example, to allow `rsh` command execution from a database server named `db2srv1` by a user named `db2inst1`, you would add the following line to the `/etc/hosts.equiv` file on the IBM System Storage N series:

```
db2srv1 db2inst1
```

4. Export the volumes if NFS is used (Skip this step if FCP or iSCSI is used).

To access the volumes on an IBM System Storage N series using the NFS protocol, you must export them. A volume can be exported by adding an entry to the `/etc/exports` file found on the IBM System Storage N series and giving the database server appropriate access to it. You can use FilerView (see Figure 10-3) or the `exportfs` command for this purpose. NFS exports are managed using the following command:

```
exportfs -p rw=<hostname>,root=<hostname> <pathname>
```

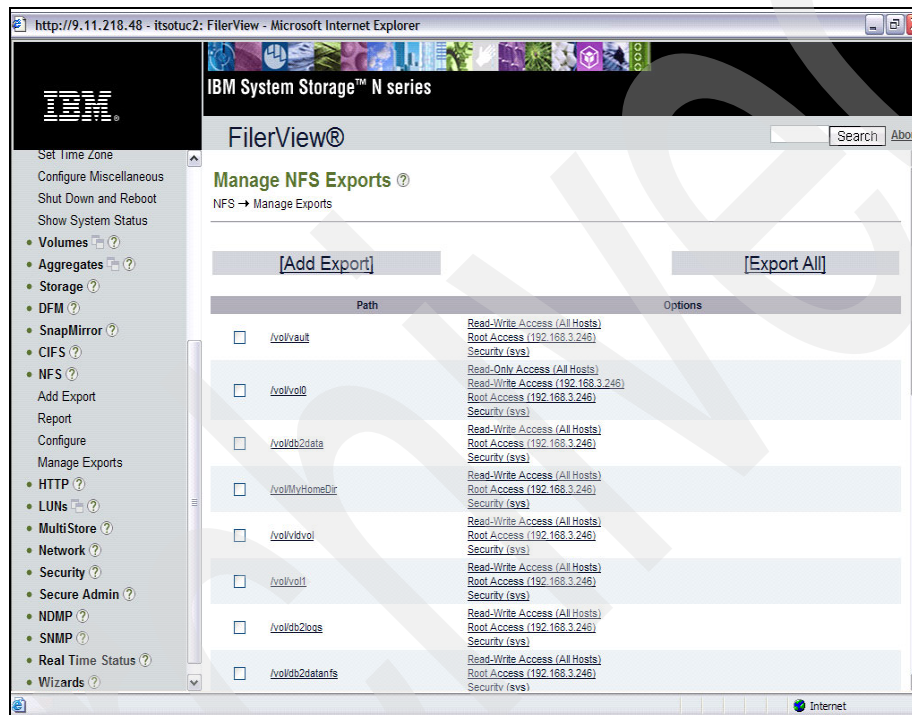


Figure 10-3 FilerView

For example, to create an export entry for a volume named `db2data` and allow root access to it from a database server named `db2srv1`, you would execute the following command on the IBM System Storage N series:

```
exportfs -p rw=db2srv1,root=db2srv1 /vol/db2data
```

Repeat this step and create an export entry for all the volumes to be used for the database.

10.2.2 Configuration and Management

The IBM System Storage N series provides two methods for configuration and management: a Web-based GUI interface and a command line interface. In this section, we use the command line interface.

1. To get to the storage system's command line interface, telnet or remote shell (`rsh`) to the storage system's hostname or IP address.

2. Create space to store data and transaction log files.

In order to create a database on the IBM System Storage N series, you need to create aggregates, flexible volumes, and in some cases LUNs. An aggregate is a physical pool of storage at a RAID level (Figure 10-4) and is created using the following command:

```
aggr create <aggrname> -r <raidsize> <ndisks@disksize> -t <raidtype>
```

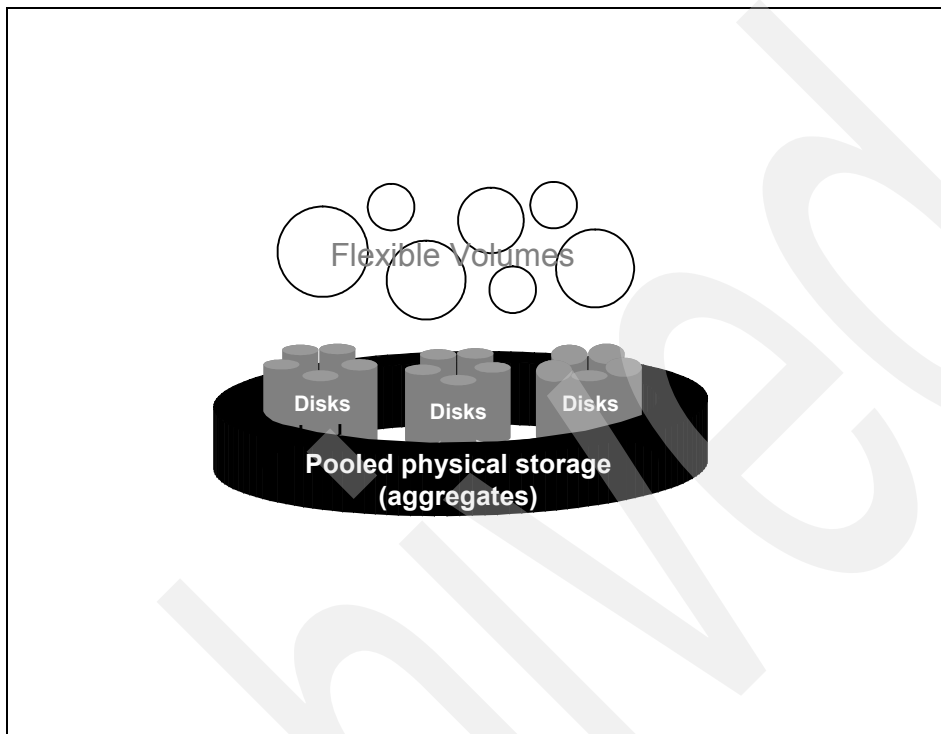


Figure 10-4 Aggregates

For example, to create an aggregate named db2aggr1 that has eight 72 GB disks in it and that uses RAID-DP, execute the following command on the IBM System Storage N series:

```
aggr create db2aggr1 -r 8 8072g -t raid_dp
```

A FlexVol volume is a logical storage container inside an aggregate. A FlexVol volume can be as small as a few megabytes and as large as the aggregate itself. A FlexVol volume can be created using the following command:

```
vol create <vol-name>
{ [-l <language-code>] [-s {none | file | volume}]
  <hosting-aggr-name> <size>[k|m|g|t]
  [-S remotehost:remotevolume] }
```

For example, to create a flexible volume named db2data that is 100 GB in size and that resides in the aggregate named db2aggr1, execute the following command on the IBM System Storage N series:

```
vol create db2data db2aggr1 100GB
```

We create one aggregate to store the volumes for both the data and logs for our sample database. To create an aggregate named db2aggr1 that has four 72GB disks in it and uses RAID-DP, issue the following command within a telnet session to the storage system:

```
aggr create db2aggr1 -t raid_dp -r 4 4072
```

You can issue **aggr status** to ensure that your aggregate was created. The output of this command on our test environment is shown in Example 10-3.

*Example 10-3 Output of **aggr status***

```
itsotuc2> aggr status
      Aggr State      Status      Options
      db2aggr1 online  raid_dp, aggr  root
itsotuc2>
```

- Now that we have our aggregate, we can create one volume for data and another one for logs. To create a volume named db2data that is 50 GB in size within the db2aggr1 aggregate, issue the following command:

vol create db2data db2aggr1 50g

To create a volume named db2logs that is 30GB in size within the db2aggr1 aggregate, issue the following command:

vol create db2logs db2aggr1 30g

To verify the creation of both volumes, issue **vol status** to see the status of all volumes. The output is similar to Example 10-4.

*Example 10-4 Output of the **vol status** command*

```
tsotuc2> vol status
      Volume State      Status      Options
      vol0 online      raid_dp, flex  root
      db2data online    raid_dp, flex
      db2logs online    raid_dp, flex
      vol1 online      raid_dp, flex
tsotuc2>
```

- Because our database server is AIX, we need to ensure that the volumes we use for DB2 support either the Unix or mixed security style. To set the db2data and db2logs volumes to use the Unix security style, issue the following commands:

qtree security /vol/db2data unix

qtree security /vol/db2logs unix

To verify that the security was set properly, issue **qtree status**. The results of this command are shown in Example 10-5. As you can see, the db2data and db2logs volumes are shown to be using the Unix security style now.

Example 10-5 Output showing security styles of volumes

```
itsotuc2> qtree status
Volume  Tree      Style  Oplocks  Status
-----
vol0    ntfs    enabled  normal
vol1    ntfs    enabled  normal
db2data  unix    enabled  normal
db2logs  unix    enabled  normal
itsotuc2>
```

5. Normally, a database is backed up on a set schedule. Therefore, we recommend that you disable the automatic Snapshot feature for any volume used by DB2. We discuss manually taking snapshots during database backup in a later chapter. To turn off the automatic Snapshot feature for the db2data and db2logs volumes, issue the following commands:

```
vol options db2data nosnap on
```

```
vol options db2logs nosnap on
```

To verify that this option is set, issue **vol status** to see the status and options for each volume. You can see the output from this command, in Example 10-6, which is that the db2data and db2logs volumes now show the *nosnap=on* option.

Example 10-6 Output of the vol status command

```
tsotuc2> vol status
      Volume State      Status      Options
      vo10 online      raid_dp, flex  root
      db2data online    raid_dp, flex  nosnap=on
      db2logs online    raid_dp, flex  nosnap=on
      vol1 online      raid_dp, flex
```

6. In order for the AIX server to be able to mount these two volumes we created, we export them using the NFS protocol. To export the db2data and db2logs volumes and allow read/write and root access from database server aixdb2, issue the following command:

```
exportfs -p rw=aixdb2,root=aixdb2 /vol/db2data
```

```
exportfs -p rw=aixdb2,root=aixdb2 /vol/db2logs
```

Ensure that the parameters for *rw* and *root* point to either the hostname or IP address of the AIX server. To verify your work, issue **exportfs** on its own. It should look similar to Example 10-7.

Example 10-7 Output of exportfs

```
itsotuc2> exportfs
/vol/db2data -sec=sys,rw=aixdb2,root=aixdb2
/vol/db2logs -sec=sys,rw=aixdb2,root=aixdb2
itsotuc2>
```

Now, we are ready to configure the database server to access these volumes.

10.2.3 Accessing the volumes on the database server

The next step is to mount the db2data and db2logs volumes on the AIX database server. The following commands must be executed as the root user on the AIX server.

1. Log onto the AIX server as root.
2. Create mount points to mount the volumes you created earlier:

```
mkdir -p mount_directory
```

where *mount_directory* is the directory where you want to mount the device or file system. In our test environment, we issue the following commands:

```
mkdir -p /mnt/db2data
```

```
mkdir -p /mnt/db2logs
```

3. Modify the `/etc/filesystems` to give AIX the filesystem information it needs to properly mount the volumes. In our example, we add the lines in Example 10-8 to our `/etc/filesystems` file on AIX:

Example 10-8 `/etc/filesystems` file

```
/mnt/db2data:
    dev      = /vol/db2data
    mount    = /true
    vfs      = nfs
    nodename = 192.168.3.251
    options  = bg,nointr,rw
    type     = nfs_mount
    account  = false

/mnt/db2logs:
    dev      = /vol/db2logs
    mount    = /true
    vfs      = nfs
    nodename = 192.168.3.251
    options  = bg,nointr,rw
    type     = nfs_mount
    account  = false
```

The entry `mount = true` makes the specified volume persistent across system reboots. The `nodename` entry specifies the hostname or IP address of the IBM System Storage N series, which in our case is 192.168.3.251. Set the `nodename` entry accordingly.

4. Mount the volumes to the mount points:

```
mount /mnt/db2data
```

```
mount /mnt/db2logs
```

5. In our environment, we already created a DB2 instance called `db2inst1`. Every DB2 instance on Unix and Linux environments has an associated user ID with the same name known as the *instance owner*. In our case, that user ID is `db2inst1`. The instance owner requires ownership of the file systems we mounted. To change the ownership of the `db2data` and `db2logs` mounts to the instance owner `db2inst1`—whose primary group is `db2adm`—issue the following commands:

```
chown -R db2inst1:db2adm /mnt/db2data
```

```
chown -R db2inst1:db2adm /mnt/db2logs
```

10.2.4 Creating the sample database on the mounts

DB2 has a tool called `db2sampl` that creates a small sample database with sample tables and data for the purposes of testing and verification. We use this tool to create the sample database in our test environment.

1. Log onto the AIX server as the instance owner. In our test environment, we log on as `db2inst1`.
2. We create the sample database using the `db2sampl` tool. The syntax for this tool is as follows:

```
db2sampl path
```

This command creates a database named `sample` and *path* is the directory where this database is created. In our test environment, we want the sample database to be created

within the db2data volume. The mount point for this volume is /mnt/db2data. Issue the following command:

```
db2samp1 /mnt/db2data
```

Alternative: As stated earlier, the sample database contains sample tables and data. If you want to create a brand new database instead, use the following DB2 CREATE DATABASE command:

db2 “create database *database* on *path*”

Please see DB2 UDB documentation for a complete list of options.

3. By default, when the sample database is created, circular logging is used, and log files are created in a subdirectory within the path where the database was created. Because we want our logs to reside in the db2logs volume, we need to tell DB2 to put the logs within the /mnt/db2logs mount point instead. To accomplish this, we update the configuration parameter NEWLOGPATH for the sample database. The syntax for updating a database configuration parameter is as follows:

db2 “update db cfg for *database* using *parameter value*”

where *database* is the database alias, *parameter* is the parameter you want to change, and *value* is the new value for the parameter. In our test environment, issue the following command:

db2 “update db cfg for sample using NEWLOGPATH /mnt/db2logs”

4. To verify that the database was created correctly in the volume we intended, issue the following command:

db2 “list db directory”

You should see an output similar to Example 10-9.

Example 10-9 Listing of the database directory

```
$ db2 “list db directory”
```

```
System Database Directory
```

```
Number of entries in the directory = 1
```

```
Database 1 entry:
```

Database alias	= SAMPLE
Database name	= SAMPLE
Local database directory	= /mnt/db2data
Database release level	= a.00
Comment	=
Directory entry type	= Indirect
Catalog database partition number	= 0
Alternate server hostname	=
Alternate server port number	=

```
$
```

5. Finally, we verify that the tablespace containers for the sample database reside on the db2data volume.

a. Connect to the sample database using the following command:

db2 "connect to sample"

b. List all the table spaces for the database we are currently connected to using the following command:

db2 "list tablespaces"

You should see an output similar to Example 10-10.

Example 10-10 Listing of table spaces

```
$ db2 "list tablespaces"
```

Tablespaces for Current Database

```
Tablespace ID          = 0
Name                    = SYSCATSPACE
Type                    = System managed space
Contents                = Any data
State                   = 0x0000
Detailed explanation:
    Normal

Tablespace ID          = 1
Name                    = TEMPSPACE1
Type                    = System managed space
Contents                = System Temporary data
State                   = 0x0000
Detailed explanation:
    Normal

Tablespace ID          = 2
Name                    = USERSPACE1
Type                    = System managed space
Contents                = Any data
State                   = 0x0000
Detailed explanation:
    Normal
```

```
$
```

c. For each of the tablespace IDs listed, issue the following command:

db2 "list tablespace containers for ID show detail"

where *ID* corresponds to a tablespace ID listed in the previous step. In Example 10-11, we see the output of this command issued against the USERSPACE1 tablespace (tablespace ID 2). The output shows that there is one container for this tablespace, with ID 0. This container resides in /mnt/db2data/db2inst1/NODE0000/SQL00001/SQLT0002.0. This verifies that the container is in the correct mount; hence, the correct volume on N series.

Example 10-11 Listing of tablespace containers for the USERSPACE1 tablespace

```
$ db2 "list tablespace containers for 2 show detail"
```

Tablespace Containers for Tablespace 2

Container ID	= 0
Name	= /mnt/db2data/db2inst1/NODE0000/SQL00001/SQLT0002.0
Type	= Path
Total pages	= 407
Useable pages	= 407
Accessible	= Yes

\$

10.3 Creating a sample database on N series in a SAN environment

In a SAN environment, Unix servers are called *initiators*, and the IBM System Storage N series are called *targets*. On the target, we create logical storage units called *LUNs* (*logical units*) that the initiators will access through the SAN. From the AIX server's perspective, each LUN is seen as a physical hard disk.

10.3.1 Configuring AIX Host for FCP

1. Install N series' host Attach Kit.

We recommend that you download and install the IBM System Storage N series provided SAN (FCP) Host Attach Kit for AIX on the database server. It is located at the following Web site:

http://www-1.ibm.com/support/docview.wss?rs=1149&context=STCG93G&dc=DA400&uid=sg1S7001639&loc=en_US&cs=utf-8&lang=en

This product simplifies the configuration and management of the AIX host in an IBM System Storage N series SAN environment. In order to install this product, you need to complete the following basic steps:

- IBM System Storage N series FCP AIX Host Attach Kit 4.0 binaries are provided to licensed customers only and can be obtained by contacting IBM support.
- Obtain and copy the downloaded software tar file to a work directory (/tmp/nseries) on the database server and uncompress the file using the following command:

```
uncompress /tmp/nseries/ntap_aix_fcp_4.0.tar.Z
```

- Extract the file by executing the following command on the database server:

```
tar -xvf <file name>
```

For example, you would execute the following command on the database server to extract a file named ntap_aix_fcp_4.0.tar:

```
tar -xvf /tmp/nseries/ntap_aix_fcp_4.0.tar
```

The above command extracts the attach kit software in to the ntap_aix_fcp_1.2 directory.

- Change the directory to ntap_aix_fcp_4.0, where you have extracted the attach kit software, and execute the following command on the database server:

```
./install
```

Answer the prompt and complete the installation.

- e. Add the following lines to the .profile file found on the database server in the user's home directory:

```
export PATH=$PATH:/opt/OnTap/santools/bin
export MANPATH=/usr/share/man:/opt/OnTap/santools/man
```

- f. After installation is complete, reboot the system using the following command:

```
shutdown -F -r now
```

For detailed installation steps, please refer to the N series FCP AIX Host Attach Kit 4.0 product documentation on the IBM Support for FCP Host Attach Kits site.

2. Install the HBA and driver.

Before you install HBA on the database server, you need to check the product's compatibility at the following IBM support Web site:

<http://www-03.ibm.com/servers/storage/support/nas/index.html>.

The compatibility matrix is available on the IBM System Storage N series and TotalStorage® NAS interoperability matrixes. After confirming compatibility, install the HBA and driver on the database server. For more information about the HBA and driver installation, please refer to the product's Installation and configuration guide.

3. Obtain the WWPN for the HBA.

Each FC HBA attached to the database server is uniquely identified by a WWPN. To create an igroup on the IBM System Storage N series, the WWPN for the HBA is required. To obtain the WWPN, complete the following steps:

- a. After installing the IBM host Attach kit for AIX, HBA, and the drivers on the database server, you can find the WWPN by executing the following command:

```
sanlun fcp show adapter -c
```

10.3.2 Setting up LUNs on N series

We start off by creating volumes for storing our data and logs. However, rather than exporting the volumes as NFS mounts, we create LUNs that the AIX server can detect from its Fibre Channel ports (fcp) as hard disk devices. As in the previous section, we will continue to use the command line interface.

1. To get to the storage system's command line interface, telnet or remote shell (rsh) to the storage system's hostname or IP address.
2. An aggregate is a physical pool of storage for volumes to reside. We create one aggregate to store the volumes for both the data and logs for our sample database. To create an aggregate named db2aggr1 that has four 72 GB disks in it and uses RAID-DP, issue within a telnet session to the storage system the following command:

```
aggr create db2aggr1 -t raid_dp -r 4 4072
```

You can issue **aggr status** to ensure that your aggregate was created. The output of this command on our test environment is shown in Example 10-12.

Example 10-12 Output of aggr status

```
itsotuc2> aggr status
      Aggr State      Status      Options
      db2aggr1 online  raid_dp, aggr  root
itsotuc2>
```

3. Now that we have our aggregate, we can create one volume for data and another one for logs. To create a volume named db2data that is 50 GB in size within the db2aggr1 aggregate, issue the following command:

```
vol create db2data db2aggr1 50g
```

To create a volume named db2logs that is 30 GB in size within the db2aggr1 aggregate, issue the following command:

```
vol create db2logs db2aggr1 30g
```

To verify the creation of both volumes, issue **vol status** to see the status of all volumes. The output is similar to Example 10-13.

Example 10-13 Output of the vol status command

```
tsotuc2> vol status
      Volume State      Status      Options
      vo10 online      raid_dp, flex  root
      db2data online    raid_dp, flex
      db2logs online    raid_dp, flex
      vo11 online      raid_dp, flex
tsotuc2>
```

4. Since our database server is AIX, we need to ensure that the volumes we use for DB2 support either the Unix or mixed security style. To set the db2data and db2logs volumes to use the Unix security style, issue the following command:

```
qtree security /vol/db2data unix
```

```
qtree security /vol/db2logs unix
```

To verify that the security was set properly, issue **qtree status**. The results of this command are shown in Example 10-14. As you can see, the db2data and db2logs volumes are shown to be using the Unix security style now.

Example 10-14 Output showing security styles of volumes

```
itsotuc2> qtree status
Volume  Tree      Style Oplocks  Status
-----
vo10    ntfs  enabled  normal
vo11    ntfs  enabled  normal
db2data  unix  enabled  normal
db2logs  unix  enabled  normal
itsotuc2>
```

5. Normally, a database is backed up on a set schedule. Therefore, we recommend that you disable the automatic Snapshot feature for any volume used by DB2. We discuss manually taking snapshots during database backup in a later chapter. To turn off the automatic Snapshot feature for the db2data and db2logs volumes, issue the following commands:

```
vol options db2data nosnap on
```

```
vol options db2logs nosnap on
```

To verify that this option is set, issue **vol status** to see the status and options for each volume. You can see the output from this command—in Example 10-15 on page 185—that the db2data and db2logs volumes now show the *nosnap=on* option.

Example 10-15 Output of the vol status command

```
tsotuc2> vol status
      Volume State      Status      Options
      vol0 online      raid_dp, flex  root
      db2data online    raid_dp, flex  nosnap=on
      db2logs online    raid_dp, flex  nosnap=on
      vol1 online      raid_dp, flex
tsotuc2>
```

6. Now that the volumes are properly set up, create two LUNs within these volumes: one for data and the other for logs. When creating LUNs, you need to specify a size. In our example, we attempt to size the LUNs to fill up the entire volume. We call the LUNs /vol/db2data/lun and /vol/db2logs/lun, respectively. Issue the following command:

```
lun create -s 50g -t aix /vol/db2data/lun
```

```
lun create -s 30g -t aix /vol/db2logs/lun
```

You will likely get an error complaining that there is no space left on the device. Due to overhead data that is written into the volumes, you cannot specify the LUN size equal to its respective volume. However, along with the error message, you will be shown the maximum LUN size allowed for the volume. In Example 10-17, we attempt to create a 50 GB LUN within the db2data volume. The command fails but informs us that the maximum size allowed is 40873 MB. To verify that the LUNs were created properly, issue the **lun show** command. The output of this command can be seen in Example 10-16.

Example 10-16 Output of the lun show command

```
itsotuc2> lun show
      /vol/db2data/lun      39.9g (42858446848)  (r/w, online)
      /vol/db2logs/lun     23.9g (25715277824)  (r/w, online)
itsotuc2>
```

Example 10-17 Output of the lun create command

```
itsotuc2*> lun create -s 50g -t aix /vol/db2data/lun
lun create: No space left on device
lun create: max size: 40873m (42858446848)
itsotuc2*> lun create -s 40873m -t aix /vol/db2data/lun
itsotuc2*>
```

7. Next, we create an igroup (initiator group) that tells the storage system which initiator(s) can access a LUN. Before you create an igroup, you first need the WWPN (world wide port name) for the fibre-channel host bus adapter (HBA) on your AIX server that is connected to the SAN. The WWPN is a unique identifier for each fibre-channel HBA. To obtain the WWPNs, do the following:

- Log onto the AIX server as the root user.
- Issue the following command on the AIX server as the root user:

```
sanlun fcp show adapter -c
```

Tip: The **sanlun** utility is included in the *IBM FCP AIX Host Attach Kit*. For more information about this utility, please refer to the *FCP AIX Host Attach Kit Installation and Setup Guide*, available at the following Web site:

<http://www-1.ibm.com/support/docview.wss?uid=ssg1S7001347&aid=5>

Example 10-18 shows the output of this command. The `-c` option of the `sanlun` utility generates an **igroup create** command that can be issued on the storage system's command line interface to create an igroup. From the example, you can see that there are two WWPNs, 10000000c933158e and 10000000c93316f8, meaning that we have two HBAs on our test AIX server.

Example 10-18 Output showing the WWPNs of the host bus adapters

```
# sanlun fcp show adapter -c

Enter this filer command to create an initiator group for this system:
igroup create -f -t aix "aixdb2" 10000000c933158e 10000000c93316f8
#
```

On the storage system, issue the **igroup create** command:

```
igroup create -f -t aix "aixdb2" 10000000c933158e 10000000c93316f8
```

The name of the resulting igroup is `aixdb2`. To verify that the igroup was created, issue the **igroup show** command. The output of this command is shown in Example 10-19. As you can see from the example, there are two WWPNs listed in the igroup `aixdb2`, but only one is logged on, meaning that only one of the HBAs are connected to the SAN.

Example 10-19 Output of the igroup show command

```
itsotuc2> igroup show
aixdb2 (FCP) (ostype: aix):
    10:00:00:00:c9:33:15:8e (logged in on: 0c, vtic)
    10:00:00:00:c9:33:16:f8 (not logged in)
itsotuc2>
```

8. Now that the LUNs and igroup are created, map the LUNs to the igroup. This mapping allows the WWPNs in the igroup to access the LUNs. To create the mappings, issue the following command:

```
lun map /vol/db2data/lun aixdb2
lun map /vol/db2logs/lun aixdb2
```

The output of these commands is shown in Example 10-20.

Example 10-20 Output of the lun map commands

```
itsotuc2> lun map /vol/db2data/lun aixdb2
lun map: auto-assigned aixdb2=0
itsotuc2> lun map /vol/db2logs/lun aixdb2
lun map: auto-assigned aixdb2=1
itsotuc2>
```

You can also show the lun mappings using the **lun show -m** command, as shown in Example 10-21.

Example 10-21 Display of the LUN mappings

```
itsotuc2> lun show -m
```

LUN path	Mapped to	LUN ID	Protocol

/vol/db2data/lun	aixdb2	0	FCP
/vol/db2logs/lun	aixdb2	1	FCP

```
itsotuc2>
```

Setup is complete on the IBM System Storage N series. Now, we will move onto the database server.

10.3.3 Accessing the LUNs on the database server

Now that the LUNs are created, the AIX server should be able to detect them as physical devices. As such, we will create volume groups and file systems on these devices. This means that, when using the storage system through a SAN, the file system management is handled by AIX. When setting up the storage system using NFS, however, the IBM System Storage N series handles the file system.

1. First, log onto the AIX server as the root user.
2. Ensure that your fibre-channel driver is loaded by issuing the following command:

```
cfgmgr -l device
```

where *device* is the name of the device corresponding to your fibre-channel adapter. You can determine which device corresponds to your fibre-channel adapter by issuing the **lsdev** command. In our environment, we determined that the **fcs0** device corresponds to our fibre-channel adapter, so we issue the following command:

```
cfgmgr -l fcs0
```

3. List all disk devices known to AIX using the following command:

```
lsdev -C -c disk
```

You should see an output similar to Example 10-22.

Example 10-22 Output listing of disk devices known to AIX

```
# lsdev -C -c disk
hdisk0 Available 1S-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 1S-08-00-9,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 1n-08-01 NetApp LUN
hdisk3 Available 1n-08-01 NetApp LUN
hdisk4 Available 1n-08-01 NetApp LUN
hdisk5 Available 1n-08-01 NetApp LUN
#
```

4. Upon execution, the **sanlun lun show** command shows the devices along with the assigned names (see Example 10-23).

Example 10-23 sanlun lun show command

```
# sanlun lun show
filer:          lun-pathname      device filename      adapter
protocol      lun size      lun state
itotuc2:      /vol/db2data/lun      hdisk3              fcs1
FCP           39.9g (42858446848)  GOOD
itotuc2:      /vol/db2logs/lun      hdisk4              fcs1
FCP           23.9g (25715277824)  GOOD
itotuc2:      /vol/db2datanfs/lun      hdisk5              fcs1
FCP           8.0g (8571060224)    GOOD
itotuc2:      /vol/db2logsnfs/lun      hdisk6              fcs1
FCP           8.0g (8571060224)    GOOD
itotuc2:      /vol/db2logsc1/lun      hdisk7              fcs1
FCP           23.9g (25715277824)  GOOD
```

```

    itsotuc2:  /vol/db2datacl/lun          hdisk8          fcs1
FCP          39.9g (42858446848)      G00D
    itsotuc2:  /vol/vol5/test1          hdisk9          fcs1
FCP          15m (15728640)          G00D
#

```

5. We make one volume group for the data LUN and another volume group for the logs LUN. We issue the following commands:

```
mkvg -f -y db2datavg hdisk2
```

```
mkvg -f -y db2logsvg hdisk3
```

Issue lsvg to verify that the volume groups were created (Example 10-24).

Example 10-24 Output showing the volume groups created

```

# lsvg
rootvg
db2datavg
db2logsvg
#

```

6. Now we want to determine the amount of free space on the disks. To do this, issue the following commands:

```
lspv hdisk2
```

```
lspv hdisk3
```

Example 10-25 shows the output of these commands on our test environment. The portion of the output that is important to us is the FREE PPs, which display the number of free physical partitions (as well as corresponding free space in MBs). You can see that hdisk2 has 637 free physical partitions (40768MB) and hdisk3 has 765 free physical partitions (24480MB). We will need the number of free MBs for step 8 on page 189.

Example 10-25 Listing of disk information showing the amount of free physical partitions

```

# lspv hdisk2
PHYSICAL VOLUME:  hdisk2          VOLUME GROUP:  db2datavg
PV IDENTIFIER:    0007041ab599cf0e VG IDENTIFIER   0007041a00004c000000010bb
5b5be28
PV STATE:         active
STALE PARTITIONS: 0
PP SIZE:          64 megabyte(s)  LOGICAL VOLUMES: 1
TOTAL PPs:        638 (40832 megabytes) VG DESCRIPTORS: 2
FREE PPs:         637 (40768 megabytes) HOT SPARE:      no
USED PPs:         1 (64 megabytes)
FREE DISTRIBUTION: 128..127..127..127..128
USED DISTRIBUTION: 00..01..00..00..00
# lspv hdisk3
PHYSICAL VOLUME:  hdisk3          VOLUME GROUP:  db2logsvg
PV IDENTIFIER:    0007041ab5e344e7 VG IDENTIFIER   0007041a00004c000000010bb
5f3d9d3
PV STATE:         active
STALE PARTITIONS: 0
PP SIZE:          32 megabyte(s)  LOGICAL VOLUMES: 1
TOTAL PPs:        766 (24512 megabytes) VG DESCRIPTORS: 2
FREE PPs:         765 (24480 megabytes) HOT SPARE:      no

```



```

USED PPs:          1 (32 megabytes)
FREE DISTRIBUTION: 154..152..153..153..153
USED DISTRIBUTION: 00..01..00..00..00
#

```

7. Create the mount points for the disks using the following command:

```
mkdir -p mount_directory
```

where *mount_directory* is the directory where you want to mount the device or file system. In our test environment, we issue the following commands:

```
mkdir -p /mnt/db2datalun
```

```
mkdir -p /mnt/db2logslun
```

8. We now create the file systems on the disks using the **crfs** (create filesystem) command. However, we need to supply it with some extra options. The commands to enter are as follows:

```
crfs -v jfs -g db2datavg -m /mnt/db2datalun -a size=fs_size -a nbpi=4096 -a ag=64 -a bf=true -p 'rw'
```

```
crfs -v jfs -g db2logsvg -m /mnt/db2logslun -a size=fs_size -a nbpi=4096 -a ag=64 -a bf=true -p 'rw'
```

The explanation of the options specified are available in Table 10-1.

Table 10-1 Explanation of options specified in the **crfs** commands

Option	Definition
-v jfs	Specifies that we want a journaled file system.
-g <i>volume_group</i>	Specifies that we want to create a filesystem on a volume group named <i>volume_group</i> .
-m <i>mount_point</i>	We associate the file system with mount point <i>mount_point</i> .
-a size=<i>fs_size</i>	We specify the size of this file system to be <i>fs_size</i> .
-a nbpi=4096	Specifies the number of bytes per i-node (nbpi). The nbpi value is inversely proportional to the number of i-nodes on the file system.
-a ag=64	Specifies the allocation group size in MBs. An allocation group is a grouping of i-nodes and disk blocks.
-a bf=true	Specifies that we want a large file enabled file system.
-p 'rw'	Sets the permissions for the file system to read-write (rw).

In our test environment, we determined the amount of free space of the db2datavg and db2logsvg to be 40768MB and 24480MB, respectively. So, we issue the following commands:

```
crfs -v jfs -g db2datavg -m /mnt/db2datalun -a size=40768M -a nbpi=4096 -a ag=64 -a bf=true -p 'rw'
```

```
crfs -v jfs -g db2logsvg -m /mnt/db2logslun -a size=24480M -a nbpi=4096 -a ag=64  
-a bf=true -p 'rw'
```

The output of these commands are shown in Example 10-26.

Example 10-26 Output of the crfs commands.

```
# crfs -v jfs -g db2datavg -m /mnt/db2datalun -a size=40768M -a nbpi=4096 -a ag=64  
-a bf=true -p 'rw'  
Based on the parameters chosen, the new /mnt/db2datalun JFS file system is limited  
to a maximum size of 134217728 (512 byte blocks)  
New File System size is 83492864  
# crfs -v jfs -g db2logsvg -m /mnt/db2logslun -a size=24480M -a nbpi=4096 -a ag=64  
-a bf=true -p 'rw'  
Based on the parameters chosen, the new /mnt/db2logslun JFS file system is limited  
to a maximum size of 134217728 (512 byte blocks)  
New File System size is 50135040  
#
```

9. Mount the filesystems to the mount points using the following commands:

```
mount /mnt/db2datalun
```

```
mount /mnt/db2logslun
```

10. To make the mount points persistent across reboots, look for the mount points in `/etc/filesystems`, and add `mount = true`, as in Example 10-27.

Example 10-27 A snippet of the /etc/filesystems file

```
/mnt/db2datalun:  
dev          = /dev/lv00  
vfs          = jfs  
log          = /dev/loglv00  
options      = rw  
account      = false  
mount        = true  
  
/mnt/db2logslun:  
dev          = /dev/lv01  
vfs          = jfs  
log          = /dev/loglv01  
options      = rw  
account      = false  
mount        = true
```

11. In our environment, we already created a DB2 instance called `db2inst1`. Every DB2 instance on Unix and Linux environments has an associated user ID with the same name known as the *instance owner*. In our case, that user ID is `db2inst1`. The instance owner requires ownership of the file systems we mounted. To change the ownership of the `db2datalun` and `db2logslun` mounts to the instance owner `db2inst1`—whose primary group is `db2adm`—issue the following commands:

```
chown -R db2inst1:db2adm /mnt/db2datalun
```

```
chown -R db2inst1:db2adm /mnt/db2logslun
```

10.3.4 Creating the sample database on the LUNs

This section consists of the same steps as in the section, 10.2.4, “Creating the sample database on the mounts” on page 179. The only differences are that the mount name for the data is /mnt/db2datalun instead of /mnt/db2data and the mount name for the logs is /mnt/db2logslun instead of /mnt/db2logs.

Archived

DB2 Continuous Availability with IBM System Storage N series

This chapter provides information about how we can achieve Continuous Availability for DB2 database systems using the IBM System Storage N series storage solution. Database Continuous Availability is a state of functionality that virtually guarantees database system operational continuity in any downtime event. Continuous Availability concerns itself with the following:

- ▶ Recovering applications, data, and data transactions committed up to the moment of system loss.
- ▶ Seamless, 24x7 system availability that offsets any planned or unplanned downtime event.

High availability is generally achieved by making different system components redundant. The more components you make redundant, the higher the level of availability you can achieve.

The ways one can achieve high availability with a database system and especially with a DB2 database system is with the following:

- ▶ Disk redundancy
- ▶ Backup and recovery
- ▶ Failover support during server failure

11.1 Disk redundancy

Disk redundancy is achieved in today's environment with several technologies. In this section we focus on the redundancy technology of N series.

- ▶ RAID DP
- ▶ Spares
- ▶ Cluster Failover - two N series Nodes

1. One level of redundancy for High Availability is disk redundancy. This means that within the disk storage system you have a way to automatically reconstruct a failed or corrupted disk. This becomes possible with RAID, which is available in N Series. Another additional redundancy feature in N Series is RAID Double Parity, which further strengthens the High Availability of the disk by introducing an additional level of redundancy for maximum security and data availability in storage. More information about this topic can be found in Chapter 2, “Introduction to Data ONTAP” on page 31 of this book and in the Redpaper publication titled *IBM System Storage N Series Implementation of RAID Double Parity for Data Protection*, REDP-4169-00.

<http://www.redbooks.ibm.com/abstracts/redp4169.html?Open>

2. Another level of high or Continuous Availability, is the existence of spare drives in the IBM System Storage N series. The spare drives provide the capability to reconstruct the data of a failed drive in a spare drive.

– Disk failure with a hot spare disk

The storage system replaces the disk with a spare and reconstructs data. If a disk fails, the storage system does the following:

- Replaces the failed disk with a hot spare disk—if RAID DP is enabled and double disk failure occurs in the RAID group, the filer replaces each failed disk with a separate spare disk. Data ONTAP first attempts to use a hot spare disk of the same size as the failed disk. If no disk of the same size is available, Data ONTAP replaces the failed disk with a spare disk or the next available size up.
- Reconstructs, in the background, the missing data onto the hot spare disk or disks.
- Logs the activity in the /etc/messages file on the root volume.

Note: With RAID DP, the above processes can be carried out even in the event of simultaneous failure on two disks in a RAID group.

Attention: During reconstruction, file service can slow down.

After the filer is finished reconstructing data, replace the failed disk or disks with new hot spare disks as soon as possible so that hot spare disks are always available in the system.

3. The IBM System Storage N series can cluster using two nodes, sharing control of all the disks. This means that if one node fails, the other takes over all the disks of the failed node. This provides extra security in the case of a failure. Our test storage server, the N3700, has two nodes for high availability purposes. The basics of an IBM System Storage N series cluster consists of two nodes that can takeover/failover their resources or services to the associated counterpart nodes. This functionality assumes that all resources can be accessed by each node. That means both nodes must have access to all disks physically (cabling) and logically (see cluster software Figure 11-1 on page 195). The N3700 Model A20 combines both cluster nodes in one shelf; therefore, it does not require interconnect cables.

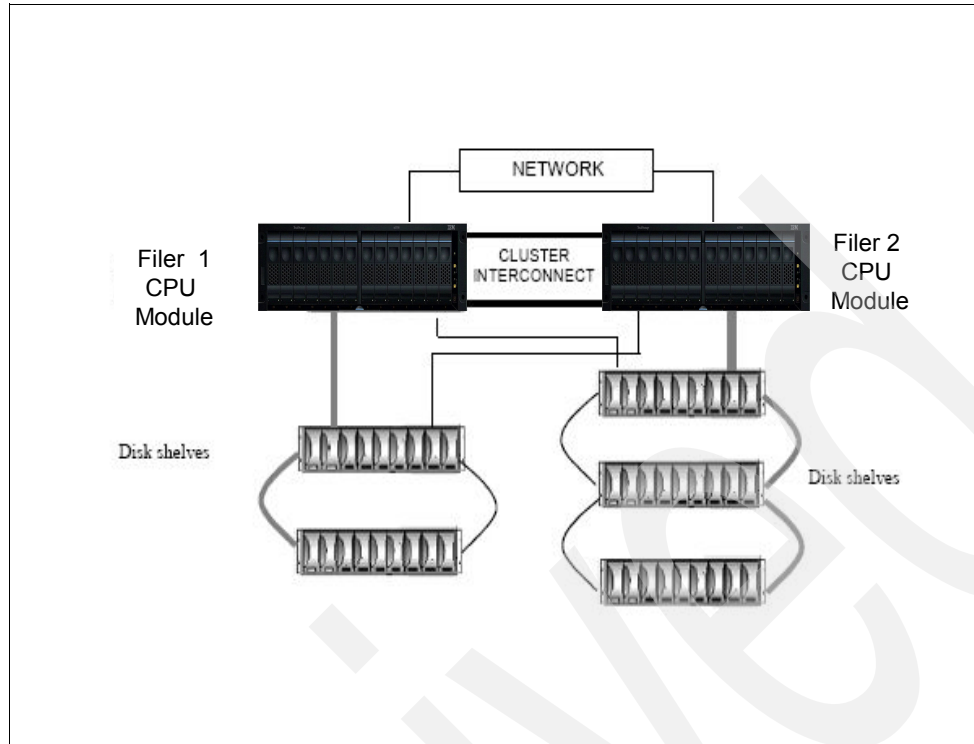


Figure 11-1 Cluster config

11.2 Backup and recovery

Backup and recovery is an important part of high availability and Continuous Availability because it allows the database to be recovered in case failure or corruption occurs and the database is not accessible. The way backup and recovery is implemented is extremely important for the Continuous Availability and the performance of the database system. We usually aim to use the smallest possible maintenance window for taking a backup so that the system can be online and un-disrupted for as long as possible.

Together with the traditional backup and recovery utilities available in DB2, N Series provides an additional feature for much faster and more frequent backups, and thus for smaller maintenance windows and higher availability. This feature is the snapshot.

11.2.1 How Snapshots work with DB2

The IBM System Storage N series Snapshot and SnapRestore technologies offer unique features to address the challenges presented by traditional backup and recovery methods.

The N series' Snapshots are a feature of the Write Anywhere File Layout (WAFL) storage virtualization technology that is a part of Data ONTAP, which is the microkernel that ships with every IBM System Storage N series. A Snapshot copy is a locally retained point-in-time "frozen" image of a WAFL volume that provides easy access to old versions of files, directory hierarchies, and logical unit numbers (LUNs). The high performance of the N series' Snapshot copies makes them highly scalable. A Snapshot copy takes only a few seconds to create (typically less than 1 second), regardless of the size of the volume or the level of activity on the IBM System Storage N series. After a Snapshot copy is created, changes to data objects are reflected in updates to the current version of the objects, as though the

Snapshot copy did not exist. Meanwhile, the Snapshot versions of the data remain completely stable. Therefore, Snapshot copies incur no performance overhead. Users can comfortably store up to 255 Snapshot copies per WAFL volume, all of which can be accessible as read-only, online versions of the data.

IBM System Storage N series' Snapshot can be used to create an online or offline database backup in seconds. The time needed to create a Snapshot copy is independent of the size of the database because Snapshot does not move data blocks. Snapshot copies vastly improve the frequency and reliability of backups because they incur virtually no performance overhead and can be safely created while a database is up and running. Customers running an IBM DB2 UDB database on an IBM System Storage N series typically take several Snapshot copies throughout the day.

An N Series Snapshot copy also provides key advantages for the database recovery operation. The SnapRestore feature of Data ONTAP provides a way to restore the entire database or parts of the database to the state it was in at the point in time when any available Snapshot copy was taken. Because no copying of data is involved, an incredible amount of time is saved as the file system is returned to its earlier state. The restore process can be done in a few minutes, independently of the size of the database. In addition, when low-impact Snapshot backups are created frequently throughout the day, fewer transaction logs need to be reapplied as part of the recovery process, resulting in a dramatic reduction in recovery time. In other words, the mean time to recover (MTTR), which consists of the time needed for restore and recovery, is reduced dramatically to several minutes, compared to several hours with conventional backup methods (Figure 11-2).

- **Database Recovery Scenario - An Example**

- 300 GB database and the entire database requires recovery
- Tape recovery time is 60 GB/hour
- Normal recovery time is 5 hours + log replay time
- SnapRestore reverts volume to same state as when backup was taken. Duration - 10 Secs
- Total recovery time: 10 Secs + log replay time

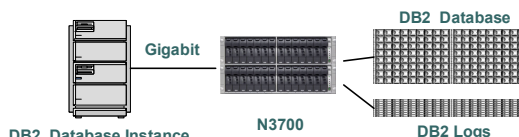


Figure 11-2 Database Recovery

Snapshot backups are stored on the same storage system as the database. Therefore, we recommend using Snapshot backups as a supplement, not a replacement, for backups to a second location, whether backing up to disk or to tape. Although backups to a second location are still necessary, there is only a slight probability that these backups are needed for a restore and recovery. Most restore and recovery actions can be handled by using SnapRestore. Restores from a second location (disk or tape) are necessary only in situations where the primary storage system holding the Snapshot copies is damaged or there is a need to restore a database from a backup that is no longer available in the form of a Snapshot copy—for instance, a two-month-old backup.

Note: Automatic snapshots should be turned off when using the Snapshot technology because we want to ensure data integrity in the storage server.

Key Advantages

The core of this approach to database backup and recovery is the use of Snapshot technology, which offers various features including stability, high performance, and storage efficiency. Following are the key advantages of the N series Snapshot technology in terms of backup and recovery:

- ▶ **Fast backup:** To meet the challenge posed by shrinking backup windows, the Snapshot feature of Data ONTAP is extremely useful. A Snapshot copy of a database can be created in a matter of seconds, regardless of the size of the database or the level of activity on the IBM System Storage N series. This dramatically reduces the database backup window from hours to seconds and helps DBAs to schedule frequent low-impact database backups.
- ▶ **Quick Recovery:** Using the Data ONTAP SnapRestore command, an entire database can be restored in a matter of seconds from a Snapshot backup. Because only changed data is restored, an incredible amount of time is saved when a database is returned to the state it was in at the time the Snapshot copy was created. Additionally, because Snapshot copies can be taken quickly and a large number of Snapshot copies can be retained, the amount of time needed to perform a roll-forward recovery operation against a database can be greatly reduced.
- ▶ **High availability:** The need for 24x7 availability is fast becoming a reality for organizations of all sizes. Companies cannot tolerate scheduled downtime, nor can they afford extended periods of slow system response that is often caused by traditional database backup methods. Snapshot copies, on the other hand, can be taken in a matter of seconds without any impact on system response time. This ensures high availability and uninterrupted system response.
- ▶ **High reliability:** The RAID architecture used for IBM System Storage N series is unique and provides greater reliability than direct-attached storage. If a RAID member disk fails, it is automatically reconstructed (using parity disk data) without any user intervention. IBM System Storage N series supports single parity as well as Double Parity RAID. Double Parity RAID, known as RAID-DP, is considered approximately 10,000 times more reliable than traditional RAID. For more details on RAID-DP, refer to the IBM Redbooks publication *IBM System Storage N Series*, SG24-7129.
- ▶ **Uninterrupted System Response:** Because a Snapshot copy is just a “frozen” image of the file system at a specific point-in-time, the process of creating a database backup with a Snapshot copy does not require the actual copying of data; therefore, it has virtually no impact on system response time.
- ▶ **Minimum Storage Requirement:** Two Snapshot copies taken in sequence differ from one another by the blocks added or changed in the time interval between the two. This block-incremental behavior minimizes the amount of storage space consumed.

11.3 Setting up Snapshots with DB2

In this section we detail the procedure to implement Snapshots with IBM DB2.

11.3.1 System names and assumptions

Our environment was set up as follows:

- ▶ The name of the DB2 administrator account is db2inst1.
- ▶ The database server IP address is 192.168.3.239.
- ▶ The name of the IBM DB2 UDB ESE v8.2 instance is db2inst1.
- ▶ The name of the database is sample.
- ▶ The name of the IBM System Storage N series server is itsotuc2.
- ▶ The name of the aggregate on the IBM System Storage N series is aggr0.
- ▶ The database's table data is stored on a FlexVol volume named db2datalun.
- ▶ The database's transaction logs are stored on a FlexVol volume named db2logslun.
- ▶ The database server is using the mount points named /mnt/db2data/lun and /mnt/db2logs/lun.

The steps and scripts outlined in this document may require significant modifications to run under your database/host UNIX environment.

11.3.2 Separate data from transaction logs files on N series

The N series Snapshot technology works at the volume level; therefore, when planning your database's physical layout, if Snapshot, SnapMirror, or SnapRestore technology will be incorporated into your backup and recovery strategy, you need to ensure that the database's data and transaction logs files are stored on separate FlexVols on the IBM System Storage N series. If a database recovery operation becomes necessary, maintaining separate FlexVols for data and logs will enable you to easily restore the database files from the appropriate Snapshot copy and roll-forward using the database's transaction logs. If your logs are not on separate volumes, then when you restore your database from a previous snapshot, you will over-write the new logs with the old logs and you will no longer be able to roll-forward and apply the latest changes to the database.

11.3.3 Overview of DB2 UDB components for backup and recovery

The main purpose of creating a database backup is to have a copy of mission-critical application data available in case the original data becomes corrupted or unavailable. Using a backup copy, application data can be returned to the state it was in at the point-in-time when the backup image was created. Before looking at how Snapshot copies can be used to create database backup images, it helps to be familiar with the components that DB2 UDB uses for backup and recovery.

Transaction log files

IBM DB2 keeps track of all changes made to a database's data and its objects. The changes are recorded in the transaction log files. Thus, transaction log files can be used to recover a

database. Depending upon how it is configured, a DB2 database can use two types of transaction log files:

Active/online log files: Log files with contents that were not yet applied to the database's data files are called active or online log files. The online log files are created in the active database log directory and are required to perform crash as well as roll-forward recovery.

Archive/offline log files: Log files with contents that were applied to the database's data files are called archive log files. They are not required for crash recovery, but they are crucial for roll-forward recovery. An active log file becomes a candidate for archiving as soon as its contents are written to the data files and may be moved automatically from the active log directory to an archive location.

Logging modes

IBM DB2 supports the following two different techniques for managing log files:

Circular logging: Only online logs are retained, and log files are used in round-robin fashion. The online logs are used only for crash recovery. The primary means of database recovery is version recovery from a full database backup copy. For circular logging, the `logarchmeth1` and `logarchmeth2` database configuration parameters are set to the value OFF.

Archive logging: Both the online and the archive logs are retained. Online logs are used for crash recovery. Both online and archive logs are used for roll-forward recovery. Upon commit, log files are closed and become available for offline archiving. Archival logging can be enabled by setting the `logarchmeth1` and `logarchmeth2` database configuration parameters to a value other than the value OFF.

Recovery history file

IBM DB2 UDB uses a special type of log file, called the recovery history file, as a repository for tracking various database activities such as backup, restore, roll-forward, alter, rename, quiesce tablespace, load, drop, reorganization of table, and update of table statistics. Entries in this file can be seen by executing the LIST HISTORY command.

Recovery methods

IBM DB2 supports following database recovery methods.

Crash recovery: Database transactions (or units of work) can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state. Crash recovery is the process by which the database is returned to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred. (Transaction records and corresponding commit records stored in transaction log files are used to return the database to a consistent state.) The order in which statements are rolled back is the reverse of the order in which they were originally executed.

Version recovery: This type of recovery allows for the restoration of a previous version of a database using a backup copy of the database that was created by the DB2 backup utility. This method of recovery is normally used for restoring a non-recoverable database. You can also use this method to restore a recoverable database if the DB2 restore utility is invoked with the WITHOUT ROLLING FORWARD option specified. A version recovery reconstructs an entire database using a backup copy created earlier.

Roll-forward recovery: Roll-forward recovery extends version recovery by using full database backups in conjunction with archive and active log files. When a roll-forward recovery

operation is performed, the database must first be restored from a backup copy, and then transaction records stored in log files are applied to the restored database. This procedure returns a database or a tablespace to the state it was in, at a particular point-in-time. The roll-forward recovery requires archival logging to be enabled.

Non-recoverable databases

An IBM DB2 UDB database is considered to be non-recoverable if it is using circular logging. For circular logging, only active logs required for crash recovery are retained. Version recovery is the primary method used to restore a damaged database. Roll-forward recovery is not possible, and such a database can be restored only to the state it was in at the time the backup image being used for recovery was created.

Recoverable databases

An IBM DB2 UDB database is considered to be recoverable if it has archive logging enabled and the `logarchmeth1` and `logarchmeth2` configuration parameters are set to a value other than the value `OFF`. The active logs as well as archive logs are available for recovery. Active logs are used for crash recovery, and archive logs make roll-forward recovery possible. If a database is recoverable, tablespace level backup and recovery are possible as well.

Temporarily suspending database I/O

To obtain a consistent backup for an online recoverable database using the N series product's Snapshot copy, all write operations must be suspended before invoking the Snapshot command. IBM DB2 UDB 7.1 (FixPak 2) and later provides support for temporarily suspending writes made by a DB2 UDB database. This support is provided by using the `SET WRITE SUSPEND` and `SET WRITE RESUME` commands. When the `SET WRITE SUSPEND` command is executed, all writes to a database and its corresponding transaction log files are suspended. Read-only transactions can continue executing; however, some transactions may wait if they require disk I/O (for example, flushing dirty pages from the buffer pool or flushing records from the log buffer). These transactions proceed normally after the write operations on the database are resumed.

To suspend write activity for a DB2 UDB database, you execute the following command on the database server:

```
db2 set write suspend for database
```

To resume write activity for a DB2 UDB database for which I/O was suspended, you execute the following command on the database server:

```
db2 set write resume for database
```

The db2inidb utility

The `db2inidb` utility was introduced in DB2 UDB EE FixPak 2 for version 7.1. Its purpose is to initialize a split mirror database. The syntax for the `db2inidb` command is as follows:

```
db2inidb <database alias> as <snapshot|standby|mirror> relocate using  
<config_file>
```

Note: Never run `db2inidb` on the primary database. You can run it only on the Snapshot database image.

The db2rfpn utility

The db2rfpn utility places a database that was restored from an offline backup into roll-forward pending state. After a database is restored from a backup image, it must be placed in the roll-forward pending state before a roll-forward recovery operation can be performed. The syntax for this command is as follows:

```
db2rfpn on <database alias>
```

For example, to put a database named myproddb in roll-forward pending state, you would execute the following command on the database server:

```
db2rfpn on sample
```

11.3.4 Backing up a DB2 UDB database using Snapshot technology

A conventional database backup to tape generates a significant load on both the database server and the storage system and takes several hours to complete. The system availability and performance are decreased while the backup is being performed.

On the other hand, a Snapshot-based backup copy can be created in a few seconds, and it has virtually no impact on the database server or the IBM System Storage N series. This low impact on the system enables DBAs to take Snapshot-based backups much more frequently, for example, every hour. The actual backup time is in seconds; therefore, frequent backups are possible.

Depending upon the retention policy used, there will always be a certain number of Snapshot copies available that can be used to restore the database. A higher frequency of backups provide a more flexible restore strategy and a shorter database restore time. The shorter the time frame between backups, the smaller the number of transaction logs that need to be applied during a roll-forward recovery operation.

An online as well as an offline backup of an IBM DB2 UDB database can be created by using a Snapshot copy. If the database remains online, it must be placed in write-suspend mode before the Snapshot copies of the FlexVol volumes are created. For offline backup, all application connections to the database need to be terminated and the database should be inactive. Write-suspend mode is not applicable for offline backup.

The steps required to create an offline and an online database backup for both recoverable and non-recoverable DB2 UDB databases are described in the following sections.

11.3.5 Accessing N series from the database server

You can use the **rsh** command to access the storage server. To do this, you need to grant access to the user ID of the database server which is going to send commands to the storage server. The rsh is only used within the scripts that automate the backup and restore procedure.

We need to first grant access to the db2inst1 user ID of the database server from which we are going to issue commands remotely to the storage server.

To grant access to the db2inst1, we need to update the /etc/hosts file of the IBM System Storage N series. This ensures that we add a line that gives permission to another server and user ID to access the storage server remotely via rsh. In order to update the /etc/hosts file we need to export the file to the database server and edit it in order to add the user ID and host name of the server who is going to access it.

You can do this via the GUI interface of N series as follows:

1. Open an Internet browser, and go to the following IP address of the storage server:
http://192.168.3.251/na_admin/
2. Click FilerView. The following panel appears:

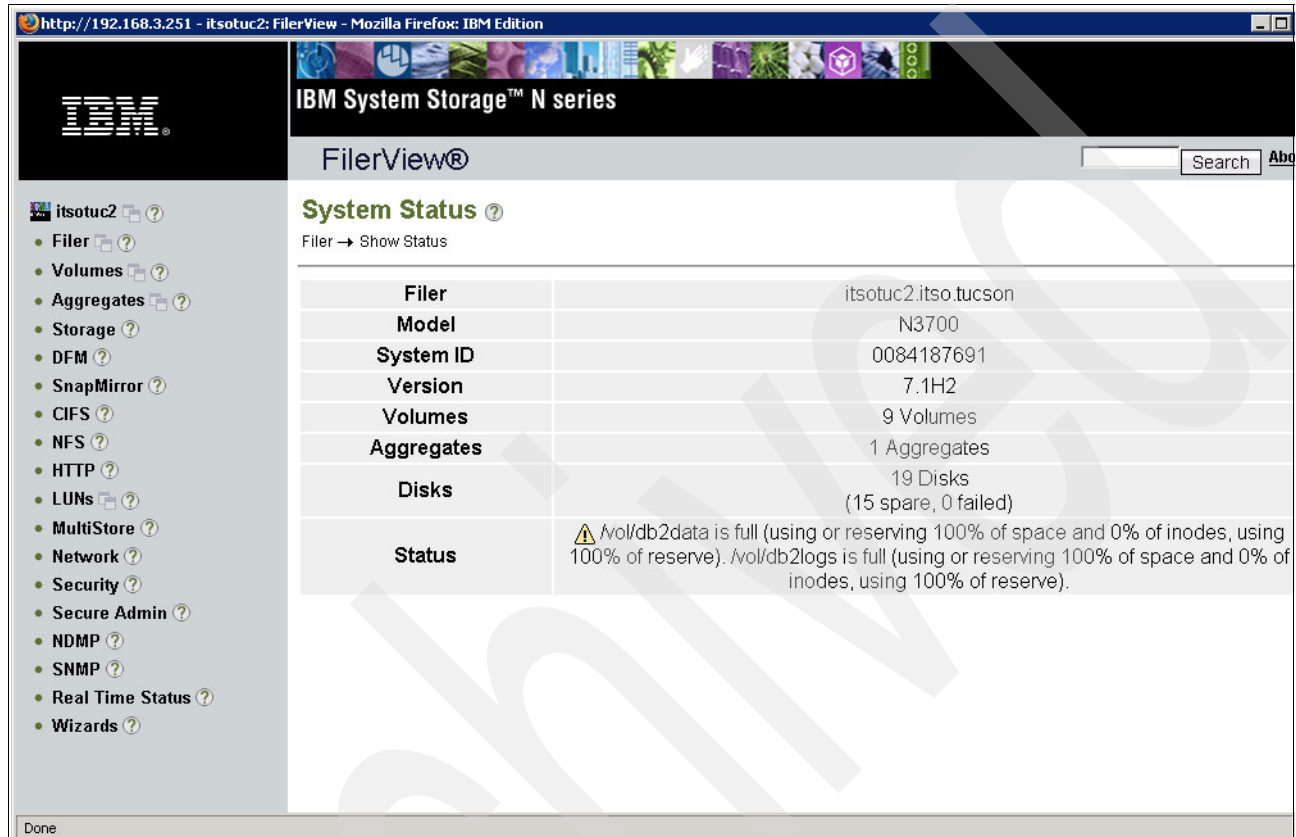


Figure 11-3 System Status panel

3. Click Security.
4. Next click Manage Rsh Access. Figure 11-4 on page 203 appears:

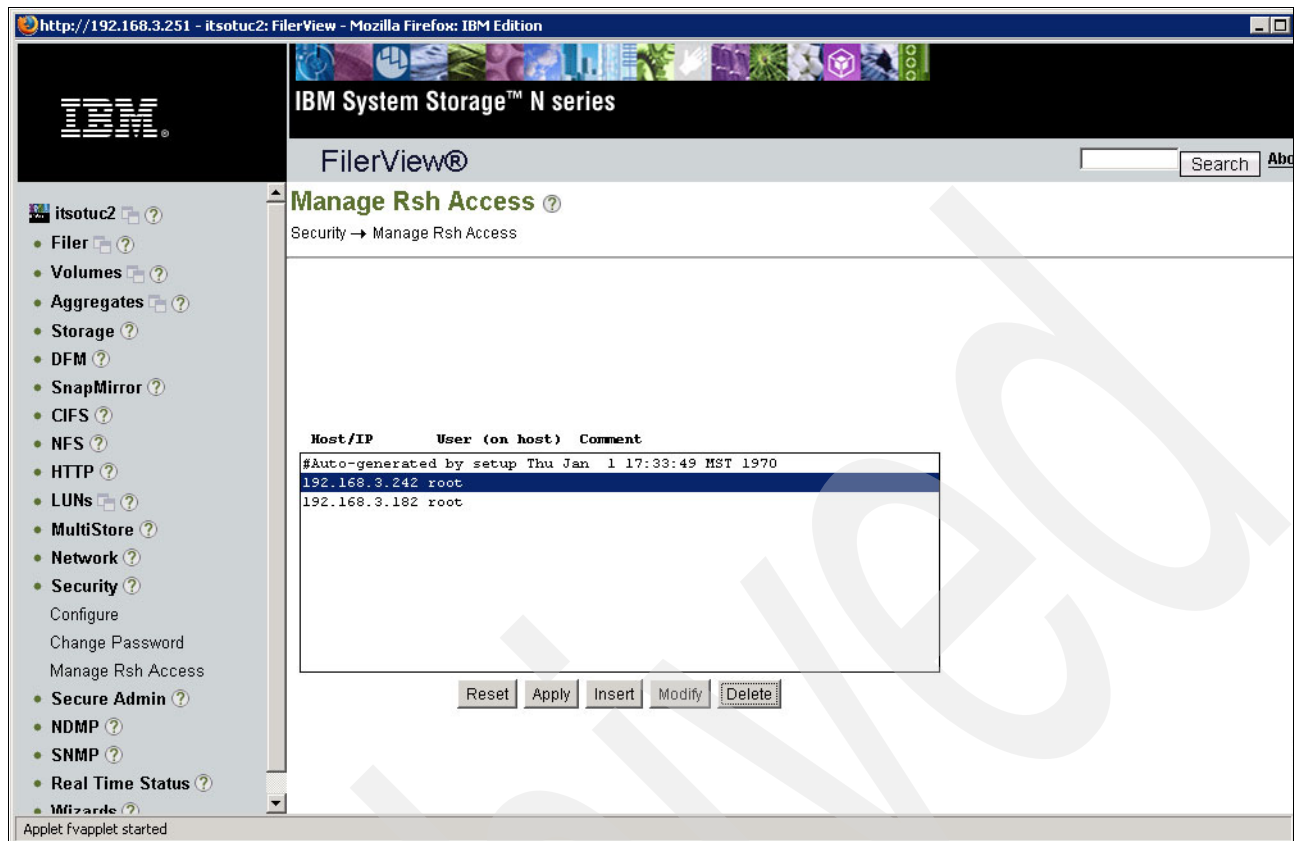


Figure 11-4 Remote Access Administration

5. At this point you want to add one more line to the list above, which will contain the user id db2inst1 and the host 192.168.3.239, which is the host name of the database server.
6. Click Insert. Figure 11-5 appears.

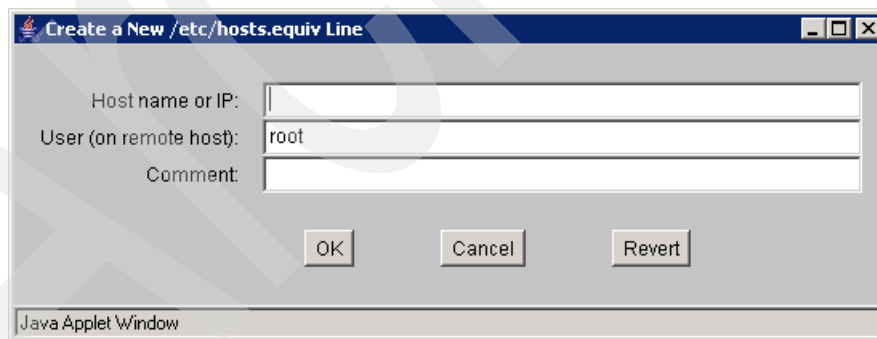


Figure 11-5 hosts.equiv editing

7. Type the following into the fields, as seen in Figure 11-6 on page 204:
Host IP : 192.168.3.239
User : db2inst1
8. Press Ok.

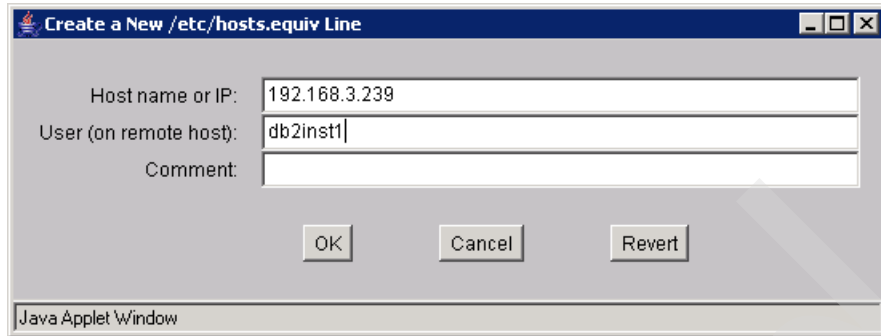


Figure 11-6 modifying hosts.equiv

9. After pressing the OK button you should be able to see your additional user ID authentication for Rsh access to the storage server (Figure 11-7).

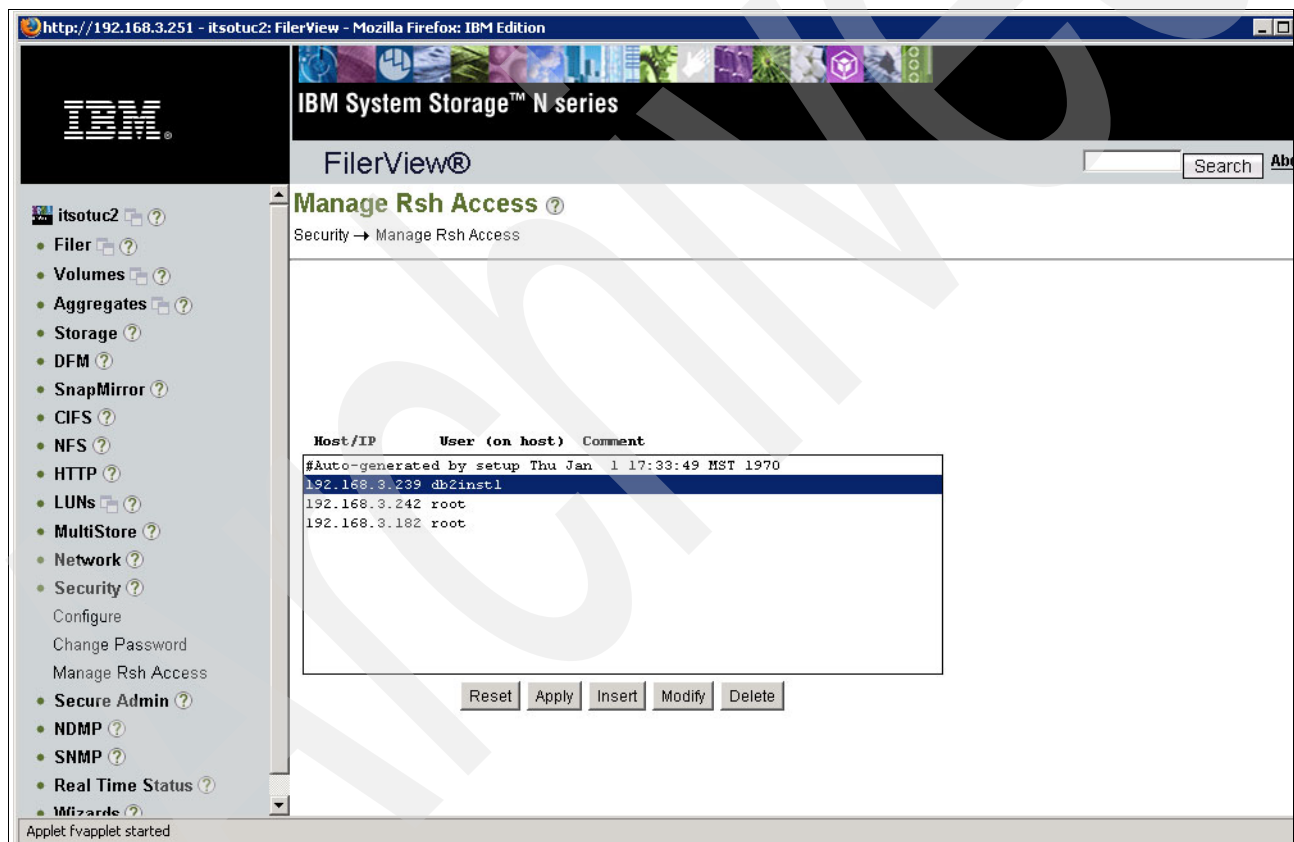


Figure 11-7 Results of modification

You can also give rsh access to a user using the command line and issuing the following commands:

On the N series, export the /etc directory:

```
export /vol/vol0/etc
```

```
assign 192.168.3.239 root access
```

```
mount
```

```
192.168.3.251:/vol/vol0/etc on /itsotuc2 type nfs (rw,addr=192.168.3.251)
```



```
exportfs  
/vol/vol0/etc -sec=sys,rw,root=192.168.3.239
```

Now that you have access via rsh, you can issue commands to your storage server from your database server. This is very useful when you automate the commands in the following sections via scripts.

Creating an offline database backup image with a Snapshot copy

A database is considered offline when all connections to it are terminated, and the database is no longer active. To take an offline backup of a DB2 UDB database, complete the following steps:

1. Terminate all the connections to the database by executing the following commands on the database server:
db2 force applications all
2. Create a database backup by creating a Snapshot copy for each IBM System Storage N series volume that is used by the database. A Snapshot copy of a FlexVol volume can be created by executing the following command on the N series product:

```
snap create <volume name> <snapshot name>
```

In our case we will take a snapshot of the data volume and the log volume. Note that we recommend that you take a snapshot of both the log and the data volumes. See Example 11-1. Although in an offline snapshot you cannot roll-forward, it is a good practice to keep the logs for a possible use of a log forward in case of accidental deletion of current logs:

Example 11-1 Snap create

```
snap create db2data db2data_snp.1  
snap create db2logs db2logs_snp.1
```

It is important to note that you can also create a Snapshot copy for a FlexVol volume by executing an **rsh** command from the database server.

With these two easy steps, you are finished with database backup. Now you can start using the database.

Automating the creation of the snapshot

To automate the creation of a cold snapshot, use the script in Example 11-2, which allows you to run snapshots often and to name them appropriately based on the time stamp.

Example 11-2 Snapshot script

```
# User-Defined Constants  
FILER1='itotuc2'  
FILER1_VOLUME1='db2data'  
FILER1_VOLUME2='db2logs'  
  
# Stop The DB2 Database Manager  
echo "\nStopping the DB2 Database Manager"  
db2stop force  
echo  
  
# Create The Date And Time String That Will Be Used To Name The Snapshots  
SNAPTIME=`date '+%m%d_%H%M%S'`
```

```

# Take A Snapshot Of Each Filer Volume Used
echo "Taking Snapshots ...\n"
rsh ${FILER1} -l db2inst1 snap create ${FILER1_VOLUME1} ${SNAPTIME} 2>&1 >
/dev/null
if [ $? = 0 ]
then
    echo "A Snapshot for volume '${FILER1_VOLUME1}' has been created"
fi

rsh ${FILER1} -l db2inst1 snap create ${FILER1_VOLUME2} ${SNAPTIME} 2>&1 >
/dev/null
if [ $? = 0 ]
then
    echo "A Snapshot for volume '${FILER1_VOLUME2}' has been created"
fi

# List All Snapshots That Exist On The Database Filer Volumes Used
echo "\nThe following Snapshots now exist on the filer volumes specified:\n"
echo "Filer 1 - (${FILER1})\n"
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME1}
echo
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME2}
echo

# Restart The DB2 Database Manager
echo "\nRestarting the DB2 Database Manager"
db2start

# Display A Message Saying The Operation Has Completed
echo "\nDone!\n"

```

Creating an online database backup image with a Snapshot copy

A database is considered active when one or more connections are established to it. To take an online backup of a DB2 UDB database, complete the following steps:

Temporarily suspend all write activity for the database by executing the commands in Example 11-3 on the database server:

Example 11-3 db2 commands

```

db2 connect to sample user db2inst1 using db2inst1

db2 set write suspend for database

```

It is important to note that while a DB2 database is in write suspended state, only write activities for the database are suspended; indeed, read operations still continue uninterrupted. The write activities will hold on as though a commit is not yet issued. If the autocommit option is set to OFF, then the application will continue its activity, but the write transactions are not committed until the write I/O is resumed.

When using a SAN or IP-SAN (FCP or iSCSI) configuration, the host system has control of the file system on the IBM System Storage N series. Because of this, file system integrity needs to be ensured before a Snapshot copy is created. Such integrity is ensured by executing the **sync** system command on the server. When executed, the **sync** command flushes all previously unwritten system buffers, including modified super blocks, modified inodes, and delayed block I/O to the disk.

For example, to flush all previously unwritten system buffers to disk before taking a Snapshot copy, open a telnet session to the database server, log-in as the root user, and execute the command in Example 11-4.

Example 11-4 sync command

```
sync
```

Create a database backup by creating a Snapshot copy for each IBM System Storage N series volume that holds the database's data. A Snapshot copy of a FlexVol volume can be created by executing the command in Example 11-5 on the IBM System Storage N series:

Example 11-5 snap create syntax

```
snap create <volume name> <snapshot name>
```

For example, to create a Snapshot copy for a FlexVol volume named db2data, execute the command in Example 11-6 on the IBM System Storage N series:

Example 11-6 snap create

```
snap create db2data db2data_snp.1
```

Now that one or more Snapshot copies are taken, resume writes to the database by executing the command in Example 11-7 on the database server:

Example 11-7 db2 set write resume syntax

```
db2 set write resume for database
```

In our case, we execute the command in Example 11-8.

Example 11-8 db2set for database sample

```
db2set write resume for sample
```

To automate this process, an example of a script, which can be used with the previously mentioned commands, is shown in Example 11-9.

Example 11-9 Sample script

```
# User-Defined Constants
```

```

DB_ALIAS='SAMPLE'

FILER1='itotuc2'
FILER1_VOLUME1='db2data'
FILER1_VOLUME2='db2logs'

# Connect To The Appropriate DB2 Database
echo "\nConnecting to database ${DB_ALIAS} ... \c"
db2 "connect to ${DB_ALIAS}" 2>&1 > error.log
if [ $? != 0 ]
then
    echo "\n ERROR - Unable to connect to ${DB_ALIAS}. Refer to the file
\c"
    echo "'error.log' for more information.\n"
    exit 1
else
    rm error.log
    echo "Done!\n"
fi

# Suspend All I/O To The Database (SET WRITE SUSPEND)
echo "Suspending I/O to database ${DB_ALIAS} (SET WRITE SUSPEND) ... \c"
db2 "set write suspend for database" 2>&1 > error.log
if [ $? != 0 ]
then
    echo "\n ERROR - Unable to suspend writing to ${DB_ALIAS}. Refer to \c"
    echo "the file 'error.log' for more information.\n"
    exit 1
else
    rm error.log
    echo "Done!\n"
fi

#Use the sync command to flash memory to disk for system integrity

sync

# Create The Date And Time String That Will Be Used To Name The Snapshots
SNAPTIME=`date '+%m%d_%H%M%S'`

# Take A Snapshot Of Each Filer Volume Used
echo "Taking Snapshots ...\n"
rsh ${FILER1} -l db2inst1 snap create ${FILER1_VOLUME1} ${SNAPTIME} 2>&1 >
/dev/null
if [ $? = 0 ]
then
    echo "A Snapshot for volume '${FILER1_VOLUME1}' has been created"
fi

rsh ${FILER1} -l db2inst1 snap create ${FILER1_VOLUME2} ${SNAPTIME} 2>&1 >
/dev/null
if [ $? = 0 ]
then
    echo "A Snapshot for volume '${FILER1_VOLUME2}' has been created"
fi

```

```

# Resume I/O To The Database (SET WRITE RESUME)
echo "\nResuming I/O to database ${DB_ALIAS} (SET WRITE RESUME) ... \c"
db2 "set write resume for database" 2>&1 > error.log
if [ $? != 0 ]
then
    echo "\n ERROR - Unable to resume writing to ${DB_ALIAS}. Refer to \c"
    echo "the file 'error.log' for more information.\n"
    exit 1
else
    rm error.log
    echo "Done!\n"
fi

# Disconnect From The Appropriate DB2 Database
echo "Disconnecting from database ${DB_ALIAS} ... \c"
db2 "disconnect ${DB_ALIAS}" 2>&1 > /dev/null
echo "Done!\n"

# List All Snapshots That Exist On The Database Filer Volumes Used
echo "\nThe following Snapshots now exist on the filer volumes specified:\n"
echo "Filer 1 - (${FILER1})\n"
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME1}
echo
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME2}

# Display A Message Saying The Operation Has Completed
echo "\nDone!\n"

```

Restoring a DB2 UDB database using SnapRestore technology

As stated in an earlier section of this document, you can easily integrate the N series product's SnapRestore technology into your existing DB2 UDB database backup and recovery strategy. Maintaining separate volumes on the IBM System Storage N series to hold the database's data and transaction log files provides more flexibility for the database restore operation. We recommend creating a snapshot of the log volumes, even if your database is not recoverable and you cannot roll-forward the logs. It is possible that these logs may be used for a recovery process, and it is a good practice to keep them in snapshots.

The recovery process using IBM System Storage N series' SnapRestore technology is similar to that of recovery with a database Snapshot option; however, it is much faster because the IBM System Storage N series has merely to change a pointer on the file system rather than physically copying the files from the backup copy.

Restoring a non-recoverable database from an offline backup Snapshot

The IBM System Storage N series Data ONTAP SnapRestore feature allows database recovery from Snapshot copies. You can restore a non-recoverable and a recoverable DB2 database by using Snapshot copies of the FlexVol volumes that were created when the database was offline. This type of recovery is called version recovery. Complete the following basic steps to perform the version recovery of a database:

1. Disconnect all the applications, and stop the database by executing the commands in Example 11-10 on page 210.

Example 11-10 Disconnecting applications

```
db2 force applications all
```

```
db2stop
```

-
2. Remove all DB2 interprocess communication (IPC) resources by executing the command in Example 11-11 on the database server:

Example 11-11 Removing all DB2 IPC resources

```
~/sqlllib/bin/db2_kill
```

```
~/sqlllib/bin/ipclean db2inst1
```

-
3. Unmount the IBM System Storage N series volumes that are used for the database and mounted on the database server by executing the command in Example 11-12.

Example 11-12 un mount command syntax

```
umount <mount point1, mountpoint2,...>
```

For example, to un mount IBM System Storage N series volumes mounted on mount points named /mnt/db2datalun and /mnt/db2logslun, you execute the command in Example 11-13 on the database server:

Example 11-13 un mount command

```
umount /mnt/db2datalun /mnt/dblogslun
```

-
4. Recover the database by restoring the volumes that are used for the database's data and transaction logs using the Snapshot copies that you created in section 7.1, "Managing MSSQL with the N series product" on page 133. You can safely restore a database using Snapshot copies by executing the command in Example 11-14 on the IBM System Storage N series:

Example 11-14 Snap Restore syntax

```
snap restore -s <Snapshot name> <flexvol name>
```

For example, to restore a volume named dbdata from a Snapshot copy named dbdata_snp.1, execute the command in Example 11-15 on the IBM System Storage N series:

Example 11-15 Snap Restore command

```
snap restore -s db2data_snp.1 db2data
```

-
5. On execution, the snap restore command reverts the FlexVol volume to the state it was in when the Snapshot copy was created.

Skip this step if your system configuration uses NFS.

6. After FlexVol volumes are restored, check the LUN mappings and status by executing the commands in Example 11-16 on the IBM System Storage N series:

Example 11-16 LUN commands

```
lun show  
  
lun show -m
```

The LUNs should be online and have the same mapping as they had at the time the Snapshot copy was taken.

7. Refresh the Fibre Channel driver on the database server. For example, to refresh it on the AIX host, execute the command in Example 11-17.

Example 11-17 configuration manager execution

```
cfgmgr -l fcs0
```

For any other operating system (OS) and drivers please refer to the OS reference manual.

8. After the restore process, you need to mount the FlexVol volumes (or LUNs in the case of the FC/iSCSI) on the database server by executing the command in Example 11-18.

Example 11-18 mount syntax

```
mount <mount point name>
```

For example, to mount a FlexVol volume that has mount details specified in /etc/fstab of the Linux database host to a mount point named /mnt/dbdata, execute the command in Example 11-19.

Example 11-19 mount command

```
mount /mnt/db2data
```

After completing these steps, your database recovery is complete and you can connect to the database and continue your work.

To automate the work above, you can use the example script shown in Example 11-20.

Example 11-20 Script example

```
# User-Defined Constants  
FILER1='itsotuc2'  
FILER1_VOLUME1='db2data'  
FILER1_VOLUME2='db2logs'  
  
# List All Snapshots That Exist On The Database Filer Volumes Used  
echo "\nThe following Snapshots exist on the filer volumes specified:\n"  
echo "Filer 1 - (${FILER1})\n"  
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME1}  
echo
```

```

rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME2}

# Ask The User To Provide A Snapshot Name
echo "\n\nWhat is the name of the Snapshot you would like to restore the
database from: \c"
read SNAPNAME

# Stop The DB2 Database Manager
echo "\nStopping the DB2 Database Manager"
db2stop force
echo

# Remove all DB2 inter-process communication resources from memory :
echo "\nCleaning processes from memory"
/home/db2inst1/sqllib/bin/db2_kill
/home/db2inst1/sqllib/bin/ipclean db2inst1
echo

# Unmount The File Systems Used
echo "Unmount all file systems that exist on the filer volumes being
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers
#used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
umount /mnt/db2datalun
umount /mnt/db2logslun

# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME1} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME1}
2>&1 > error.log
if [ $? != 0 ]
then
    echo " ERROR - Unable to restore volume ${FILER1_VOLUME1}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1
else
    rm -f error.log 2>&1 > /dev/null
fi

# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME2} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME2}
2>&1 > error.log
if [ $? != 0 ]
then
    echo " ERROR - Unable to restore volume ${FILER1_VOLUME2}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1

```



```

else
    rm -f error.log 2>&1 > /dev/null
fi

#Refresh the device driver for the fiber channel. Skip this step if you use
NFS.
echo "Refresh the Fiber Channel device driver"
cfgmgr -l fcs0
echo

# Remount The File Systems Used
echo "\nMount all file systems that exist on the filer volumes that were
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers #used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
mount /mnt/db2data1un
mount /mnt/db2logslun
echo

#Place the database in rollforward mode
echo "Start the database and resume write I/O"
db2start;
#db2 restart db sample write resume
echo

# Display A Message Saying The Operation Has Completed
echo "\nThe database has been restored. Run db2dart to verify the database
is usable.\n"

```

Restoring a non-recoverable database from an online backup Snapshot copy

Version recovery for a non-recoverable database can be performed using online Snapshot copies of the database's data and transaction logs volumes that were created when the database was active. Complete the following steps to perform the database recovery:

1. Disconnect all the applications, and stop the database by executing the commands shown in Example 11-21 on the database server:

Example 11-21 Disconnecting applications

```

db2 force applications all

db2stop

```

2. Remove all DB2 inter-process communications (IPC) resources by executing the command in Example 11-22 on page 214.

Example 11-22 Remove DB2 IPC resources

```
~/sqlllib/bin/db2_kill
~/sqlllib/bin/ipclean db2inst1
```

3. On the database server, become the root user and execute the command in Example 11-23 to unmount the IBM System Storage N series volumes mounted on the database server:

Example 11-23 un mount syntax

```
umount <mount point1, mountpoint2,...>
```

For example, to unmount the N Series product volumes mounted on mount points named /mnt/db2data and /mnt/db2logs on an AIX host, execute the command in Example 11-24 on the database server:

Example 11-24 un mount command

```
umount /mnt/db2data lun /mnt/db2logs lun
```

4. Recover the database by restoring the volumes that are used for the database using the Snapshot copies that are created, as described in section 7.2, “SnapManager” on page 133. A database volume can be restored by executing the command in Example 11-25 on the IBM System Storage N series:

Example 11-25 snap restore syntax

```
snap restore -s <snapshot name> <flexvol name>
```

For example, to restore a volume named db2data from a Snapshot copy named db2data_snp.1, execute the command in Example 11-26 on the IBM System Storage N series.

Example 11-26 snap restore command

```
snap restore -s db2data_snp.1 db2data
```

5. On execution, the snap restore command reverts the FlexVol volume to the state it was in when the Snapshot copy was created.

Skip this step if your system configuration uses NFS.

6. After FlexVol volumes are restored, check the LUN mappings and status by executing the commands in Example 11-27 on the IBM System Storage N series:

Example 11-27 LUN Mappings verification

```
lun show
lun show -m
```

The LUNs should be online and have same mapping as they had at the time the Snapshot copy was created.

7. Refresh the Fiber Channel driver on the database server. See Example 11-28.

Example 11-28 running configuration manager

```
cfgmgr -l fcs0
```

For other Operating Systems and drivers, refer to the OS reference manual.

8. After the restore process, you need to mount the FlexVol volumes (or LUNs in case of the FC/iSCSI) on the database server by executing the command in Example 11-29.

Example 11-29 mount syntax

```
mount <mount point name>
```

For example, to mount a FlexVol volume that has mount details specified in `/etc/fstab` of the Linux database host to a mount point named `/mnt/db2datalun`, execute the command in Example 11-30.

Example 11-30 mount command

```
mount /mnt/db2datalun
```

9. Now the restored database is ready, but the database Snapshot copies were created after the database I/O was suspended by executing the set write suspend command. Therefore the restored database is in write suspend mode. In order to access the database, you need to resume writes to the database by executing the command in Example 11-31.

Example 11-31 Resuming the database syntax

```
db2start;  
db2 restart db <database alias> write resume
```

For example, execute the command in Example 11-32 on the database server to resume write operation for a database named sample:

Example 11-32 resume write operation

```
db2start;  
db2 restart db sample write resume
```

After completing the steps described above, the database is restored and ready to use. Now you can run some data verification scripts to check your data see Example 11-33.

Example 11-33 verification script

```
# User-Defined Constants  
FILER1='itotuc2'  
FILER1_VOLUME1='db2data'  
FILER1_VOLUME2='db2logs'
```

```

# List All Snapshots That Exist On The Database Filer Volumes Used
echo "\nThe following Snapshots exist on the filer volumes specified:\n"
echo "Filer 1 - (${FILER1})\n"
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME1}
echo
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME2}


# Ask The User To Provide A Snapshot Name
echo "\n\nWhat is the name of the Snapshot you would like to restore the
database from: \c"
read SNAPNAME


# Stop The DB2 Database Manager
echo "\nStopping the DB2 Database Manager"
db2stop force
echo


# Remove all DB2 inter-process communication resources from memory :
echo "\nCleaning processes from memory"
/home/db2inst1/sqllib/bin/db2_kill
/home/db2inst1/sqllib/bin/ipclean db2inst1
echo


# Unmount The File Systems Used
echo "Unmount all file systems that exist on the filer volumes being
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers
#used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
umount /mnt/db2datalun
umount /mnt/db2logslun


# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME1} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME1}
2>&1 > error.log
if [ $? != 0 ]
then
    echo " ERROR - Unable to restore volume ${FILER1_VOLUME1}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1
else
    rm -f error.log 2>&1 > /dev/null
fi


# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME2} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME2}
2>&1 > error.log

```

```

if [ $? != 0 ]
then
    echo "  ERROR - Unable to restore volume ${FILER1_VOLUME2}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1
else
    rm -f error.log 2>&1 > /dev/null
fi

#Refresh the device driver for the fiber channel. Skip this step if you use
NFS.
echo "Refresh the Fiber Channel device driver"
cfgmgr -l fcs0
echo

# Remount The File Systems Used
echo "\nMount all file systems that exist on the filer volumes that were
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers #used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
mount /mnt/db2data1un
mount /mnt/db2logslun
echo

#Place the database in rollforward mode
echo "Start the database and resume write I/O"
db2start;
#db2 restart db sample write resume
echo

# Display A Message Saying The Operation Has Completed
echo "\nThe database has been restored. Run db2dart to verify the database
is usable.\n"

```

Restoring a non-recoverable database from an online backup Snapshot copy

Version recovery for a non-recoverable database can be performed using online Snapshot copies of the database's data and transaction logs volumes that were created when the database was active. Complete the following steps to perform the database recovery:

1. Disconnect all the applications, and stop the database by executing the commands in Example 11-34 on the database server:

Example 11-34 Disconnecting applications

```

db2 force applications all

db2stop

```

2. Remove all DB2 inter-process communications (IPC) resources by executing the command in Example 11-35.

Example 11-35 Remove IPC resources

```
~/sqlllib/bin/db2_kill  
~/sqlllib/bin/ipclean db2inst1
```

3. On the database server, become the root user, and execute the command in Example 11-36 to unmount the IBM System Storage N series volumes mounted on the database server:

Example 11-36 unmount syntax

```
umount <mount point1, mountpoint2,...>
```

For example, to unmount IBM System Storage N series volumes mounted on mount points named /mnt/db2datalun and /mnt/db2logslun on an AIX host, you would execute the command in Example 11-37 on the database server:

Example 11-37 unmount command

```
umount /mnt/db2datalun /mnt/db2logslun
```

4. Recover the database by restoring the volumes that are used for the database using the Snapshot copies that are created, as described in section 7.2, “SnapManager” on page 133. A database volume can be restored by executing the command in Example 11-38 on the IBM System Storage N series:

Example 11-38 snap restore syntax

```
snap restore -s <snapshot name> <flexvol name>
```

For example, to restore a volume named db2data from a Snapshot copy named db2data_snp.1, you would execute the command in Example 11-39 on the IBM System Storage N series:

Example 11-39 snap restore command

```
snap restore -s db2data_snp.1 db2data
```

5. On execution, the snap restore command reverts the FlexVol volume to the state it was in when the Snapshot copy was created.
Skip this step if your system configuration uses NFS.
6. After FlexVol volumes are restored, check the LUN mappings and status by executing the commands in Example 11-40 on page 219 on the IBM System Storage N series.

Example 11-40 LUN commands

```
lun show  
lun show -m
```

The LUNs should be online and have same mapping as they had at the time the Snapshot copy was created.

7. Refresh the Fiber Channel driver on the database server. See Example 11-41.

Example 11-41 Configuration manager

```
cfgmgr -l fcs0
```

For other Operating Systems and drivers, refer to the OS reference manual.

8. After the restore process, you need to mount the FlexVol volumes (or LUNs in case of the FC/iSCSI) on the database server by executing the command in Example 11-42.

Example 11-42 mount syntax

```
mount <mount point name>
```

For example, to mount a FlexVol volume that has mount details specified in /etc/fstab of the AIX database host to a mount point named /mnt/db2datalun, you would execute the command in Example 11-43.

Example 11-43 mount command

```
mount /mnt/db2datalun
```

9. Now the restored database is ready, but the database Snapshot copies were created after the database I/O was suspended by executing the set write suspend command. Therefore the restored database is in write suspend mode. In order to access the database, you need to resume writes to the database by executing the command in Example 11-44.

Example 11-44 resuming writes syntax

```
db2start;  
db2 restart db <database alias> write resume
```

For example, you would execute the command in Example 11-45 on the database server to resume write operation for a database named sample:

Example 11-45 resume write command

```
db2start;  
db2 restart db sample write resume
```

In this case we can also use the command in Example 11-46. This command should provide the same results as the “db2 restart db sample write resume.”

Example 11-46 db2inidb command

db2inidb sample as snapshot

After completing the steps described above, the database is restored and ready to use. Now you can run some data verification scripts to check your data.

To automate the steps above, you could use the script in Example 11-47.

Example 11-47 Sample script

```
# User-Defined Constants
FILER1='itsotuc2'
FILER1_VOLUME1='db2data'
FILER1_VOLUME2='db2logs'

# List All Snapshots That Exist On The Database Filer Volumes Used
echo "\nThe following Snapshots exist on the filer volumes specified:\n"
echo "Filer 1 - (${FILER1})\n"
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME1}
echo
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME2}

# Ask The User To Provide A Snapshot Name
echo "\n\nWhat is the name of the Snapshot you would like to restore the
database from: \c"
read SNAPNAME

# Stop The DB2 Database Manager
echo "\nStopping the DB2 Database Manager"
db2stop force
echo

# Remove all DB2 inter-process communication resources from memory :
echo "\nCleaning processes from memory"
/home/db2inst1/sqllib/bin/db2_kill
/home/db2inst1/sqllib/bin/ipclean db2inst1
echo

# Unmount The File Systems Used
echo "Unmount all file systems that exist on the filer volumes being
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers
#used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
umount /mnt/db2datalun
umount /mnt/db2logslun
```



```

# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME1} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME1}
2>&1 > error.log
if [ $? != 0 ]
then
    echo " ERROR - Unable to restore volume ${FILER1_VOLUME1}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1
else
    rm -f error.log 2>&1 > /dev/null
fi

# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME2} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME2}
2>&1 > error.log
if [ $? != 0 ]
then
    echo " ERROR - Unable to restore volume ${FILER1_VOLUME2}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1
else
    rm -f error.log 2>&1 > /dev/null
fi

#Refresh the device driver for the fiber channel. Skip this step if you use
NFS.
echo "Refresh the Fiber Channel device driver"
cfgmgr -l fcs0
echo

# Remount The File Systems Used
echo "\nMount all file systems that exist on the filer volumes that were
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers #used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
mount /mnt/db2data1un
mount /mnt/db2logslun
echo

#Place the database in rollforward mode
echo "Start the database and resume write I/O"
db2start;
db2 restart db sample write resume
echo

# Display A Message Saying The Operation Has Completed

```

```
echo "\nThe database has been restored. Run db2dart to verify the database
is usable.\n"
```

Restoring a recoverable database from an offline backup Snapshot copy

If roll-forward recovery is needed for a recoverable database from an offline backup copy, then the recovery steps vary slightly. You need to bring the database to roll-forward pending state by explicitly using the db2rften utility after the database is restored. In order to perform database recovery using SnapRestore, Snapshots copies for the volumes that hold the database's data and that are created as described in the section for offline Snapshot creation, are required. Make sure that you restore only the database's data volumes and *not the database's logs volumes*. Complete the following steps to recover the database:

1. Disconnect all the applications, and stop the database by executing the commands in Example 11-48.

Example 11-48 Disconnecting applications

```
db2 force applications all
db2stop
```

2. Remove all DB2 inter-process communication (IPC) resources by executing the commands in Example 11-49 on the database server.

Example 11-49 Removing DB2 IPC resources

```
~/sqlllib/bin/db2_kill
~/sqlllib/bin/ipclean db2inst1
```

3. Unmount the IBM System Storage N series' volumes that hold the database's data and that are mounted on the database server by executing the command in Example 11-50.

Example 11-50 unmount syntax

```
umount <<mount Point1> <mount point2> ..>
```

In our case we only unmount the data volume. See Example 11-51.

Example 11-51 unmount command

```
umount /mnt/db2data1un
```

4. As the first step in the roll-forward recovery, restore the volumes that are used for the database's data from the Snapshot copies created as described in section 7.1, "Managing MSSQL with the N series product" on page 133. A volume can be restored by executing the command in Example 11-52 on page 223 on the IBM System Storage N series.

Example 11-52 snap restore syntax

```
snap restore -s <snapshot name> <flexvol name>
```

For example, to restore a volume named dbdata from a Snapshot copy named dbdata_snp.1, you would execute the command in Example 11-53 on the IBM System Storage N series.

Example 11-53 snap restore

```
snap restore -s db2data_snp.1 db2data
```

On execution, the snap restore command reverts the FlexVol volume to the state it was in when the Snapshot copy was created.

It is important to note that for roll-forward recovery, only volumes that hold the database's data are restored. Volumes that hold the database's transaction and archive logs are not restored.

Skip this step if your system configuration uses NFS.

5. After FlexVol volumes are restored, check the LUN mappings and status by executing the commands in Example 11-54 on the IBM System Storage N series.

Example 11-54 Checking LUN mappings

```
lun show
```

```
lun show -m
```

The LUNs should be online and have the same mapping they had at the time the Snapshot copy was created.

6. Refresh the Fiber channel driver on the database server. See Example 11-55.

Example 11-55 Configuration manager

```
cfgmgr -l fcs0
```

7. After restoring the database's data volumes, mount them on the database server by executing the mount command. See Example 11-56.

Example 11-56 mount syntax

```
mount <mount point name>
```

For example, to mount a volume (or LUN device) that holds the database's data to a mount point named /mnt/db2datalun on the AIX host, you would execute the command in Example 11-57.

Example 11-57 mount command

```
mount /mnt/db2datalun
```

This command assumes that you specified mount details, including mount options for the volume (or LUN) in the /etc/fstab file on the database server.

8. Now the database is restored to the state it was in at the time that backup Snapshots copies were created, but recovery is not complete yet. To recover the database without any data loss, you need to perform roll-forward recovery. The database at this state is not in roll-forward pending state; therefore, you need to explicitly bring the database to roll-forward pending state by executing the following command on the database server.

Example 11-58 Roll-forward syntax

```
db2rfpen on <database alias>
```

For example, to put a database named myproddb in roll-forward pending state, you would execute the command in Example 11-59 on the database server.

Example 11-59 Roll-forward command

```
db2rfpen on sample
```

9. After the database is in roll-forward pending state, reapply the logs and perform roll-forward recovery by executing the commands in Example 11-60 on the database server.

Example 11-60 Roll-forward recovery

```
db2start
```

```
db2 "rollforward database sample to end of logs and complete"
```

```
db2 connect to sample
```

After completing these steps, the database is restored and ready to use. Now you can run some verification scripts to check your data. The script in Example 11-61 automates this process:

Example 11-61 Sample script

```
# User-Defined Constants
FILER1='itsotuc2'
FILER1_VOLUME1='db2data'
FILER1_VOLUME2='db2logs'

# List All Snapshots That Exist On The Database Filer Volumes Used
echo "\nThe following Snapshots exist on the filer volumes specified:\n"
echo "Filer 1 - (${FILER1})\n"
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME1}
echo
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME2}

# Ask The User To Provide A Snapshot Name
echo "\n\nWhat is the name of the Snapshot you would like to restore the
database from: \c"
read SNAPNAME
```

```

# Stop The DB2 Database Manager
echo "\nStopping the DB2 Database Manager"
db2stop force
echo

# Remove all DB2 inter-process communication resources from memory :
echo "\nCleaning processes from memory"
/home/db2inst1/sqllib/bin/db2_kill
/home/db2inst1/sqllib/bin/ipclean db2inst1
echo

# Unmount The File Systems Used
echo "Unmount all file systems that exist on the filer volumes being
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers
#used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
umount /mnt/db2datalun

# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME1} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME1}
2>&1 > error.log
if [ $? != 0 ]
then
    echo "  ERROR - Unable to restore volume ${FILER1_VOLUME1}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1
else
    rm -f error.log 2>&1 > /dev/null
fi

#Refresh the device driver for the fiber channel. Skip this step if you use
NFS.
echo "Refresh the Fiber Channel device driver"
cfgmgr -l fcs0
echo

# Remount The File Systems Used
echo "\nMount all file systems that exist on the filer volumes that were
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers #used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
mount /mnt/db2datalun
echo

```

```
#Place the database in rollforward mode
echo "Using the db2rfdpn command to put the database in roll forward mode"
db2rfdpn on sample
echo

#Rollforward to the end of logs
echo "Rollforwarding to the end of logs"
db2start;
db2 "rollforward database sample to end of logs and complete"

# Restart The DB2 Database Manager
echo "\nRestarting the DB2 Database Manager"
db2start

# Display A Message Saying The Operation Has Completed
echo "\nThe database has been restored. Run db2dart to verify the database
is usable.\n"
```

Restoring a recoverable database from an online backup Snapshot copy

Roll-forward recovery for a recoverable database is a two-step process. In the first step, the database is restored from a full database backup, and in the second step, transaction logs are reapplied. For roll-forward recovery, it is necessary that the database is running in archive mode and that active as well as archived logs are available. A database is running in archive mode when the LOGRETAIN parameter in the database configuration is set to RECOVERY, or when the USEREXIT is set to a destination where a program will archive the logs or LOGRETAIN and USEREXIT are set. To get the database configuration, issue the command in Example 11-62.

Example 11-62 Getting database configuration

```
db2 connect to sample;

db2 get db cfg
```

Next, check that the LOGRETAIN or the USEREXIT parameters or both are set to a proper value other than OFF.

In order to recover such a database using the SnapRestore feature, Snapshot copies for the volumes that hold the database's data and that are created as described in section 7.2, "SnapManager" on page 133 are required. Complete the following steps to recover the database:

1. Disconnect all the applications, and stop the database by executing the commands in Example 11-63.

Example 11-63 Disconnecting applications

```
db2 force applications all

db2stop
```

2. Remove all DB2 inter-process communication (IPC) resources by executing the commands in Example 11-64 on the database server.

Example 11-64 Removing DB2 IPC resources

```
~/sqlllib/bin/db2_kill  
~/sqlllib/bin/ipclean db2inst1
```

3. Unmount the IBM System Storage N series' volumes that hold database data by executing the command in Example 11-65 on the database server. In our test system we were able to mount and unmount our volumes from within the database user ID, and we did not have to switch to root for the mounting and unmounting. Nevertheless, it may be the case in some systems that you have to be root to be able to mount or unmount the storage system volumes. In this case please change the script in Example 11-65 accordingly.

Example 11-65 unmount syntax

```
umount <<mount Point1> <mount point2> .. >
```

For example, to unmount an IBM System Storage N series volume mounted on a mount point named /mnt/db2datalun on the AIX host, execute the command in Example 11-66 on the database server.

Example 11-66 unmount command

```
umount /mnt/db2datalun
```

4. As the first step in the roll-forward recovery, restore the volumes that are used for the database data from the Snapshot copies that are created as described in section 7.2, "SnapManager" on page 133. A volume can be restored by executing the command in Example 11-67 on the IBM System Storage N series.

Example 11-67 snap restore syntax

```
snap restore -s <snapshot name> <flexvol name>
```

For example, to restore a volume named db2data from a Snapshot copy named db2data_snp.1, you would execute the command in Example 11-68 on the IBM System Storage N series.

Example 11-68 snap restore command

```
snap restore -s db2data_snp.1 db2data
```

On execution, the snap restore command reverts the FlexVol volume to the state it was in when the Snapshot copy was created.

It is important to note that for roll-forward recovery, only volumes that hold the database's data are restored. Volumes that hold the database's transaction and archive logs are not restored.

Skip this step if your system configuration uses NFS.

5. After FlexVol volumes are restored, check the LUN mappings and status by executing the commands in Example 11-69 on the IBM System Storage N series.

Example 11-69 LUN status

```
lun show  
  
lun show -m
```

The LUNs should be on line and have the same mapping they had at the time that the Snapshot copy was taken.

6. Refresh the Fiber Channel driver. See Example 11-70.

Example 11-70 Configuration manager

```
cfgmgr -l fcs0
```

For other operating systems and drivers, please refer to the OS reference manual.

7. After restoring the database's data volumes, you need to mount them on the database server by executing the command in Example 11-71.

Example 11-71 mount syntax

```
mount <mount point name>
```

In our example, to mount a volume (or LUN device) that holds the database's data to a mount point named /mnt/db2datalun on the AIX host, you would execute the command in Example 11-72. This command assumes that you specified mount details, including mount options for the volume (or LUN) in the /etc/fstab file on the database server.

Example 11-72 mount command

```
mount /mnt/db2datalun
```

8. Now the database is restored to the state it was at the time the backup Snapshots copies were created, but to recover the database without any data loss, you need to perform roll-forward recovery by reapplying the archive and active logs. Before you perform roll-forward recovery, you need to place the database in roll-forward pending state by executing the command in Example 11-73 on the database server after you start the database server.

Example 11-73 Roll-forward command syntax

```
db2start ;  
  
db2inidb <database alias> as mirror
```

For example, to put our database named sample in roll-forward pending state, you would execute the command in Example 11-74 on page 229 on the database server.

Example 11-74 Roll-forward command

```
db2inidb sample as mirror
```

9. After the database is in roll-forward pending state, reapply the archive and active logs, and perform roll-forward recovery by executing the command in Example 11-75 on the database server.

Example 11-75 Roll-forward syntax

```
db2 "rollforward database <database alias> to end of logs and complete"
```

In our example, to perform roll-forward recover for a database named sample, you would execute the command in Example 11-76 on the database server.

Example 11-76 Roll-forward command

```
db2 "rollforward database sample to end of logs and complete"
```

After completing the steps described above, the database is restored and ready to use. Now you can run some verification scripts to check your data. You can use the script in Example 11-77, which automates these tasks.

Example 11-77 Script example

```
# User-Defined Constants
FILER1='itsotuc2'
FILER1_VOLUME1='db2data'
FILER1_VOLUME2='db2logs'

# List All Snapshots That Exist On The Database Filer Volumes Used
echo "\nThe following Snapshots exist on the filer volumes specified:\n"
echo "Filer 1 - (${FILER1})\n"
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME1}
echo
rsh ${FILER1} -l db2inst1 snap list ${FILER1_VOLUME2}

# Ask The User To Provide A Snapshot Name
echo "\n\nWhat is the name of the Snapshot you would like to restore the
database from: \c"
read SNAPNAME

# Stop The DB2 Database Manager
echo "\nStopping the DB2 Database Manager"
db2stop force
echo

# Remove all DB2 inter-process communication resources from memory :
echo "\nCleaning processes from memory"
/home/db2inst1/sqllib/bin/db2_kill
/home/db2inst1/sqllib/bin/ipclean db2inst1
echo
```

```

# Unmount The File Systems Used
echo "Unmount all data file systems that exist on the filer volumes being
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers
#used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
umount /mnt/db2datalun

# Restore The Filer Volume Using The Snapshot Specified
echo "\nRestoring volume ${FILER1_VOLUME1} using Snapshot ${SNAPNAME}"
rsh ${FILER1} -l db2inst1 snap restore -f -s ${SNAPNAME} ${FILER1_VOLUME1}
2>&1 > error.log
if [ $? != 0 ]
then
    echo "  ERROR - Unable to restore volume ${FILER1_VOLUME1}. Refer to the
\c"
    echo "file 'error.log' for more information.\n"
    exit 1
else
    rm -f error.log 2>&1 > /dev/null
fi

#Refresh the device driver for the fiber channel. Skip this step if you use
NFS.
echo "Refresh the Fiber Channel device driver"
cfgmgr -l fcs0
echo

# Remount The File Systems Used
echo "\nMount all file systems that exist on the filer volumes that were
restored."
#echo "NOTE: This operation must be performed by the root user on all
servers #used \c"
#echo "by the database!"
#echo "Press the Enter key when ready to continue ... \c"
#read READY
mount /mnt/db2datalun
echo

#Place the database in rollforward mode
echo "Using the db2inidb command to put the database in roll forward mode"
db2start;
db2inidb sample as mirror
echo

#Rollforward to the end of logs
echo "Rollforwarding to the end of logs"
db2 "rollforward database sample to end of logs and complete"

```

```
# Display A Message Saying The Operation Has Completed
echo "\n\nThe database has been restored. Run db2dart to verify the database
is usable.\n"
```

Failover support during server failure

A very important part of high availability, is the ability of a database system to be active in a very fast time after a primary server failure. There are many ways in the database and hardware technology to achieve this, and one way is disk replication. This scenario is possible in a few variations, but the basic idea behind this is that the data can be replicated very fast from one storage server to the other. The servers can be continuously synchronized and replicated over time, so that when a failure on the primary server or storage server occurs, the secondary server or storage takes over. This technique can be very fast and very reliable, but also very expensive with traditional storage systems. N series has the advantage of providing this technology at a lower cost.

The other advantage of this technology, in general, is that all data can be replicated to the other storage server, in contrast with data replication or log replication, which only replicates log-based transactions. This means that if the DBA is making changes to the DDL or to the database parameters, these changes are not replicated automatically with data or log replication. Storage replication, on the contrary, provides this automation because data is replicated at the physical level and not at the database level.

11.4 Conclusion

The IBM System Storage N series offers DBAs a compelling advantage in terms of database backup and recovery. By taking Snapshot copies frequently and keeping an appropriate number of Snapshot copies online, the DBA can restore a database without incurring the overhead normally associated with recovery from tape. Additionally, backup and recovery performance is dramatically improved—the process of creating or restoring from a Snapshot copy can be done in a fraction of the time needed when conventional methods are used.

Archived



Cloning a DB2 UDB database in the IBM System Storage N series environment

This chapter provides instructions for cloning a DB2 UDB database in the IBM System Storage N series environment.

12.1 Selecting a database server to access the cloned database

You need to select a database server that will be used to access the cloned database.

12.1.1 Selecting the production database server and database

Select the database server that accesses the production (parent) database. In this case, you can use an existing DB2 instance or create a new one using the same DB2 UDB V8 code as the production instance.

To create a new DB2 instance you need to complete the following steps:

1. Switch the authority to the user root and create a user named db2instc on the database server by executing the following command:

```
useradd -c "DB2 clone db instance owner" -u 710 -g db2adm -G db2adm  
db2instc -p db2instc
```

The new user will own the DB2 instance used to access the cloned database.

2. Next create a new DB2 instance using the same DB2 code as the production database instance by executing the following command on the database server:

```
[DB2InstallationPath]/instance/db2icrt -u [FencedUser]  
[InstanceName]
```

where,

- DB2InstallationPath identifies the directory where the DB2 UDB V8 code was installed.
- FencedUser identifies the ID of the user under which fenced user-defined functions and fenced stored procedures will run.
- InstanceName identifies the name that is to be assigned to the new instance

For example, if the DB2 UDB V8 were installed in /opt/IBM/db2/V8.1 directory, you would execute the following command on the database server to create a DB2 instance named db2instc:

```
/opt/IBM/db2/V8.FP11/instance/db2icrt -u db2instc db2instc
```

3. Check the list of instances and DB2 UDB code used by executing the following command on the database server:

```
/opt/IBM/db2/V8.FP11/instance/db2ilist -a
```

The output from the above command should look similar to Example 12-1.

Example 12-1 db2list output

db2inst1	32	/opt/IBM/db2/V8.1
db2instc	32	/opt/IBM/db2/V8.1

12.1.2 Select a database server with a different DB2 UDB version

Select a database server other than the production database server, which has a version of DB2 UDB installed that is different from the production DB2 UDB Version. In this case, you need to install the same DB2 UDB V8 code as the production database on the database server, and create a database instance as described in the previous section, “12.1.1, “Selecting the production database server and database” on page 234”. The new instance name can be the same as the production DB2 instance provided there is no other instance with the same name on this server.

12.1.3 Select a non production server without DB2 UDB installed

Install a database server other than the production database server that does not have DB2 UDB installed.

In this case, you need to install the same DB2 UDB V8 code as the production database on the database server and create a database instance as described in the 12.1.1, “Selecting the production database server and database” on page 234 section. The new instance name can be the same as the production DB2 instance.

12.2 Clone an offline database on the same storage system

Use the steps in the following sections to clone a database that is offline.

12.2.1 Bring the source database offline

To bring the database offline, terminate all the application connections to the database that are to be cloned by executing the following command on the database server:

```
db2 force applications all
```

For example, to terminate all application connections, you would execute the following command on the database server:

```
db2 force applications all
```

12.2.2 Create Snapshot copies of the database FlexVol volumes

Next, create a Snapshot copy of each FlexVol volume that is used for the production database by executing the following command on the storage system:

```
snap create [VolName] [SnapName]
```

where

- ▶ VolName identifies the name assigned to the FlexClone volume that is to be created.
- ▶ SnapName identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named dbdata_snap01 for a FlexVol volume named dbdata, you would execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp01
```

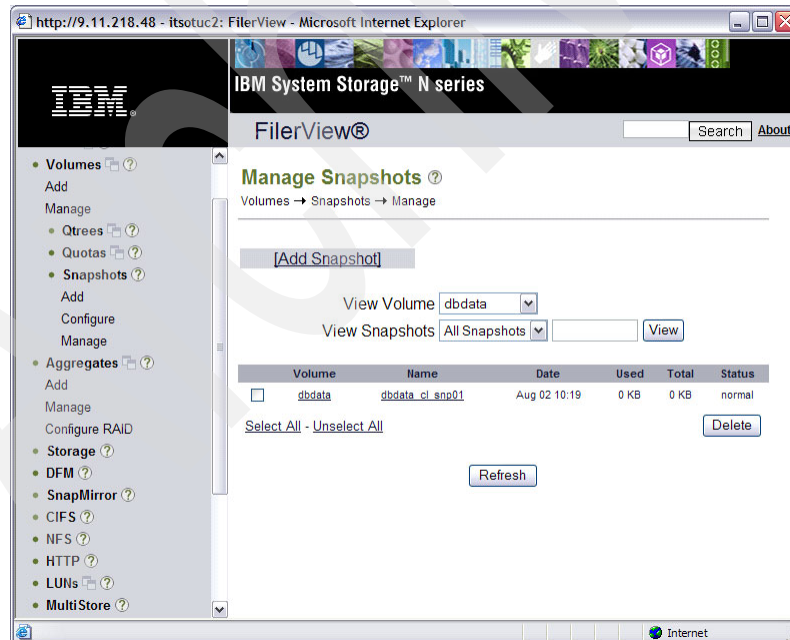


Figure 12-1 Snapshot results

We recommend that you develop a naming convention, and assign a meaningful name to the Snapshot copies that are created for cloning purpose.

12.2.3 Start the source database

After the Snapshot copies of the FlexVol volumes of the source database are created, you can connect to the source database and start using it.

12.2.4 Clone the FlexVol volumes

1. Next, create a clone of each FlexVol volume using the Snapshot copies created in section 12.2.2, “Create Snapshot copies of the database FlexVol volumes” on page 236. A clone volume can be created by executing the following command on the storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

where

- CloneVol identifies the name of the FlexClone volume that is being created.
- ParentVol identifies the name of the FlexVol volume that is source for the clone volume.
- ParentSnap identifies the name of the parent FlexVol volume snapshot that is used as source for the clone volume.

For example, to create a clone volume of a FlexVol volume named dblogs using the Snapshot copy named dbdata_cl_snp.01, you would execute the following command on the IBM System Storage N series:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.01
```

2. Verify the clone you just created with the following command:

```
vol status dbdata_cl
```

The output is seen in Example 12-2 on page 238.

```
itsotuc2> vol status dbdata_cl
      Volume State      Status      Options
      dbdata_cl online   raid_dp, flex  create_ucose=on,
                                     convert_ucose=on
                                     Clone, backed by volume 'dbdata', snapshot
'dbdata_cl_snp.01'
      Containing aggregate: 'aggr0'
```

Example 12-2 vol status example

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

Note: A Snapshot copy is not required to create a clone of a FlexVol volume—if you do not explicitly create a Snapshot copy and specify it when executing the vol clone command, a Snapshot copy will be implicitly created and used for the clone volume. A Snapshot copy created implicitly will have a system assigned name. We recommend explicitly creating a Snapshot and assigning it a meaningful name before creating a clone FlexVol volume.

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

12.2.5 Create an export entry for the clone volume

To mount a clone volume to the database server, you need to create an export entry for it in the /etc/exports file that resides on the IBM System Storage N series. The export entry can be created by executing the following command on the IBM System Storage N series:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

where,

- ▶ HostName identifies the name assigned to the database server.
- ▶ PathName identifies the name assigned to the flexible volume.

For example, to create an export entry for a clone volume named dbdata_cl and allow root access from the database server named hostdst for it, you would execute the following command on the storage system:

```
exportfs -p rw=hostdst,root=hostdst /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is used for the clone database.

You can verify your exports with FilerView. See Figure 12-2 on page 239.

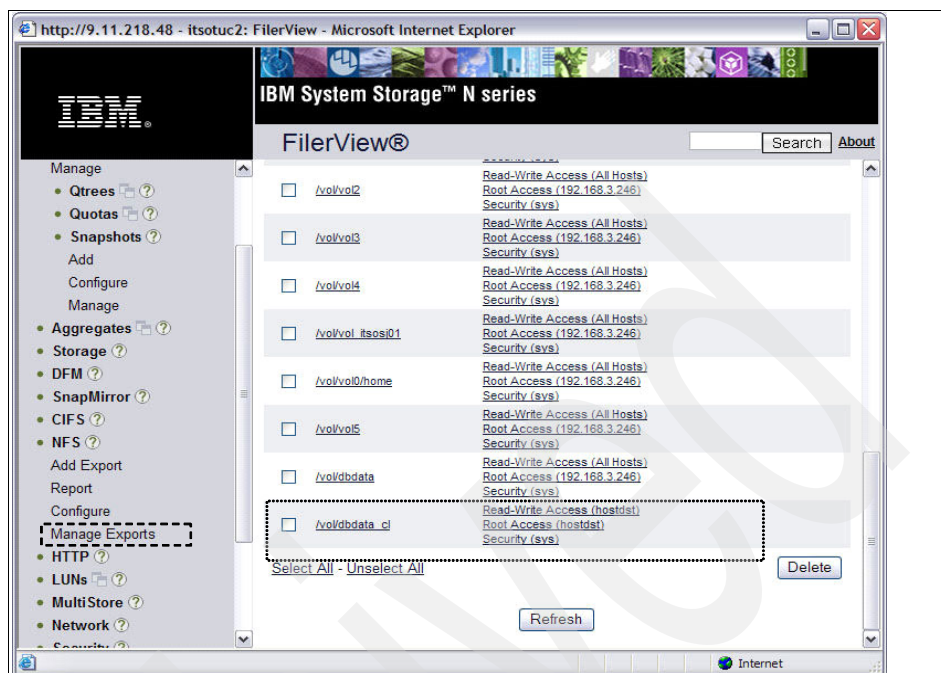


Figure 12-2 Verifying the exports

12.2.6 Mount the clone volumes

The clone database can be accessed from the same database server that is used to access the source database or from a completely different server. The scenarios described in this book were produced using a second database server to access the clone database.

In order to access the clone database, you need to mount the clone volumes to a database server. First, you need to create a mount point for each clone volume, and append a mount entry to the `/etc/fstab` file. The mount entry should specify the mount options, and it should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsz=32768,wsz=32768,bg,vers=3,tcp 0 0
```

where,

- ▶ `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- ▶ `FlexVolName` identifies the name assigned to the clone volume.

- MountPoint identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named `dbdata_cl` that resides on a storage system named `srcstore`, you would append the following entry to the `/etc/fstab` file on the database server:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsiz=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

where, MountPoint identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has mount entry specified in the `/etc/fstab` file, you would execute the following command on the second database server named `hostdst`:

```
mount /mnt/dbdata_cl
```

The database servers we used had a Linux operating system.

In order to operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

where,

- InstanceOwner identifies the name assigned to the user who owns the database instance.
- InstanceOwnerGroup identifies the name assigned to the user's group that owns the database instance.
- FileSystem identifies the name of the file system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, you would execute the following command on the second database server:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

12.2.7 Configuring the cloned database

The clone volumes created in section 12.2.4, “Clone the FlexVol volumes” on page 237 and mounted in section 12.2.6, “Mount the clone volumes” on page 239 are going to be used as the storage containers for the cloned database. You can skip the parts (1) and (2) of this step if the following two conditions are true for your environment:

- ▶ The name of the DB2 instance used for the clone database is the same as the production or source database instance name.
 - ▶ The mount points that are used to mount the clone volumes have the same name as the mount points that are used to mount volumes of the production database.
1. By default when the database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

[DBDir]/[InstanceName]/NODE000n

where,

- ▶ DBDir identifies the name assigned to the directory/device the database is created on.
- ▶ InstanceName identifies the name assigned to the DB2 instance the database belongs to.

For example, if the DB2 instance name is db2inst1 and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory.

/mnt/dbdata/db2inst1/NODE0000

For a database server, the DB2 instance name has to be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. In order to access the clone database from a different instance name you need to change the default tablespace container's path name by executing the following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n
  [DBDir]/[NewInstanceName]/NODE000n
```

where,

- ▶ DBDir identifies the name assigned to the directory/device the database is created on.

- ▶ OldInstanceName identifies the name assigned to the DB2 instance the production database belongs to.
- ▶ NewInstanceName identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

```
mv /mnt/dbdata_c1/db2inst1 /mnt/dbdata_c1/db2instc
```

2. Change the database name and tablespace container's header information using the db2inidb command. To do this, you need to create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to Example 12-3:

Example 12-3 Sample configuration file

```
DB_NAME=mydb,mydbc1
DB_PATH=/mnt/dbdata,/mnt/dbdata_c1
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_c1/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_c1/*
```

In this configuration file, the source database name is mydb, and it has its logs on /mnt/dblogs and its data on /mnt/dbdata. The clone database is to be renamed to mydbc1. It has its data on /mnt/dbdata_c1 and logs on /mnt/dblogs_c1. The source database instance name is db2inst1 and clone the database instance name is db2instc.

Save the configuration file as /home/db2inst1/dbrelocate.cfg, and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing `db2start`.

The production database was offline during the Snapshot creation process. Therefore, you can connect to the clone database and start using it.

12.2.8 Catalog the source database if necessary

If the cloned database is accessed from the same DB2 instance as the production database on the production database server, then on execution the **`db2relocatedb`** command uncatalogs the source database. Therefore, you need to catalog the source database by executing the following command on the database server:

```
db2 "catalog database [DatabaseName] as [DatabaseAlias] on [FileSystem]"
```

where,

- ▶ `DatabaseName` identifies the name assigned to the database that is being cataloged.
- ▶ `DatabaseAlias` identifies the alias name assigned to the database that is being cataloged.
- ▶ `FileSystem` specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named `mydb` that resides on file system named `/mnt/dbdata`, you would execute the following command on the database server:

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

12.2.9 Verify the database

After performing the above steps, you need to check the entire clone database for architectural correctness. You can do so by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

where, `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to test the cloned database named `mydbc1`, you would execute the following command on the database server:

```
db2dart mydbc1 /db
```

The db2dart utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the <\$HOME>/sqllib/db2dump/DART0000/ directory. The report has a summary at the end of the report. You can read the summary and obtain the errors, if any.

12.3 Clone an online database on the same storage system

After reading section 12.2, “Clone an offline database on the same storage system” on page 235 you should have a fair understanding of cloning an offline DB2 database. In this section, you learn how to clone an online DB2 database.

12.3.1 Bring the database into a consistent state—suspend writes

In this scenario, the source database is online. Therefore, to prevent partial page writes while database cloning is in progress, the database write operations need to be temporarily suspended by executing the following command on the database server:

```
db2 set write suspend for database
```

The SET WRITE SUSPEND FOR DATABASE command causes the DB2 database manager to suspend all write operations to tablespace containers and log files that are associated with the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The FlexVol volume clone process is completed very quickly, so the database does not need to stay in write suspend mode for more than a few seconds.

12.3.2 Create Snapshot copies of the database FlexVol volumes

Next, create a Snapshot copy of each FlexVol volume that is used for the production database.

A Snapshot copy of a FlexVol volume can be created by executing the following command on the storage system:

```
snap create [VolName] [SnapName]
```

where,

- ▶ VolName identifies the name assigned to the FlexClone volume that is to be created.
- ▶ SnapName identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_snap.1` for a FlexVol volume named `dbdata`, you would execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.1
```

We recommend that you develop a naming convention, and assign a meaningful name to the Snapshot copies that are created for cloning purpose.

12.3.3 Resume normal database operations—resume writes

After Snapshot copies are created, you need to resume write operations to the database by executing the following command on the database server:

```
set write resume for database
```

12.3.4 Clone the FlexVol volumes

Next, create a clone of each FlexVol volume using the Snapshot copies created in section 12.3.2, “Create Snapshot copies of the database FlexVol volumes” on page 244. A clone volume can be created by executing the following command on the storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

where,

- ▶ `CloneVol` identifies the name of the FlexClone volume that is being created.
- ▶ `ParentVol` identifies the name of the FlexVol volume that is source for the clone volume.
- ▶ `ParentSnap` identifies the name of the parent FlexVol volume snapshot that is used as a source for the clone volume.

For example, to create a clone volume of a FlexVol volume named `dblogs` using the Snapshot copy named `dbdata_cl_snp.01`, you would execute the following command on the IBM System Storage N series:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.01
```

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

Note: A Snapshot copy is not required to create a clone of a FlexVol volume—if you do not explicitly create a Snapshot copy and specify it when executing the vol clone command, a Snapshot copy will be implicitly created and used for the clone volume. A Snapshot copy created implicitly will have a system assigned name. We recommend explicitly creating a Snapshot and assigning it a meaningful name before creating a clone FlexVol volume.

12.3.5 Create NFS export entries for the cloned volumes

In order to mount a clone volume to the database server, you need to create an export entry for it in the `/etc/exports` file that resides on the storage system. The export entry can be created by executing the following command on the storage system:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

where,

- ▶ `HostName` identifies the database server name that will use these clone volumes as storage.
- ▶ `PathName` identifies the clone volume path.

For example, to create an export entry for a clone volume named `dbdata_cl` and allow root access to it from a database server named `dbhost1`, you would execute the following command on the storage system:

```
exportfs -p rw=dbhost1,root=dbhost1 /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is to be used for the database in the test environment.

12.3.6 Mount the cloned volumes

To access the clone database, you need to mount the clone volume to a database server. First, you create a mount point for each clone volume, and append a mount entry to the `/etc/fstab` file. The mount entry should specify the mount options, and it should look similar to Example 12-4:

Example 12-4 /etc/fstab

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs  
hard,rw,nointr,rsz=32768,wsz=32768,bg,vers=3,tcp 0 0
```

where,

- ▶ **StorageSystemName** identifies the name assigned to the storage system that is used for the database storage.
- ▶ **FlexVolName** identifies the name assigned to the clone volume.
- ▶ **MountPoint** identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named `dbdata_cl` that resides on a storage system named `srcstore`, you would append the following entry to the `/etc/fstab` file on the database server:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs  
hard,rw,nointr,rsz=32768,wsz=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

where, **MountPoint** identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has mount entry specified in the `/etc/fstab` file, you would execute the following command on the second database server, named `hostdst`:

```
mount /mnt/dbdata_cl
```

The database servers we used had Linux operating systems.

To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

where,

- ▶ **InstanceOwner** identifies the name assigned to the user who owns the database instance.
- ▶ **InstanceOwnerGroup** identifies the name assigned to the user's group that owns the database instance.
- ▶ **FileSystem** identifies the name of the file system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named /mnt/dbdata_cl, you would execute the following command on the second database server:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

12.3.7 Configuring the cloned database

The clone volumes created in section 12.3.4, “Clone the FlexVol volumes” on page 245 and mount in section 12.3.6, “Mount the cloned volumes” on page 246 of this section are going to be used as the storage containers for the cloned database. You can skip the parts (1) and (2) of this step if the following two conditions are true for you environment:

- ▶ The name of the DB2 instance used for the clone database is the same as the production or source database instance name.
 - ▶ The mount points that are used to mount the clone volumes have the same name as the mount points that are used to mount volumes of the production database.
1. By default when the database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

```
[DBDir]/[InstanceName]/NODE000n
```

where,

- ▶ DBDir identifies the name assigned to the directory/device the database is created on.
- ▶ InstanceName identifies the name assigned to the DB2 instance the database belongs to.

For example, if the DB2 instance name is db2inst1 and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory.

```
/mnt/dbdata/db2inst1/NODE0000
```

For a database server the DB2 instance name has to be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. In order to access the clone database from a different instance name, you need to change the default tablespace container's path name by executing the following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n  
[DBDir]/[NewInstanceName]/NODE000n
```

where,

- ▶ DBDir identifies the name assigned to the directory/device the database is created on.
- ▶ OldInstanceName identifies the name assigned to the DB2 instance the production database belongs to.
- ▶ NewInstanceName identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

```
mv /mnt/dbdata_c1/db2inst1 /mnt/dbdata_c1/db2instc
```

2. Now you need to change the database name and tablespace container's header information using the db2inidb command. To do this, create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look something to Example 12-5:

Example 12-5 configuration file

```
DB_NAME=mydb,mydbc1
DB_PATH=/mnt/dbdata,/mnt/dbdata_c1
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_c1/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_c1/*
```

In this configuration file, the source database name is mydb, and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to mydbc1. It has its data on /mnt/dbdata_c1 and logs on /mnt/dblogs_c1. The source database instance name is db2inst1, and the clone database instance name is db2instc.

Save the configuration file as /home/db2inst1/dbrelocate.cfg, and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing `db2start`.

The production database was online during the Snapshot creation process, you will need to use the `db2inidb` utility to initialize the clone database as a snapshot. You can do this by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

where, `DatabaseName` identifies the name of the clone database that is going to be used for the test environment.

For example, to initialize a clone database named `mydbcl`, you would execute the following command on the database server:

```
db2inidb mydbcl as snapshot
```

12.3.8 Verify the database

After performing the steps in the previous section, check the entire database for architectural correctness, by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

where, `DatabaseName` identifies the name of the clone database used for the test environment.

12.4 Cloning an offline database to a remote storage system

The FlexClone feature combined with SnapMirror (Figure 12-3 on page 251) makes it possible to clone a DB2 database on a remote IBM System Storage N series over the network or on another aggregate of the same IBM System Storage N series. This section describes the steps necessary to clone a database on the second storage system at a remote location.

Application Disaster Recovery

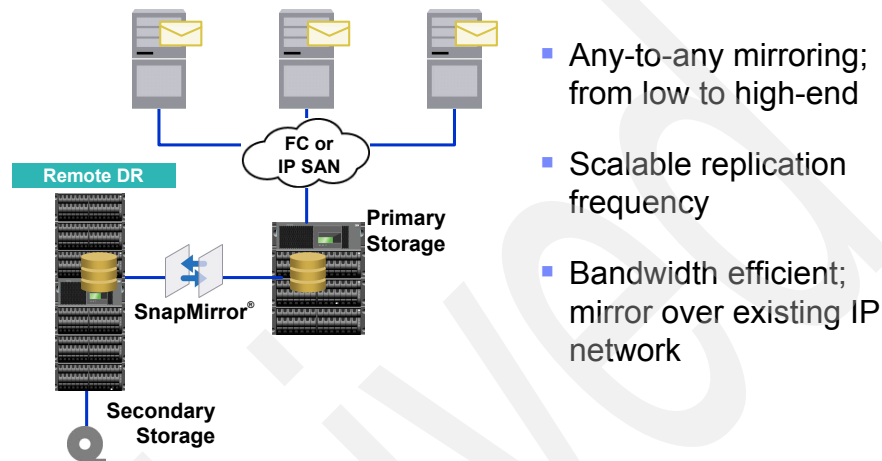


Figure 12-3 SnapMirror utilization

12.4.1 Configuring SnapMirror

To configure SnapMirror, complete the following steps on the SnapMirror source and the SnapMirror destination IBM System Storage N series:

1. Add licenses for SnapMirror and SnapMirror sync on the source and destination IBM System Storage N series by executing the following command:

```
license add [licenseCode]
```

where, licenseCode identifies the license key for the product or feature that is provided to you by the IBM sales representative.

For example, to enable a SnapMirror license key, you would execute the following command on the storage system:

```
License add 1234ABCDE
```

Verify the license through FilerView. See Figure 12-4 on page 252.

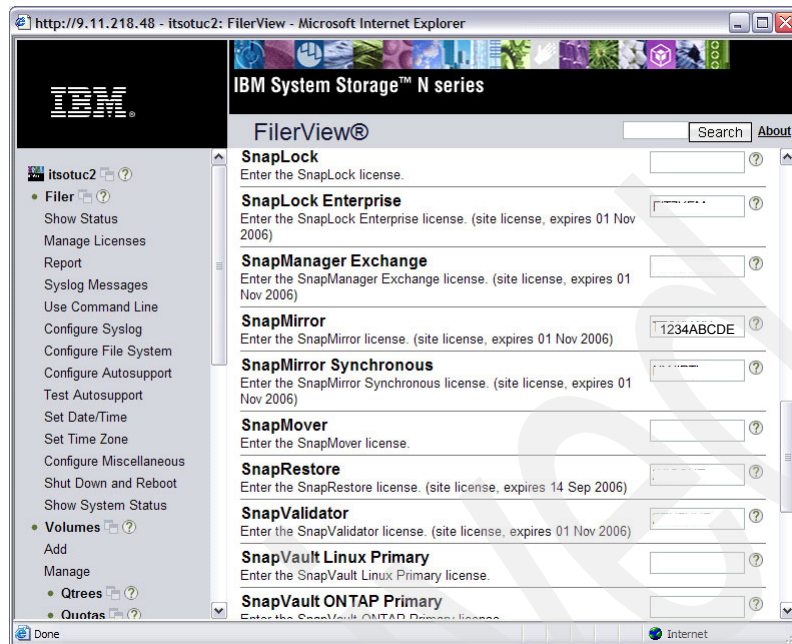


Figure 12-4 License manager

2. Enable the SnapMirror feature by executing the following command on the SnapMirror source and destination storage systems:

snapmirror on

or

options snapmirror.enable on

Verify options with the following command:

options snapmirror

The output will look similar Example 12-6.

Example 12-6 options snapmirror

```
itsotuc2> options snapmirror
snapmirror.access          host=192.168.2.2
snapmirror.checkip.enable  off
snapmirror.delayed_acks.enable on
snapmirror.enable          on
snapmirror.log.enable      on
```

3. On the SnapMirror source storage system, create a snapmirror.allow file that resides in the /vol/vol0/etc directory, and append an entry for the SnapMirror destination storage system. The entries in the snapmirror.allow file should look similar to the following:

[DestinationStorageSystemName]

where, DestinationStorageSystemName identifies the name assigned to the SnapMirror destination storage system.

For example, to append an entry for the storage system named dststore, you would add the following entry to the snapmirror.allow file on the SnapMirror source storage system:

dststore

4. Create a configuration file named snapmirror.conf that resides in the /vol/vol0/etc directory on the destination storage system. The configuration file identifies the source storage system, source volume, destination volume, and SnapMirror schedule. A snippet from a snapmirror.conf file looks similar to Example 12-7.

Example 12-7 snapmirror.conf

```
#-----  
# file name: snapmirror.conf  
# Description: This file resides on a destination storage system and  
# consists of SnapMirror configuration details. Entry format is as follows:  
# src_filer:[vol | qtreetpath] dest_filer:[vol | qtreetpath] argument schedule  
#-----  
srcstore:dbdata dststore:dbdata - 0 * * 1-5  
srcstore:dblogs dststore:dblogs - 0 * * 1-5  
# srcstore:dblog1 dststore:dblog1 - sync -- synchronous SnapMirror
```

If the word sync is specified in the snapmirror.conf file instead of a definitive schedule, that implies synchronous Snapmirror configuration for that particular volume.

12.4.2 Initializing SnapMirror

As noted earlier, the initialization process transfers data, including all Snapshot copies, from the source volume to the destination volume for the first time; thereafter, only changed blocks are transferred. To initialize the SnapMirror relationship, you need to restrict the destination volume by executing the following command on the destination storage system:

```
vol restrict [VolumeName]
```

where, VolumeName identifies the name assigned to the volume that is being used as the SnapMirror destination.

For example, to restrict a SnapMirror destination volume named dbdata, you would execute the following command on the destination storage system:

```
vol restrict dbdata
```

Verify the execution of the command with the following command:

```
vol status dbdata
```

Example 12-8 is the output from the command.

Example 12-8 vol status example

```
vol status dbdata
Volume State      Status      Options
dbdata restricted raid_dp, flex
                  Volume has clones: dbdata_cl
                  Containing aggregate: 'aggr0'
```

After restricting the destination FlexVol volumes, you need to initialize the SnapMirror relationship for each volume that is used for the database by executing the following command on the destination storage system:

```
snapmirror initialize -S
[SourceStorageSystem] : [VolumeName] [DestinationStorageSystem] : [Volume
Name]
```

where,

- ▶ SourceStorageSystem identifies the name assigned to the SnapMirror source storage system.
- ▶ VolumeName identifies the name assigned to the volume that is the SnapMirror source/destination.
- ▶ DestinationStorageSystem identifies the name assigned to the SnapMirror destination storage system.

For example, to initialize the SnapMirror relationship that is specified in the configuration file for the volume named dbdata, you would execute the following command on the destination storage system:

```
snapmirror initialize -S srcstore:dbdata dststore:dbdata
```

You can accomplish the same thing through FilerView, as shown in Figure 12-5.

The screenshot shows the IBM System Storage N series FilerView web interface. The left sidebar contains a navigation menu with options like 'Configure File System', 'Configure Autosupport', 'Test Autosupport', 'Set Date/Time', 'Set Time Zone', 'Configure Miscellaneous', 'Shut Down and Reboot', 'Show System Status', 'Volumes', 'Add', 'Manage', 'Quotas', 'Snapshots', 'Aggregates', 'Storage', 'DFM', 'SnapMirror', 'Report', 'Configure', 'Manage', 'Add', 'Log', 'Remote Access', 'Enable/Disable', and 'CIFS'. The main content area is titled 'FilerView' and contains configuration fields for 'Destination Filer' (itsotuc1), 'Destination Location' (dbdata), 'Source' (Filer: itsotuc2, Location: dbdata), 'Restart Mode' (Schedule Priority), 'Maximum Transfer Rate', and 'Repeat Mirror' (Every Minute). There are also options for 'Start Mirror on' (Always) and 'Use Custom Schedule'.

Figure 12-5 SnapMirror creation

The SnapMirror initialize command creates a Snapshot copy of the source volume and transfers data from the source to the destination volume. After baseline data transfer is finished, the destination volume data is available in read-only mode.

The progress of SnapMirror can be checked by executing the following command on the storage system:

```
snapmirror status
```

The output from this command should look similar to Example 12-9 on page 256 or Figure 12-6 on page 256.

snapmirror status

snapmirror is on.

Source	Destination	State	Lag	Status
srcstore:dbdata	dststore:dbdata	SnapMirrored	00:07:33	Idle
srcstore:dblogs	dststore:dblogs	Uninitialized	-	Transferring(116 MB done)

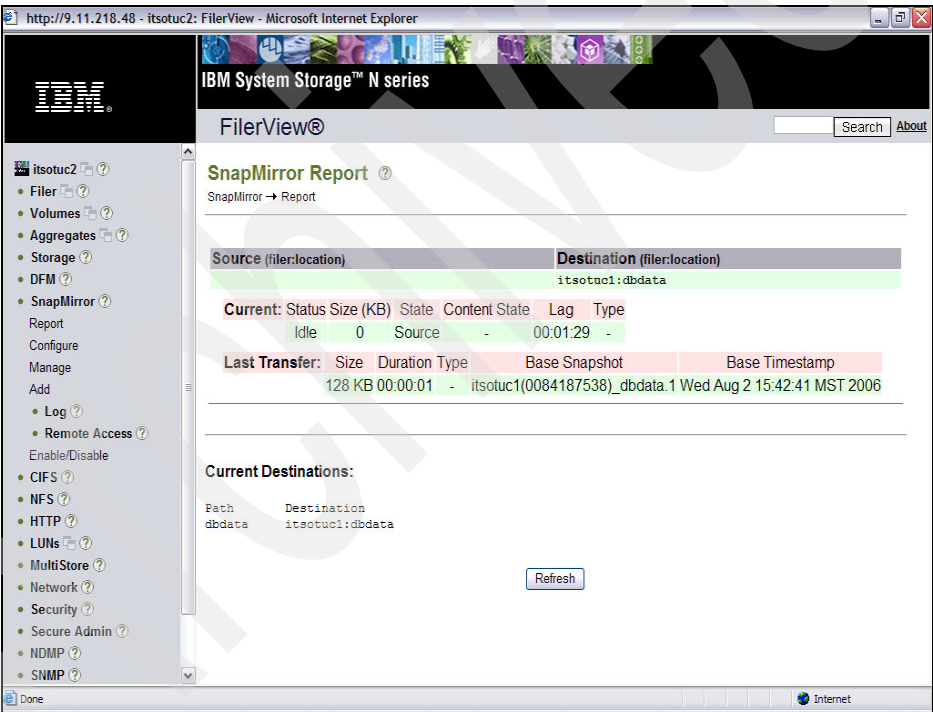


Figure 12-6 SnapMirror Status

12.4.3 Create clones of the FlexVol volumes

Create clones of the FlexVol volumes on the remote storage system. In the SnapMirror relationship, the destination volumes are read-only; therefore, to create a clone database on a remote IBM System Storage N series, you need to

create Snapshot copies of the FlexVol volumes on the SnapMirror source storage system and update the SnapMirror relationship manually. The SnapMirror update operation replicates data, including Snapshot copies, to the SnapMirror destination storage system, and the replicated Snapshot copy can be used to create the clone volume. Following are the detailed steps:

1. Bring the database offline by terminating all the application connections to the database by executing the following command on the database server:

```
db2 force applications all
```

2. Create a Snapshot copy for each volume that is used for the database by executing the following command on the SnapMirror source storage system:

```
snap create [VolumeName] [SnapName]
```

where,

- ▶ VolumeName identifies the name assigned to the FlexVol volume on the SnapMirror storage system that is used for the database.
- ▶ SnapName identifies the name assigned to the Snapshot copy of the FlexVol volume that is used as the source for the clone volume.

For example, to create a Snapshot copy named `dbdata_cl_snap.1` of a volume named `dbdata`, you would execute the following command on the SnapMirror source storage system:

```
snap create dbdata dbdata_cl_snap.1
```

You should name the Snapshot copy in a way that describes its intended use. Naming the Snapshot copy appropriately helps to avoid unintended deletion of it from the source storage system.

3. After Snapshot copies of the FlexVol volumes of the source database are created, you can connect to the database and start using it.
4. Replicate the Snapshot copy created in step 2 to the SnapMirror destination volume. To do this, you need to update the SnapMirror relationship manually by executing the following command on the destination storage system:

```
snapmirror <-S [SourceStorageSystem]:[VolumeName]> update  
[DestinationStorageSystem]:[VolumeName]
```

where,

- ▶ SourceStorageSystem identifies the name assigned to the SnapMirror source storage system.
- ▶ DestinationStorageSystem identifies the name assigned to the SnapMirror destination storage system.

- ▶ VolumeName identifies the name assigned to the source/destination volume in the SnapMirror relationship.

For example, to update SnapMirror for a volume named dbdata, you would execute the following command on the SnapMirror destination storage system named dststore:

```
snapmirror -S srcstore:dbdata update dststore:dbdata
```

When the update command is executed for asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including Snapshot copies.

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes visible on the destination volume immediately; therefore, manual update is not required.

Important: Snapshot copies are also created automatically for SnapMirror update purposes. We recommend that you not use these automatically created Snapshot copies to create a clone volume.

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the source storage system and replicated to the destination. A clone volume can be created from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

where,

- ▶ CloneVol identifies the name of the FlexClone volume that is being created.
- ▶ ParentVol identifies the name of the FlexVol volume that is the source for the clone volume.
- ▶ ParentSnap identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone volume named dbdata_cl from the Snapshot copy named dbdata_cl_snp.1 of the volume named dbdata, you would execute the following command on the destination storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.1
```

Important: The Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship will fail.

12.4.4 Creating NFS export entries for the cloned volumes.

To mount a clone volume to the database server, you need to create an export entry for it in the `/etc/exports` file that resides on the NetApp® FAS or the IBM System Storage N series. The export entry can be created by executing the following command on the IBM System Storage N series:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

where,

- ▶ `HostName` identifies the name assigned to the database server.
- ▶ `PathName` identifies the name assigned to the flexible volume.

For example, to create an export entry for a clone volume named `dbdata_cl` and allow root access from the database server named `hostdst` for it, you would execute the following command on the storage system:

```
exportfs -p rw=hostdst,root=hostdst /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is used for the clone database.

12.4.5 Mounting the clone volumes

To access the clone database, you need to mount it to a database server. First, you create a mount point for each clone volume, and append a mount entry to the `/etc/fstab` file. The mount entry should specify the mount options, and it should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs  
hard,rw,nointr,rsize=32768,wsiz=32768,bg,vers=3,tcp 0 0
```

where,

- ▶ `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- ▶ `FlexVolName` identifies the name assigned to the clone volume.
- ▶ `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named `dbdata_cl` that resides on the storage system named `srcstore`, you would append the following entry to the `/etc/fstab` file on the database server:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs  
hard,rw,nointr,rsize=32768,wsiz=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

where, MountPoint identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has mount entry specified in the /etc/fstab file, you would execute the following command on the second database server named hostdst:

```
mount /mnt/dbdata_cl
```

The database servers we used had Linux operating systems.

To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volumes that are mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

where,

- ▶ InstanceOwner identifies the name assigned to the user who owns the database instance.
- ▶ InstanceOwnerGroup identifies the name assigned to the user's group that owns the database instance.
- ▶ FileSystem identifies the name of the file system whose ownership is to be changed.

For example, to change ownership of the file system mounted on the mount point named /mnt/dbdata_cl, you would execute the following command on the database server named hostdst:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

12.4.6 Configuring the cloned database

The clone volumes created in 12.4.3, “Create clones of the FlexVol volumes” on page 256 and mount in the 12.4.5, “Mounting the clone volumes” on page 259 of this section are going to be used as the storage containers for the cloned database. You can skip parts (1) and (2) of this step if the following two conditions are true for you environment:

- ▶ The name of the DB2 instance used for the clone database is the same as the production or source database instance name.

- ▶ The mount points that are used to mount the clone volumes have the same name as the mount points that are used to mount volumes of the production database.
- 1. By default when the database is created, the tablespace containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

```
[DBDir]/[InstanceName]/NODE000n
```

where,

- ▶ DBDir identifies the name assigned to the directory/device the database is created on.
- ▶ InstanceName identifies the name assigned to the DB2 instance the database belongs to.

For example, if the DB2 instance name is db2inst1 and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory.

```
/mnt/dbdata/db2inst1/NODE0000
```

For a database server the DB2 instance name has to be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, change the default tablespace container's path name by executing the following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n
   [DBDir]/[NewInstanceName]/NODE000n
```

where,

- ▶ DBDir identifies the name assigned to the directory/device the database is created on.
- ▶ OldInstanceName identifies the name assigned to the DB2 instance the production database belongs to.
- ▶ NewInstanceName identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

```
mv /mnt/dbdata_c1/db2inst1 /mnt/dbdata_c1/db2instc
```

2. Now you need to change the database name and tablespace container's header information using the db2inidb command. To do this, create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to Example 12-10.

Example 12-10 Configuration file

```
DB_NAME=mydb,mydbc1
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata*/,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is mydb, and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to mydbc1. It has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is db2inst1, and the clone database instance name is db2instc.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline while the Snapshot copies were created, the clone database becomes available after executing db2start.

The production database was offline during the Snapshot creation process. Therefore, you can connect to the clone database and start using it.

4. Catalog the source database.

If the cloned database is accessed from the same DB2 instance as the production database on the production database server, then on execution the db2relocatedb command uncatalogs the source database. Therefore, you need to catalog the source database by executing the following command on the database server:

```
db2 "catalog database [DatabaseName] as [DatabaseAlias] on  
[FileSystem]"
```

where,

- ▶ DatabaseName identifies the name assigned to the database that is being cataloged.
- ▶ DatabaseAlias identifies the alias name assigned to the database that is being cataloged.
- ▶ FileSystem specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named mydb that resides on the file system named /mnt/dbdata, you would execute the following command on the database server:

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

12.4.7 Cataloging the source database

If the cloned database is accessed from the same DB2 instance as the production database on the production database server, then on execution the db2relocatedb command uncatalogs the source database. Therefore, you need to catalog the source database by executing the following command on the database server:

```
db2 "catalog database [DatabaseName] as [DatabaseAlias] on [FileSystem]"
```

where,

- ▶ DatabaseName identifies the name assigned to the database that is being cataloged.
- ▶ DatabaseAlias identifies the alias name assigned to the database that is being cataloged.
- ▶ FileSystem specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named mydb that resides on the file system named /mnt/dbdata, you would execute the following command on the database server:

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

12.4.8 Verifying the cloned database

After performing the steps in the previous section, check the entire database for architectural correctness by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

where, DatabaseName identifies the name of the clone database used for the test environment.

For example, to test the database named mydbc1 you would execute the following command on the database server:

```
db2dart mydbc1 /db
```

The db2dart utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the <\$HOME>/sqlib/db2dump/DART0000/ directory. The report has a summary at the end of the report. You can read the summary and obtain the errors, if any.

12.5 Cloning an online database to a remote storage system

In this section, you learn how to create a clone database on a remote storage system when the source database is online. The step-by-step cloning process is described in the following subsections.

12.5.1 Configuring and initialize SnapMirror

The steps necessary to configure and initialize a SnapMirror relationship are described in section 12.4.1, “Configuring SnapMirror” on page 251 and section 12.4.2, “Initializing SnapMirror” on page 254 respectively. After completing these steps, you should have working SnapMirror relationships for the volumes that are used for the database to be cloned.

12.5.2 Bring the source database into a consistent state—suspend writes

The source database is online. Therefore, in order to prevent partial page writes while database cloning is in progress, you must temporarily suspend the write operations to the database by executing the following command on the database server:

```
db2 set write suspend for database
```

The SET WRITE SUSPEND FOR DATABASE command causes the DB2 database manager to suspend all write operations to tablespace containers and log files that are associated with the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The cloning process is completed very

quickly, so the database does not need to stay in write suspend mode for more than a few seconds.

12.5.3 Creating Snapshot copies of the FlexVol volumes

Create a Snapshot copy of each FlexVol volume that is used for the source database by executing the following command on the source storage system:

```
snap create [VolName] [SnapName]
```

where,

- ▶ VolName identifies the name assigned to the FlexClone volume that is to be created.
- ▶ SnapName identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named dbdata_cl_snap01 for a FlexVol volume named dbdata, you would execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.1
```

We recommend that you develop a naming convention, and assign a meaningful name to the Snapshot copies that are created for cloning purposes.

Important: The Snapshot copies are created on the SnapMirror source storage system.

12.5.4 Resuming normal database operations—resume writes

After the Snapshot copies are created, you need to resume write operations to the database by executing the following command on the database server:

```
db2 set write resume for database
```

12.5.5 Updating the SnapMirror destination

Now update the SnapMirror destination volumes manually by executing the following command on the destination storage system:

```
snapmirror < -S [SourceStorageSystem]:[VolumeName]> update  
[DestinationStorageSystem]:[VolumeName]
```

where,

- ▶ SourceStorageSystem identifies the name assigned to the SnapMirror source storage system.

- ▶ VolumeName identifies the name assigned to the volume that is being used as the SnapMirror destination.
- ▶ DestinationStorageSystem identifies the name assigned to the SnapMirror destination storage system.

For example, to update SnapMirror for a volume named dbdata, you would execute the following command on the SnapMirror destination storage system named dststore:

```
snapmirror -S srcstore:dbdata update dststore:dbdata
```

Update the SnapMirror relationship for each volume that is used for the database.

When the update command is executed for an asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including Snapshot copies

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes available on the destination volume immediately; therefore, manual update is not required.

Important: Snapshot copies are also created automatically for SnapMirror update purposes. We recommend that you not use these automatically created Snapshot copies to create a clone volume.

12.5.6 Creating clone volumes using Snapshot copies

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the source storage system and replicated to the destination. A clone volume can be created from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

where,

- ▶ CloneVol identifies the name of the FlexClone volume that is being created.
- ▶ ParentVol identifies the name of the FlexVol volume that is the source for the clone volume.
- ▶ ParentSnap identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone volume named `dbdata_cl` from the Snapshot copy named `dbdata_cl_snp.1` of the volume named `dbdata`, you would execute the following command on the destination storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.1
```

Important: The Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship will fail.

12.5.7 Creating NFS export entries for the cloned volumes

To mount a clone volume to the database server, you need to create an export entry for it in the `/etc/exports` file that resides on the NetApp FAS or IBM System Storage N series. The export entry can be created by executing the following command on the IBM System Storage N series:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

where,

- ▶ `HostName` identifies the name assigned to the database server.
- ▶ `PathName` identifies the name assigned to the flexible volume.

For example, to create an export entry for a clone volume named `dbdata_cl` and allow root access from the database server named `hostdst` for it, you would execute the following command on the storage system:

```
exportfs -p rw=hostdst,root=hostdst /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is used for the clone database.

12.5.8 Mounting the cloned volumes

To access the clone database, mount the clone volumes to a database server. First, you create a mount point for each clone volume, and append a mount entry to the `/etc/fstab` file. The mount entry should specify the mount options, and it should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs  
hard,rw,nointr,rsz=32768,wsz=32768,bg,vers=3,tcp 0 0
```

where,

- ▶ `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- ▶ `FlexVolName` identifies the name assigned to the clone volume.

- MountPoint identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named `dbdata_cl` that resides on a storage system named `srcstore`, you would append the following entry to the `/etc/fstab` file on the database server:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsz=32768,wsz=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

where, MountPoint identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has a mount entry specified in the `/etc/fstab` file, you would execute the following command on the second database server named `hostdst`:

```
mount /mnt/dbdata_cl
```

The database servers we used had a Linux operating system.

To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

where,

- InstanceOwner identifies the name assigned to the user who owns the database instance.
- InstanceOwnerGroup identifies the name assigned to the user's group that owns the database instance.
- FileSystem identifies the name of the file system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, you would execute the following command on the database server named `hostdst`:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```


12.5.9 Configuring the cloned database

The clone volumes created in section 12.5.6, “Creating clone volumes using Snapshot copies” on page 266 and mount in section 12.5.8, “Mounting the cloned volumes” on page 267 of this section are going to be used as the storage containers for the cloned database. You can skip steps (1) and (2) of this procedure if the following two conditions are true for your environment:

1. The name of the DB2 instance used for the clone database is the same as the production or source database instance name.
2. The mount points that are used to mount the clone volumes have the same name as the mount points that are used to mount volumes of the production database.
1. By default when the database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

`[DBDir]/[InstanceName]/NODE000n`

where,

- ▶ `DBDir` identifies the name assigned to the directory/device the database is created on.
- ▶ `InstanceName` identifies the name assigned to the DB2 instance the database belongs to.

For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace containers will reside in the following directory.

`/mnt/dbdata/db2inst1/NODE0000`

For a database server the DB2 instance name has to be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. In order to access the clone database from a different instance name, change the default tablespace container's path name by executing following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n  
[DBDir]/[NewInstanceName]/NODE000n
```

where,

- ▶ `DBDir` identifies the name assigned to the directory/device the database is created on.
- ▶ `OldInstanceName` identifies the name assigned to the DB2 instance the production database belongs to.

- NewInstanceName identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

```
mv /mnt/dbdata_c1/db2inst1 /mnt/dbdata_c1/db2instc
```

2. Change the database name and tablespace containers header information using the db2inidb command. To do this, you need to create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to Example 12-11:

Example 12-11 Configuration file

```
DB_NAME=mydb,mydbc1
DB_PATH=/mnt/dbdata,/mnt/dbdata_c1
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_c1/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_c1/*
```

In this configuration file, the source database name is mydb and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to mydbc1. It has its data on /mnt/dbdata_c1 and logs on /mnt/dblogs_c1. The source database instance name is db2inst1, and the clone database instance name is db2instc.

Save the configuration file as /home/db2inst1/dbrelocate.cfg, and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing db2start.

The production database was online when the Snapshot copies were created. Therefore, you need to use the db2inidb utility to initialize the clone database as a snapshot. You can do this by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

where, DatabaseName identifies the name of the clone database that is going to be used for the test environment.

For example, to initialize a clone database named mydbc1, you would execute the following commands on the database server:

```
db2inidb mydbc1 as snapshot
```

12.5.10 Verifying the cloned database

After performing the steps in the previous section, check the entire database for architectural correctness. You can do so by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

where, DatabaseName identifies the name of the clone database used for the test environment.

For example, to test the database named mydbc1 you would execute the following command on the database server:

```
db2dart mydbc1 /db
```

The db2dart utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the <\$HOME>/sqlib/db2dump/DART0000/ directory. The report has a summary at the end of the report. You can read the summary and obtain the errors, if any.



DB2 and N series performance considerations

This chapter discusses the performance considerations of DB2 and IBM System Storage N series.

13.1 General performance guidelines

DB2 UDB Enterprise Server Edition, Version 8, can exist in environments ranging from simple stand-alone systems to complex combinations of servers and clients running on a wide variety of platforms. Regardless of which environment is used, one of the most important aspects, from a user's point-of-view, is how well (or how poorly) the database applications work on a day-to-day basis perform. But just what is *performance*? How do you know if it is bad, and if it is bad, what can be done to improve it? In its simplest terms, performance is the way a computer system behaves while it is executing a given task. Performance is typically measured in terms of system response time, throughput, and availability and each of these metrics can be affected by several factors, including the type of hardware being used, the system (and, in our case, the database) configuration used, the type of and number of users working concurrently, and the workload performed by each user's application. After you decide a system is performing poorly, there are usually several things you can do to tune it. However, because a wide variety of tuning options are available, tuning should always be done in an organized, concise manner with a specific goal in mind. And to be successful, the goal should be realistic, quantitative, and measurable; otherwise, performance tuning becomes a hit-or-miss exercise. The following performance-tuning guidelines—that apply to both DB2 and the IBM System Storage N series—should help you get started:

1. Check for known hardware and software problems.

Some performance problems can be corrected simply by applying service packs to your software, by upgrading your hardware, or both. Why spend the time and effort monitoring and tuning other parts of your system when a simple service may resolve the problem? With that said, make sure you understand the problem before you upgrade hardware. If you decide to add another network interface card (NIC) only to discover the system needs more memory, you just made a costly mistake that did nothing to improve performance.

2. Consider the entire system.

Usually, you cannot tune one aspect of a system without affecting at least one other part of that system. So before you make changes, consider how those changes will affect the system as a whole.

3. Measure and reconfigure by levels.

Never tune more than one level of your system at a time—even if you are sure all the changes you plan to make will be beneficial. You will have no way to evaluate how much each change contributed to any performance improvements seen. On the other hand, if you are wrong and performance goes down instead of up, you have no way of knowing which change had the negative impact. In a database server environment, the following list of levels within a system can be used as a guide:

- Hardware of both the database server and the IBM System Storage N series
- Operating system
- Communications software
- Database
- SQL statements
- Applications

4. Change one thing at a time.

For the same reasons that you should tune only one system level at a time, you should change only one thing at a time as you tune each system level.

5. Put tracking and fallback procedures in place before you start.

Unfortunately, performance tuning is not an exact science and some changes you make will result in a negative, rather than a positive, impact on performance. When such a situation occurs, you can avoid spending time trying to get the system back to an earlier state if you have a way to back out of every change made. Likewise, if you are forced to back out of several changes, be prepared to reapply every change made, if necessary.

Tip: When using IBM System Storage N series to accommodate your DB2 storage, an excellent way to back out of changes is to take a Snapshot copy of the FlexVol volume that the database is stored on before each change is made, and then restore the volume with SnapRestore if the change turns out to be detrimental to performance. You can also make a FlexClone of the database volumes.

6. Do not tune just for the sake of tuning.

Tuning should be performed to resolve an identified problem. If you tune resources that are not directly related to the primary cause of the problem you are trying to solve, there will be little or no noticeable effect until the primary problem itself is resolved. And in some cases, such actions can actually make subsequent tuning work more difficult.

7. Remember the law of diminishing returns.

Keep in mind that the greatest performance tuning gains usually come from your initial efforts. Subsequent tuning usually results in progressively smaller gains and requires progressively greater amounts of effort.

As you can see, tuning a system can be quite involved and volumes could be written (and often have been) on tuning each system level. Because the subject is quite complex, no single chapter can really teach you everything you need to know in order to tune a system for optimum performance. With that in mind, this chapter aims to provide you with basic information that you can use to tune the database level of a system when DB2 UDB is the database being used on an IBM System Storage N series.

13.2 IBM System Storage N series and DB2 tuning

The IBM System Storage N series server is highly efficient at automatically managing the data residing on it using the Write Anywhere File Layout (WAFL). The DB2 database adds optimization and file-handling characteristics that can be adjusted to exploit the underlying file system characteristics. By understanding the storage server's WAFL characteristics, more intelligent DB2 container parameters can be selected.

WAFL writes a group of 4K blocks to a data disk at a time. This increment is called a *chunk*. After a data disk has had the requisite number of blocks written to the chunk, WAFL goes to the next data disk in the RAID group to write the next group of blocks (chunk) or fraction thereof. The storage server will read data from the disks to the read-ahead buffer in thirty-two 4k block chunks. Additionally, the storage server read-ahead buffer size is up to 288k, so many storage server disks can be used to maximize throughput. WAFL also looks to write to the least-used disk in the RAID group. This causes a leveling of data among the disks in the RAID group over time. The RAID4 implementation that WAFL uses provides for enhanced availability and recoverability.

Armed with these basic facts, some reasonable numbers for tablespace prefetch sizes can be estimated. Table 13-1 on page 276 calculates reasonable prefetch sizes for a container, based on the DB2 tablespace page size and the number of data disks in the respective FlexVol volume RAID group. Optimally, we want to prefetch across an entire RAID stripe at a

time. With that in mind, the prefetch number should be equal to or a multiple of storage server chunks multiplied by the number of data disks. While a configuration based on this table may be valid when the database is first loaded, over time data fragmentation will occur. It is still a good idea to monitor query performance and perform REORGs when necessary.

Note: Although the table contains entries for 27 data disks for a RAID group, there is a diminishing return for performance if the number of data disks in a group is too large. Conversely, too few data disks in a group does not fill the storage server read-ahead buffer efficiently, and performance may be diminished. The default RAID group size is a good starting number for calculating the number of disks. *Your mileage may vary!* Use this table as a tool to help you calculate your initial configuration, not as a replacement for your normal performance testing and monitoring.

Table 13-1 Recommended prefetch size based on number of disks in RAID group and DB2 page size

Number of disks in RAID group	DB2 prefetch size with 4k pages	DB2 prefetch size with 8k pages	DB2 prefetch size with 16k pages	DB2 prefetch size with 32k pages
1	32	16	8	4
2	64	32	16	8
3	96	48	24	12
4	128	64	32	16
5	160	80	40	20
6	192	96	48	24
7	224	112	56	28
8	256	128	64	32
9	288	144	72	36
10	320	160	80	40
11	352	176	88	44
12	384	192	96	48
13	416	208	104	52
14	448	224	112	56
15	480	240	120	60
16	512	256	128	64
17	544	272	136	68
18	576	288	144	72
19	608	304	152	76
20	640	320	160	80
21	672	336	168	84
22	704	352	176	88
23	736	368	184	92

Number of disks in RAID group	DB2 prefetch size with 4k pages	DB2 prefetch size with 8k pages	DB2 prefetch size with 16k pages	DB2 prefetch size with 32k pages
24	768	384	192	96
25	800	400	200	100
26	832	416	208	104
27	864	432	216	108

For the DB2 extent size, we generally want one extent per storage server chunk. To maintain one DB2 extent per storage server chunk, use Table 13-2 to calculate the number of pages per extent.

Table 13-2 Recommended extent size based DB2 page size

DB2 page size	Extent size (in pages)
4k	32
8k	16
16k	8
32k	4

13.3 db2_parallel_io

When reading data from, or writing data to tablespace containers, DB2 may use parallel I/O if the number of containers in the database is greater than one. However, there are situations when it is beneficial to have parallel I/O enabled for single container table spaces. For example, if the container is created on a RAID device that is composed of more than one physical disk, performance may be improved if read and write calls are issued in parallel to all disks. This parallel I/O capability should be augmented with multiple physical paths to N series and further enhanced by the FlexVol and aggregate layout of N series.

To force DB2 UDB to use parallel I/O for a tablespace that only has one container, use the DB2_PARALLEL_IO registry variable. This variable can be set to asterisk (*), meaning every tablespace is to use parallel I/O. Alternately it can be set to a list of tablespace IDs that are separated by commas. For example, this command turns parallel I/O on for all table spaces:

```
db2set DB2_PARALLEL_IO=*
```

However, the following command only turns parallel I/O on for table spaces 1, 2, 4, and 8:

```
db2set DB2_PARALLEL_IO=1,2,4,8
```

The DB2_PARALLEL_IO registry variable also affects tablespaces with more than one container defined. If this registry variable is not set, the amount of I/O parallelism used is equal to the number of containers in the tablespace. However, if this registry variable is set, the I/O parallelism used is equal to the result of (prefetch size / extent size). For example, if a tablespace has two containers and the prefetch size is four times the extent size, and if the DB2_PARALLEL_IO registry variable is not set, a prefetch request for this tablespace is broken into two requests (each request will be for two extents). Provided that the prefetchers are available to do work, two prefetchers can be working on these requests in parallel. In the case where the DB2_PARALLEL_IO registry variable is set, a prefetch request for this

tablespace is broken into four requests (one extent per request) with a possibility of four prefetchers servicing the requests in parallel.

In this example, if each of the two containers had a single disk dedicated to it, setting the DB2_PARALLEL_IO registry variable might result in contention on those disks since two prefetchers are accessing each of the two disks at once. However, if each of the two containers was striped across multiple disks (a RAID disk), setting the DB2_PARALLEL_IO registry variable would potentially allow access to four different disks at the same time.

13.4 A word about db2empfa

Since SMS tablespaces are managed by the file system, rather than the DB2 Database Manager, a size for each container used does not have to be specified. That is because the system's file system is responsible for allocating additional storage space as it is needed. In the case of N series, this allocation is facilitated online by FlexVol. By default, SMS tablespaces are expanded one single page at a time. However, in certain work loads (for example, when doing a bulk insert) it might be desirable to have storage space allocated in extents rather than pages.

To force DB2 UDB to expand SMS tablespaces one extent at a time, rather than one page at a time, use the DB2EMPFA utility. The db2empfa tool is located in the bin subdirectory of the sqllib directory in which the DB2 UDB product is installed. Running it causes the multipage_alloc database configuration parameter (which is a read-only configuration parameter) to be set to YES.

13.5 Quick-start tips for general DB2 performance tuning

When you start a new instance of DB2, consider the following suggestions for a basic configuration:

- ▶ Use the Configuration Advisor in the Control Center to get advice about reasonable beginning defaults for your system. The defaults shipped with DB2 should be tuned for your unique hardware environment.

Gather information about the hardware at your site so you can answer the wizard questions. You can apply the suggested configuration parameter settings immediately, or let the wizard create a script based on your answers and run the script later.

This script also provides a list of the most commonly tuned parameters for later reference.
- ▶ Use other wizards in the Control Center and Client Configuration Assistant for performance-related administration tasks. These tasks are usually those in which you can achieve significant performance improvements by expending a little time and effort.

Other wizards can help you improve performance of individual tables and general data access. These wizards include the Create Database, Create Table, Index, and Configure Multisite Update wizards. The Health Center provides a set of monitoring and tuning tools.
- ▶ Use the Design Advisor tool from the Control Center or use the **db2adv** command to find out what indexes, materialized query tables, multi-dimensional clustering tables, and database partitions will improve query performance.
- ▶ Use the ACTIVATE DATABASE command to start databases. In a partitioned database, this command activates the database on all partitions and avoids the startup time required to initialize the database when the first application connects.

Note: If you use the `ACTIVATE DATABASE` command, you must shut down the database with the `DEACTIVATE DATABASE` command. The last application that disconnects from the database does not shut it down.

- Consult the summary tables that list and briefly describe each configuration parameter available for the database manager and each database.

These summary tables contain a column that indicates whether tuning the parameter results in high, medium, low, or no performance, changes, either for better or for worse. Use this table to find the parameters that you might tune for the largest performance improvements.

Archived

DB2 and AIX diagnostics and performance monitoring

In Chapter 3, “Introduction to IBM DB2 Universal Database” on page 65, we described the major components that make up a large database system that takes advantage of network attached storage. At times, it may become necessary to analyze the performance of such a database system in order to locate and resolve one or more problem areas that are having an adverse affect.

Fortunately, there are numerous tools at our disposal for monitoring each individual component of the system. In this chapter, we discuss some of these tools and show you how they can be used.

14.1 The DB2 Database System Monitor

A powerful tool provided with DB2 Universal Database, known as the *Database System Monitor*, can acquire information about the current state of a database system or about the state of a database system over a specified period of time. After collected, this information can be used to do the following:

- ▶ Monitor database activity.
- ▶ Assist in problem determination.
- ▶ Analyze database system performance.
- ▶ Aid in configuring and tuning the database system.

Although the Database System Monitor is often referred to as a single monitor, in reality it consists of several individual monitors that have distinct but related purposes. One of these individual monitors is known as the *snapshot monitor* (not to be confused with IBM System Storage N series Snapshots). The rest are known as *event monitors*. Both types of monitors can be controlled using graphical user interfaces provided with the Control Center, administrative application programming interface (API) functions, or DB2 commands.

14.1.1 The snapshot monitor

The *snapshot monitor* is designed to provide information about the state of a DB2 UDB instance, the data the instance controls, and to call attention to situations that appear to be abnormal. This information is provided in the form of a series of snapshots, each of which represents what the system looks like at a specific point-in-time.

The information collected by the snapshot monitor is maintained as a count value, a high water mark, or a time stamp value that identifies the last time a specific activity was performed. Snapshot monitor information can be collected for the following items:

- ▶ The DB2 Database Manager
- ▶ Databases (local, remote, or DCS)
- ▶ Applications (local, remote, or DCS)
- ▶ The Fast Communications Manager (for communications between partitions in a partitioned environment)
- ▶ Buffer pools
- ▶ Tablespaces
- ▶ Tables
- ▶ Locks
- ▶ Dynamic SQL statements

Snapshot monitor switches

In some cases, obtaining data collected by the snapshot monitor requires additional processing overhead. For example, in order to calculate the execution time of a SQL statement, the DB2 Database Manager must make a call to the operating system to obtain timestamps before and after the statement is executed.

Such system calls are generally expensive. Because of this, the snapshot monitor provides you with a great deal of flexibility in choosing what information you want to collect when a snapshot is taken. The type and amount of information returned (and the amount of overhead required) in a snapshot is determined by the use of special *snapshot monitor switches*. These switches specify what data is to be collected in a snapshot and are turned on or off as needed.

Table 14-1 shows the snapshot monitor switches available, along with a description of the type of information that is collected when each one is set.

Table 14-1 Snapshot monitor switches.

Snapshot monitor switch	DBM CFG parameter	Group	Information provided
SORT	DFT_MON_SORT	Sorts	Number of heaps used, overflows, sorts performance
LOCK	DFT_MON_LOCK	Locks	Number of locks held, number of deadlocks
TABLE	DFT_MON_TABLE	Tables	Measure activity (rows read, rows written)
BUFFERPOOL	DFT_MON_BUFPOOL	Buffer pools	Number of reads and writes, time taken
UOW	DFT_MON_UOW	Unit of work	Start times, end times, completion status
STATEMENT	DFT_MON_STMT	SQL statements	Start time, stop time, statement identification

As you can see from the information provided in Table 14-1, each snapshot monitor switch available has a corresponding database manager configuration (DBM CFG) parameter value in the DB2 Database Manager configuration file. By setting a snapshot monitor switch using a DB2 Database Manager configuration parameter, snapshot monitor information can be collected at the instance level as opposed to the application level. Snapshot monitor switches are set at the application level using the **UPDATE MONITOR SWITCHES** command. Snapshot monitor switches are set at the instance level using the **UPDATE DBM CFG** command.

Before setting an application level snapshot monitor switch (for example, by issuing the **UPDATE MONITOR SWITCHES** command from the Command Line Processor), an instance or database connection is required. After the instance or database connection is made and snapshot monitor switches set, all data collected for the selected switches is made available to the application/user until the connection is terminated. The data collected is specific to the switch settings set by the application. Other applications will have their own set of switch settings that they can turn on or off independently. In order to make snapshot information available and consistent for all connections for a particular instance, the default instance level monitor switches should be used by setting the appropriate DB2 database manager configuration (DBM CFG) parameters.

Note: Typically, when you change the value of a DB2 database manager configuration parameter, you need to restart the corresponding DB2 instance before those changes will take effect. However, changes made to parameters that correspond to the snapshot monitor switches are effective immediately and do not require a restart of the DB2 instance. Instead, all you need to do is terminate and re-establish any active connections for the changes to take effect.

Examining the current state of the snapshot monitor switches

Before setting any particular snapshot monitor switches, it is a good idea to examine the current state of all the snapshot monitor switches available. The easiest way to examine the current state of the snapshot monitor switches available is by executing the **get monitor switches** command that is issued from the db2 prompt. There is also a DB2 API called **db2MonitorSwitches()**, which can be issued from a client application. For more information about that, see the DB2 UDB Information Center.

Example 14-1 illustrates the output from the **get monitor switches** command (the time stamp values shown correspond to the date and time a particular snapshot monitor switch was reset or turned on).

Example 14-1 The get monitor switches output

```
$ db2 get monitor switches
```

Monitor Recording Switches

Switch list for db partition number 0

Buffer Pool Activity Information	(BUFFERPOOL)	= OFF
Lock Information	(LOCK)	= OFF
Sorting Information	(SORT)	= OFF
SQL Statement Information	(STATEMENT)	= OFF
Table Activity Information	(TABLE)	= OFF
Take Timestamp Information	(TIMESTAMP)	= ON 06/19/2006 16:53:38.462888
Unit of Work Information	(UOW)	= OFF

```
$
```

The easiest way to examine the current state of the DB2 instance level snapshot monitor switches available is by executing the **get dbm monitor switches** command. Example 14-2 illustrates the output from this command. Again, the time stamp values shown correspond to the date and time a particular snapshot monitor switch was reset or turned on.

Example 14-2 The get dbm monitor switches output

```
$ db2 get dbm monitor switches
```

DBM System Monitor Information Collected

Switch list for db partition number 0

Buffer Pool Activity Information	(BUFFERPOOL)	= OFF
Lock Information	(LOCK)	= OFF
Sorting Information	(SORT)	= OFF
SQL Statement Information	(STATEMENT)	= OFF
Table Activity Information	(TABLE)	= OFF
Take Timestamp Information	(TIMESTAMP)	= ON 06/19/2006 16:53:38.462888
Unit of Work Information	(UOW)	= OFF

```
$
```

Capturing snapshot monitor information

After a snapshot monitor switch is turned on, the snapshot monitor collects appropriate monitor data until the switch is turned off. To capture and view this data at a specific point-in-time, take a snapshot. Snapshots can be taken by executing the **get snapshot** command from the Command Line Processor (CLP). You can also embed the **db2GetSnapshot()** API in a client application.

There are numerous options for the **get snapshot** command to control what snapshot information you want to see. Table 14-2 shows some of the more useful commands to use in an NAS or SAN environment.

Table 14-2 Some useful CLP **get snapshot** commands

Monitor level	CLP command	Information returned
Database	get snapshot for database on <i>db_name</i>	Database level information and counters for a database. Information is returned only if there is at least one application connected to the database.
Database	get snapshot for all databases	Database level information and counters for each active database on the partition. Information is only returned if there is at least one application connected to the database.
Application	get snapshot for applications on <i>db_name</i>	Application level information for each application connected to database <i>db_name</i> . This includes cumulative counters, status information, and most recent SQL executed (if statement switch is set).
Application	get snapshot for all applications	Application level information for each active application on the partition. This includes cumulative counters, status information, and most recent SQL executed (if statement switch is set).
Table	get snapshot for tables on <i>db_name</i>	Table activity information at the database and application level for each application connected to database <i>db_name</i> . Table activity information at the table level for each table that was accessed by an application connected to the database. Requires the table switch.
tablespace	get snapshot for tablespaces on <i>db_name</i>	Information about tablespace activity for database <i>db_name</i> . Requires buffer pool switch. Also includes information about containers, quiescers, and ranges.

Monitor level	CLP command	Information returned
Buffer pool	get snapshot for bufferpools on <i>db_name</i>	Buffer pool activity counters for database <i>db_name</i> . Requires buffer pool switch.
Buffer pool	get snapshot for all bufferpools	Buffer pool activity counters for all active buffer pools on the partition. Requires the buffer pool switch.

Example 14-3 shows an example of a tablespace snapshot taken using the **get snapshot for tablespaces on *db_name*** command. The full output is very long, so only the output for the first tablespace, SYSCATSPACE, is displayed. From this output, you can see the respective containers for this tablespace as well as the number of usable pages in the container. Further down, you can see the number of disk read and write operations performed for this tablespace.

If you encounter any problems between DB2 and N series, a good starting point is analysis of the Direct reads and writes, Direct read and write requests, and Direct read and write elapsed times fields in the output. These fields pertain to the direct reading from and writing to the IBM System Storage N series. For example, if you encounter poor query performance during peak hours, you may want to take several snapshots during peak hours and during off hours to compare. If you notice that the ratio between direct reads to direct read requests during peak hours is significantly different from off hours, that is an indication that further investigation may be needed on the storage system.

Reminder: To use the **get snapshot for tablespaces on *db_name*** command, remember to turn on the buffer pool switch.

*Example 14-3 Sample output of the get snapshot for tablespaces on *db_name* command*

```
$ db2 get snapshot for tablespaces on sample
Tablespace Snapshot

First database connect timestamp      = 06/20/2006 16:57:14.813509
Last reset timestamp                  =
Snapshot timestamp                    = 06/20/2006 16:57:43.941947
Database name                         = SAMPLE
Database path                         = /mnt2/db2data/db2inst2/NODE0000/SQL
00001/
Input database alias                  = SAMPLE
Number of accessed tablespaces        = 5

Tablespace name                       = SYSCATSPACE
Tablespace ID                         = 0
Tablespace Type                       = System managed space
Tablespace Content Type               = Any data
Tablespace Page size (bytes)          = 4096
Tablespace Extent size (pages)        = 32
Automatic Prefetch size enabled       = Yes
Buffer pool ID currently in use       = 1
Buffer pool ID next startup           = 1
Using automatic storage                = No
File system caching                    = Yes
```

Tablespace State	= 0x'00000000'
Detailed explanation:	
Normal	
Tablespace Prefetch size (pages)	= 192
Total number of pages	= 5802
Number of usable pages	= 5802
Number of used pages	= 5802
Minimum Recovery Time	=
Number of quiescers	= 0
Number of containers	= 1
Container Name	= /mnt2/db2data/db2inst2/NODE0000/SQL
00001/SQLT0000.0	
Container ID	= 0
Container Type	= Path
Total Pages in Container	= 5802
Usable Pages in Container	= 5802
Stripe Set	= 0
Container is accessible	= Yes
Buffer pool data logical reads	= 45
Buffer pool data physical reads	= 14
Buffer pool temporary data logical reads	= 0
Buffer pool temporary data physical reads	= 0
Asynchronous pool data page reads	= 0
Buffer pool data writes	= 0
Asynchronous pool data page writes	= 0
Buffer pool index logical reads	= 74
Buffer pool index physical reads	= 44
Buffer pool temporary index logical reads	= 0
Buffer pool temporary index physical reads	= 0
Asynchronous pool index page reads	= 0
Buffer pool index writes	= 0
Asynchronous pool index page writes	= 0
Total buffer pool read time (millisec)	= 19
Total buffer pool write time (millisec)	= 0
Total elapsed asynchronous read time	= 0
Total elapsed asynchronous write time	= 0
Asynchronous data read requests	= 0
Asynchronous index read requests	= 0
No victim buffers available	= 0
Direct reads	= 16
Direct writes	= 0
Direct read requests	= 4
Direct write requests	= 0
Direct reads elapsed time (ms)	= 0
Direct write elapsed time (ms)	= 0
Number of files closed	= 0
Data pages copied to extended storage	= 0
Index pages copied to extended storage	= 0
Data pages copied from extended storage	= 0
Index pages copied from extended storage	= 0

<... portion removed ...>

Example 14-4 shows a snapshot output of the **get snapshot for tables on db_name** command. This command returns information about table activity, such as number of rows read and written, which is useful in determining which tables get read from or written to more often. From the sample output in Example 14-4, you can see that the EMPLOYEE table had 730429 rows read.

Reminder: To use the **get snapshot for tables on db_name** command, remember to turn on the table switch.

Example 14-4 Sample output of the get snapshot for tables on db_name command

```
$ db2 get snapshot for tables on sample
```

Table Snapshot

```
First database connect timestamp    = 06/20/2006 16:57:14.813509
Last reset timestamp               =
Snapshot timestamp                 = 06/20/2006 17:11:26.036703
Database name                     = SAMPLE
Database path                     = /mnt2/db2data/db2inst2/NODE0000/SQL00001/
Input database alias              = SAMPLE
Number of accessed tables          = 1
```

Table List

```
Table Schema    = DB2INST1
Table Name      = EMPLOYEE
Table Type      = User
Data Object Pages = 184227
Rows Read       = 730429
Rows Written    = 0
Overflows       = 0
Page Reorgs     = 0
```

```
$
```

Resetting snapshot monitor counters

As you can see from the examples provided, the output produced by a snapshot contains—among other things—cumulative information about how often a particular activity was performed within a particular time frame. Such cumulative information is collected and stored in a wide variety of counters whose current values are retrieved each time a snapshot is taken. So when does the counting start exactly? Counting begins as follows:

- ▶ When a snapshot monitor switch is turned on, or when the DB2 Database Manager is restarted after one or more of its configuration parameters that correspond to a snapshot monitor switch have been turned on.
- ▶ Each time the counters are manually reset.

To use the snapshot monitor effectively, it is usually desirable to obtain snapshot information after a specific period of time elapses. To control the window of time that is monitored, the appropriate counters are usually set to zero when monitoring is to begin, and then a snapshot is taken after the desired period of time elapses. To reset all snapshot monitor counters to zero for a particular database, execute the **reset monitor for database db_name** command.

To reset all snapshot monitor switches for all databases within an instance to zero, use the **reset monitor all** command.

Resetting snapshot monitor switches to zero effectively restarts all counting, and future snapshots will contain new counter values.

14.1.2 Event monitors

Some database activities cannot be monitored easily with the snapshot monitor. For instance, snapshots cannot effectively monitor when a deadlock cycle occurs. If the deadlock detector “awakens” and discovers that a deadlock exists in the database locking system, it randomly selects, rolls back, and terminates one of the transactions involved in the deadlock cycle. Information about this series of events cannot be easily captured by the snapshot monitor because it is difficult, if not impossible, to anticipate when a deadlock is going to occur.

The event monitor, on the other hand, could be used to capture such information. Unlike the snapshot monitor, which is used to record the state of database activity at a specific point-in-time, an event monitor records database activity as soon as a specific event or transition occurs. And although event monitors return information that is very similar to the information returned by the snapshot monitor, the event itself is what triggers data collection, not the user or application.

Specifically, event monitors can capture and write system monitor data to a file or a named pipe whenever any of the following events occur:

- ▶ A transaction is terminated
- ▶ A SQL statement is executed
- ▶ A deadlock cycle is detected
- ▶ A connection to a database is established
- ▶ A connection to a database is terminated
- ▶ A database is activated
- ▶ A database is deactivated
- ▶ A SQL statement's subsection completes processing in a partitioned environment
- ▶ The **flush event monitor** statement is executed

Unlike the snapshot monitor, which resides in the background and is controlled by the settings of the snapshot monitor switches (at the application or instance level), event monitors are created using Data Definition Language (DDL) statements. Because of this, event monitors only gather information for the database in which they are defined. Thus, event monitors cannot be used to collect information at the instance level.

When an event monitor is created, the event types that will be monitored must be included as part of the event monitor's definition. An event monitor can monitor any of the following types of events:

- ▶ **DATABASE:** Records an event record when the database is deactivated.
- ▶ **TABLES:** Records an event record for each active table. An active table is a table that has changed since the first connection to the database was established. Records when the database is deactivated.
- ▶ **DEADLOCKS:** Records an event record for each deadlock event. Records when the deadlock is detected.
- ▶ **TABLESPACES:** Records an event record for each active tablespace when the database is deactivated.
- ▶ **BUFFERPOOLS:** Records an event record for each buffer pool when the database is deactivated.

- **CONNECTIONS:** Records an event record for each database connection event. Records when the connection ends.
- **STATEMENTS:** Records an event record for each time a SQL statement is issued by an application. Records when the SQL statement completes, or for partitioned databases, when the subsection completes.
- **TRANSACTIONS:** Records an event record each time a transaction completes. Records when the unit of work (UOW) ends.

Event monitors are created by executing the **create event monitor** SQL statement. SYSADM or DBADM authority is required to create an event monitor.

Starting and stopping event monitors

After you create an event monitor, it needs to be turned on before event monitor data is collected. Event monitors can be turned on or off by executing the **set event monitor state** SQL statement. Optionally, an event monitor can be specified to be turned on automatically each time the database is started if the **autostart** option was specified when the event monitor was created.

After an event monitor is activated, it sits quietly in the background and waits for one of the events it is associated with to occur. When such an event takes place, the event monitor collects information that is appropriate for the event that triggered it and writes that information to the event monitor's target location. If the target location is a named pipe, a stream of data is written directly to the named pipe. If the target location is a directory, the stream of data is written directly to one or more files. These files are sequentially numbered and have the extension .evt (such as 00000000.evt, 00000001.evt, and so on).

Forcing an event monitor to provide data before it is activated

It is sometimes desirable to have an event monitor to write its current monitor values to its target location before the monitor triggering event occurs. In such situations, a system administrator can force an event monitor to write all information collected so far to the appropriate output location by executing the **flush event monitor** SQL statement.

The event records that are generated by this command are recorded in the Event Monitor log with a partial record identifier. Be aware that flushing out an event monitor does not cause the event monitor values to be reset.

Viewing event monitor data

Data files that are produced by an event monitor are written in binary format, so their contents cannot be viewed directly with a text editor. DB2 provides two utilities for the purpose of viewing this data:

- **Event Analyzer:** A GUI tool that reads the information stored in an event monitor data file and produces a graphical report
- **db2evmon:** A text-based tool that reads the information stored in an event monitor data file and generates a report

The Event Analyzer is activated by entering the command **db2eva** at the system command prompt. The **db2evmon** tool is activated by entering the command **db2evmon** at the system command prompt. By default, the report produced by the **db2evmon** utility is always displayed on the panel. However, if the report is quite long, you may want to redirect the output to a file, which can then be opened with a scrollable text editor or printed.

14.2 Operating system monitoring tools

Tools used for monitoring activity at the operating system level can vary from platform to platform. In our test environment we used the AIX 5.2 maintenance level 7 operating system. Therefore, in this section, we briefly cover three common AIX system monitoring tools available: **topas**, **vmstat**, and **iostat**.

14.2.1 topas

AIX is a multiuser, multitasking operating system. Therefore, at any given time more than one program (otherwise known as a process) can be running on an AIX system. While AIX makes it appear as though all processes are executing simultaneously, the reality is that only one process can execute on each available CPU at a time. To give the illusion that multiple processes are running simultaneously, each process is given a time slice—a very short period of time where it has exclusive access to the CPU. After a process uses up its time slice, it must give up access to the CPU so that the next process can use it. A useful tool for monitoring processes that are either using a CPU or are waiting to use a CPU is the **topas** command.

The **topas** command displays statistics about activity on the system during an interval (you can specify the interval of your choosing). If executed without any options, the **topas** command displays an output similar to Example 14-5. It continues to run, updating the statistics after every interval until you press **q** to quit. This output was taken during a long running query consisting of a select of approximately eight million rows. The data was stored on the IBM System Storage N series connected through a SAN. For this output, an interval of 15 seconds was specified.

Note: The **topas** command requires the **bos.perf.tools** and **perfgent.tools** filesets to be installed on the AIX system and can only be run as root.

Example 14-5 Sample topas output

Topas Monitor for host: aixdb2						EVENTS/QUEUES		FILE/TTY	
Wed Jun 21 12:37:32 2006 Interval: 15						Cswitch	35455	Readch	1735.3
						Syscall	74571	Writech	2469.2K
Kernel	17.3	#####				Reads	68	Rawin	0
User	50.9	#####				Writes	18061	Ttyout	0
Wait	1.5					Forks	0	Igets	0
Idle	30.2	#####				Execs	0	Namei	117
						Runqueue	1.9	Dirblk	0
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out	Waitqueue	0.0		
en2	0.0	0.8	0.0	0.0	0.0				
et2	0.0	0.0	0.0	0.0	0.0				
						PAGING		MEMORY	
						Faults	266	Real,MB	4095
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ	Steals	0	% Comp	18.9
hdisk6	7.1	1570.6	61.3	1569.9	0.7	PgspIn	0	% Noncomp	81.9
hdisk0	3.5	39.0	7.4	0.0	39.0	PgspOut	0	% Client	0.
						PageIn	392		
WLM-Class		CPU%	Mem%	Disk-I/O%		PageOut	10	PAGING SPACE	
						Sios	70	Size,MB	51
								% Used	1.5
Name	PID	CPU%	PgSp	Owner		NFS (calls/sec)	% Free	98.4	
db2bp	700502	44.7	1.7	db2inst2		ServerV2	0		
db2	573506	16.1	1.6	db2inst2		ClientV2	0	Press:	
dtterm	643282	0.2	1.3	root		ServerV3	0	"h" for help	

The display is organized into a number of sections. We discuss some of these sections, which are pertinent to diagnosing problems relating to the N series product.

CPU utilization section

On the upper left of Example 14-5 on page 291, we can see a section consisting of the parameters Kernel, User, Wait, and Idle. This section shows general CPU utilization. From the output, we can see the following:

- ▶ Kernel mode processes used up 17.3% of the CPU during the interval
- ▶ User mode processes used up 50.9% of the CPU during the interval
- ▶ 1.5% of CPU time was spent waiting on I/O during the interval
- ▶ The CPU was idle for 30.2% of the time during the interval

There is also a corresponding bar chart representation of this data.

Tip: If you see the CPU spending a significant portion of its time waiting on I/O, it could be an indication that your system is disk bound, meaning that there is a disk I/O bottleneck. Further investigation on disk access would be needed.

Network interfaces section

Beneath the CPU utilization section in Example 14-5 on page 291 is the network interfaces section. We can see two network interfaces, en2 and et2. This list only shows the interfaces with the highest amount of activity and is not a complete list of network interfaces. The following statistics are collected for these interfaces:

Network Name of the network interface.

KBPS Total throughput (sent and received), in KB per second, during the interval.

I-Pack Number of packets received (input) per second during the interval.

O-Pack Number of packets sent (output) per second during the interval.

KB-In Number of KB received per second during the interval.

KB-Out Number of KB sent per second during the interval.

As you can see, there was not much activity occurring through the network during this interval. This is because our data was going through the SAN, via fibre-channel, which is not a network interface.

Tip: The network interfaces section can be useful in determining performance related problems when your storage is set up through the LAN via NFS mounts. For example, if a long running query is running unusually slow and we see the throughput of the corresponding network interface is close to its limit, then we have good evidence that the bottleneck is in the network.

Physical disks section

In Example 14-5 on page 291, beneath the network interfaces section is the physical disks section. This is not a complete list of disks. It only shows the disks with the highest amount of activity. The following statistics are collected for these disks:

Disk The name of the disk.

Busy% Percentage of time the disk was active during the interval.

KBPS	General throughput (read and write) of the disk in KB per second during the interval.
TPS	Number of transfers (I/O request to the disk) per second during the interval.
KB-Read	Number of KB read per second during the interval.
KB-Writ	Number of KB written per second during the interval.

As you can see from the output, there is a significant amount of reads done on hdisk6, which corresponds to one of our LUNs on the IBM System Storage N series. This is expected as the **topas** output was taken during a long running query.

Tip: The physical disks section can be useful in determining performance related problems when your storage is set up through a SAN. A throughput (KBPS) that is close to the limits of the fibre-channel interface is a good indication that the SAN is the bottleneck.

Processes section

The final section we will discuss from Example 14-5 on page 291 is the processes section, located at the bottom left-corner. This section displays a list of hot processes—processes that consume the most CPU cycles. The following statistics are shown:

Name	Name of the executable program executing in the process.
PID	Process ID of the process.
CPU%	Average CPU utilization of the process during the interval.
PgSp	Size of the paging space allocated to this process.
Owner	User ID of the user who owns the process.

This information is helpful in determining the root cause of performance issues. For instance, this information tells you if an application's associated processes are taking up a lot of CPU cycles, which is an indication to focus diagnosis on the application instead of the network or the storage system. You can also run **topas** with the **-P** option, which shows a more detailed version of the processes section. Example 14-6 shows a sample output of this command. From this output, you can see that the db2bp process (the db2 CLP backend process) is taking up around 93.7% of the CPU. In this case, the CLP is working close to maximum capacity, so increasing throughput from the storage system would not likely add any benefit.

Example 14-6 Sample output of the topas -P command

Topas Monitor for host:				aixdb2		Interval: 10		Wed Jun 21 16:09:26 2006				
				DATA		TEXT	PAGE	PGFAULTS				
USER	PID	PPID	PRI	NI	RES	RES	SPACE	TIME	CPU%	I/O	OTH	COMMAND
db2ins	544912	1	91	20	439	99	439	0:50	93.7	0	0	db2bp
db2ins	311320	700478	75	20	433	6	433	0:16	31.7	0	0	db2
db2ins	565268	389150	62	20	1018	13	1359	0:05	5.8	0	0	db2sysc
root	643282	696454	61	20	356	9	356	0:00	3.2	0	0	dtterm
db2ins	712864	446684	60	20	106	13	308	0:00	1.1	247	360	db2sysc
root	237694	196620	60	20	642	319	642	0:16	0.2	0	0	X
db2ins	479310	610364	60	20	97	13	292	0:00	0.1	0	0	db2sysc
root	163920	0	36	41	14	0	14	0:02	0.0	0	0	netm
root	168018	0	37	41	31	0	31	0:20	0.0	0	0	gil
root	172116	0	16	41	17	0	17	0:00	0.0	0	0	wlmsched
root	180436	1	60	20	159	1	159	2:18	0.0	0	0	syncd
root	184440	241790	60	20	503	33	503	0:00	0.0	0	0	dtsession
root	188576	295104	60	20	292	28	298	0:00	0.0	0	0	ervicerMtd

root	196620	1	60	20	45	39	102	0:00	0.0	0	0	dtlogin
root	200934	1	60	20	48	0	48	0:22	0.0	0	0	rpc.lockd
root	204922	0	50	41	14	0	14	0:00	0.0	0	0	jfsz
root	217208	0	60	20	14	0	14	0:00	0.0	0	0	lvmdb
root	221296	0	60	20	20	0	20	0:00	0.0	0	0	dog
root	225456	1	60	20	71	13	233	0:00	0.0	0	0	db2sysc
root	229496	1	60	20	227	21	227	0:00	0.0	0	0	errdemon

14.2.2 vmstat

Another useful tool for obtaining information about processes is a program called **vmstat**. Specifically, the **vmstat** command shows an overall picture of CPU, paging, and memory usage. If the **vmstat** command is used without any of the options that are available, or with the interval option, then the first line of numbers returned is an average of all activity since system reboot.

Example 14-7 shows a sample output of data that was collected by the **vmstat** program. In this example, we issued the command **vmstat 30 20** to specify that we want 20 lines of data at 30 second intervals. During the gathering of this data, we ran a long running query against data residing on the IBM System Storage N series through a SAN. You can see evidence of a spike in CPU utilization and memory usage in the output which corresponds to the time that the query was run.

Example 14-7 Sample vmstat output

```
# vmstat 30 20
System Configuration: 1cpu=2 mem=4096MB
kthr      memory          page        faults          cpu
-----
 r  b   avm    fre   re  pi  po  fr   sr  cy   in    sy    cs us sy id wa
1  1 189719  8351    0   0   0   5    8   0 254  1763   399   1   1 98   0
2  0 196489  1571    0   0   0   0    0   0 306 62148 30084  44  14 40   2
1  0 196492  1568    0   0   0   0    0   0 306 73075 35604  51  16 31   1
1  0 196492  1568    0   0   0   0    0   0 306 73085 35631  52  16 31   1
2  0 196492  1568    0   0   0   0    0   0 305 72853 35702  51  17 31   1
1  1 199478   860    0   0   0  78   78   0 315 78350 34447  51  17 29   2
1  1 199478   860    0   0   0   0    0   0 305 72995 35631  52  16 31   1
1  0 199478   886    0   0   0   0    0   0 306 72964 35608  51  17 31   1
1  1 199478   886    0   0   0   0    0   0 305 72963 35643  51  17 31   1
2  1 199478   846    0   0   0   0    0   0 309 73289 35583  51  16 31   2
2  1 199478   846    0   0   0   0    0   0 305 72945 35677  52  16 31   1
1  0 199480   870    0   0   0   0    0   0 306 75392 35484  51  15 32   1
2  1 199480   879    0   0   0   0    0   0 305 72768 35633  51  17 31   1
1  1 199480   879    0   0   0   0    0   0 306 73060 35562  51  16 31   1
1  1 199480   879    0   0   0   0    0   0 305 72980 35666  51  16 31   1
2  0 199524   862    0   0   0   1    1   0 311 73667 35640  52  16 31   1
2  0 192784  7610    0   0   0   0    0   0 289 50231 24335  35  11 53   1
1  1 192778  7616    0   0   0   0    0   0 256   3092   160   0   1 99   0
0  0 192778  7616    0   0   0   0    0   0 254    692   143   0   1 99   0
1  0 192833  7603    0   0   0   0    0   0 256    812   143   0   0 99   0
#
```

The output of **vmstat** is arranged into the following main headings:

- kthr** Shows the number of kernel threads in various queues. **r** shows the average number of runnable kernel threads. **b** shows the average number of kernel threads in the VMM wait queue.
- memory** Shows general memory usage. **avm** is the amount of active virtual memory. **fre** is the amount of free real memory.
- page** Shows paging activity. **re** is currently not used. **pi** shows the number of paged in pages. **po** shows the number of paged out pages. **fr** shows the number of freed pages. **sr** shows the number of pages examined by the page-replacement algorithm. **cy** shows the number of cycles per second of the clock algorithm.
- faults** Shows process control, such as traps and interrupt rate. **in** shows the number of device interrupts. **sy** shows the number of system calls. **cs** shows the number of context switches.
- cpu** Shows the percentage breakdown of CPU time usage during the interval. **us** shows the percentage of CPU time spent in user mode. **sy** shows the percentage of CPU time in system mode. **id** shows the percentage of CPU idle time. **wa** shows the percentage of CPU time spent waiting for disk I/O.

vmstat also displays disk performance. You can do this by specifying the name of the disk within the **vmstat** command. For instance, the command **vmstat hdisk6 2 10** would additionally output disk transfer statistics for **hdisk6** at two second intervals, for 10 intervals. You can specify up to four disks by this method. However, there is another tool we can use that shows much more information about disk usage. It is **iostat**.

14.2.3 iostat

The **iostat** command shows information about disk utilization. You can specify the size of intervals in which this data is collected. The output shows information about all disks on the system and includes information such as throughput and amount of reads and writes. Example 14-8 shows a sample output of this command.

Note: This tool is mainly used for storage created in a SAN environment. In a LAN environment, network storage is accessed through NFS mounts (or CIFS on Windows environments), which are not displayed with this command.

Example 14-8 Sample iostat output

```
# iostat 2 10
```

```
System configuration: lcpu=2 disk=11
```

```
<... portion removed...>
```

tty:	tin	tout	avg-cpu:	% user	% sys	% idle	% iowait
	0.0	117.1		50.6	16.6	32.4	0.4
Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn		
hdisk1	0.0	0.0	0.0	0	0		
hdisk0	0.0	0.0	0.0	0	0		
hdisk2	0.0	0.0	0.0	0	0		
hdisk5	0.0	0.0	0.0	0	0		
hdisk3	0.0	0.0	0.0	0	0		
hdisk4	0.0	0.0	0.0	0	0		

cd0	0.0	0.0	0.0	0	0
hdisk9	0.0	0.0	0.0	0	0
hdisk8	0.0	0.0	0.0	0	0
hdisk6	7.1	1390.7	43.5	3136	0
hdisk7	0.0	0.0	0.0	0	0
#					

There are the following three reports in this output:

- TTY report** Displays TTY information for all TTY devices. **tin** shows the amount of TTY input. **tout** shows TTY output.
- CPU report** Displays CPU statistics. This information is also available from the **topas** and **vmstat** commands. **% user**, **% sys**, **% idle**, and **% iowait** correspond to % of CPU spent on user processes, % of CPU spent on system processes, % of CPU idle time, and % of CPU waiting for I/O, respectively.
- Disks report** Displays disk statistics. **Disks** shows the names of the physical volumes. **%tm_act** shows the percentage of time that the disk was active. **kbps** shows the throughput (read and write) of the disk in KB per second. **tps** shows the number of transfers (I/O request) per second. **Kb_read** shows the number of KB read per second. **Kb_wrtn** shows the number of KB written per second.

The report we are most interested in here is the Disks report. Taken alone, there is no unacceptable value for any of the above fields because statistics are too closely related to application characteristics, system configuration, and type of physical disk drives and adapters. Therefore, when you are evaluating data, look for patterns and relationships. The most common relationship is between disk utilization (**%tm_act**) and data transfer rate (**tps**).

To draw any valid conclusions from this data, you have to understand the application's disk data access patterns such as sequential, random, or combination, as well as the type of physical disk drives and adapters on the system. For example, if an application reads/writes sequentially, you should expect a high disk transfer rate (Kbps) when you have a high disk busy rate (**%tm_act**). Columns **Kb_read** and **Kb_wrtn** can confirm an understanding of an application's read/write behavior. However, these columns provide no information about the data access patterns.

Generally you do not need to be concerned about a high disk busy rate (**%tm_act**) as long as the disk transfer rate (Kbps) is also high. However, if you get a high disk busy rate and a low disk transfer rate, you may have a fragmented logical volume, file system, or individual file.

Discussions of disk, logical volume and file system performance sometimes lead to the conclusion that the more drives you have on your system, the better the disk I/O performance. This is not always true because there is a limit to the amount of data that can be handled by a host bus adapter (HBA). The HBA can also become a bottleneck. If, on the storage system, you have an aggregate that spans many disks and only one HBA on the AIX server, you might benefit from using multiple HBAs.

To see if a particular HBA is saturated, use the **iostat** command and add up all the Kbps amounts for the LUNs attached to the HBA. For maximum aggregate performance, the total of the transfer rates (Kbps) must be below the HBA throughput rating. In most cases, use 70% of the throughput rate. In operating system versions later than 4.3.3, the **-a** or **-A** option will display this information.

14.3 IBM System Storage N series monitoring tools

This section discusses the tools that are available for IBM System Storage N series.

14.4 Performance factors

Performance monitoring is a collection of data from several factors to create a conclusion and a subsequent action plan. Following are the performance influencers with the N series product:

- ▶ CPU utilization
 - Single
 - Multiple
- ▶ Fibre channel Throughput
- ▶ Disk write performance
- ▶ Cache Hit rate
- ▶ Network efficiency
- ▶ Network design
- ▶ Workload Characteristics
- ▶ Client OS
 - Client Hardware

14.5 The Performance toolchest

In the analysis of performance, the following tools and approaches are available:

- ▶ Configuration analysis—make sure the hardware is configured properly for the highest performance.
 - Multipathing is enabled
 - Multiple NICS are available
 - Proper Zoning
 - Number of network hops
 - Network settings
 - Speed
 - Negotiation
 - Frames
 - Memory
 - Client
 - IBM System Storage N series
 - Persistent bindings for FCP
 - Persistent mounts for NFS
 - Domain Controllers for CIFS
 - Make sure the Domain Controller is not overloaded
 - Application software configuration

- ▶ Performance monitoring and tools

Later in this chapter, we discuss the performance monitoring and tools in detail. The usage and application of these tools will help determine performance bottlenecks.

- ▶ Fragmentation

More discussion later in this chapter will occur. Volume fragmentation is one of the factors that can reduce IBM System Storage N series performance.

- ▶ A review of volume and RAID group sizing to ensure the best configuration is used.

14.5.1 Performance and monitoring

This section covers some of the tools available with the IBM System Storage N series.

Sysstat

Sysstat reports aggregated filer performance statistics such as the current CPU utilization, the amount of network I/O, the amount of disk I/O, and the amount of tape I/O. When invoked with no arguments **sysstat** prints a new line of statistics every 15 seconds. Use control-C or set the interval count (-c *count*) to stop **sysstat**.

Example 14-9 Sysstat

CPU	NFS	CIFS	HTTP	Net kB/s		Disk kB/s		Tape kB/s		Cache
				in	out	read	write	read	write	age
2%	0	0	0	1	0	108	97	0	0	>60
1%	0	0	0	1	0	3	11	0	0	>60
2%	0	0	0	1	0	100	90	0	0	>60
1%	0	0	0	1	0	3	11	0	0	>60
2%	0	0	0	1	0	119	100	0	0	>60
1%	0	0	0	1	0	2	6	0	0	>60
2%	0	0	0	1	0	103	97	0	0	>60
1%	0	0	0	1	0	3	11	0	0	>60

- ▶ CPU - cpu utilization
- ▶ NFS - NFS operations
- ▶ CIFS - CIFS operations
- ▶ HTTP - HTTP operations
- ▶ Net kB/s in - incoming network traffic
- ▶ Net kB/s - outgoing network traffic
- ▶ Disk kB/s read - read data transfer amount
- ▶ Disk KB/s write - write data transfer amount
- ▶ Tape kB/s read - amount of data read from tape
- ▶ Tape KB/s write - amount of data written to tape
- ▶ Cache Age - age of data in cache in seconds

Qtree Stats

The **qtree stats** command shows the number of NFS or CIFS operations, as a result of user accesses into qtrees.

Example 14-10 qtree command results

Volume	Tree	NFS ops	CIFS ops
-----	-----	-----	-----
vol1bkup	unix_qtree	148	546
vol1bkup	ntfs_qtree	20	253
vol1bkup	mixed_qtree	11	127

N series maintains counters for each qtree in each of the N series volumes. The values displayed correspond to the operations on qtrees, since the volume (in which the qtrees exist) was created, or since it was made online on the IBM System Storage N series, or since the counters were last reset, whichever occurred most recently.

The "-z" flag can be used to reset the counters. The "-i" flag can be used to show internal operations on qtrees when priv set is Advanced (Example 14-11).

Example 14-11 priv set -i

Volume	Tree	NFS ops	CIFS ops	Internal ops
-----	-----	-----	-----	-----
vol1bkup	unix_qtree	0	0	0
vol1bkup	ntfs_qtree	0	0	0
vol1bkup	mixed_qtree	0	0	0

Windows Perfmon

You can capture IBM System Storage N series performance data using Microsoft Windows Perfmon. You first have to add the desired counters to Perfmon (Figure 14-1).

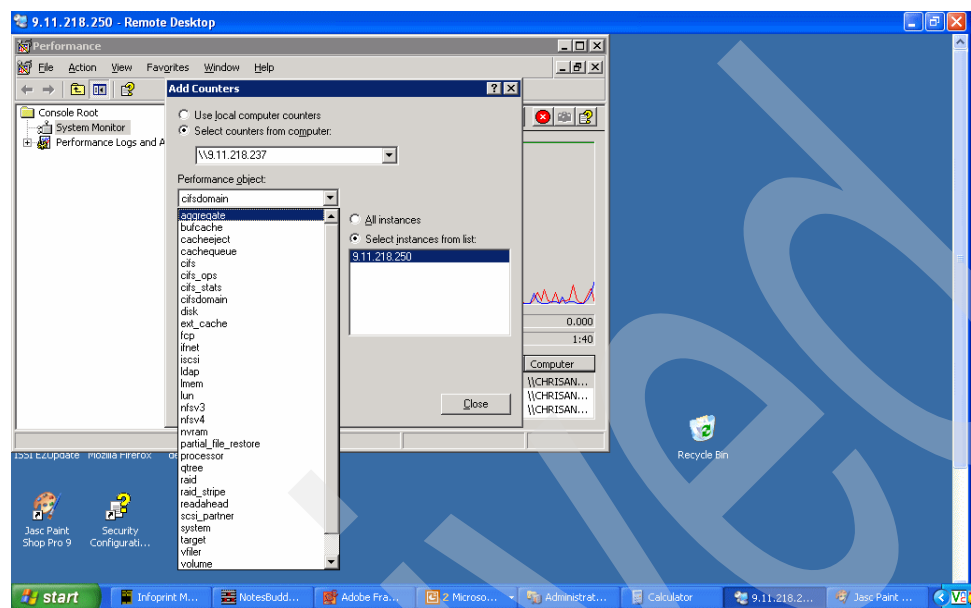


Figure 14-1 Performance counters

In the Figure 14-2, NVRAM statistics were selected.

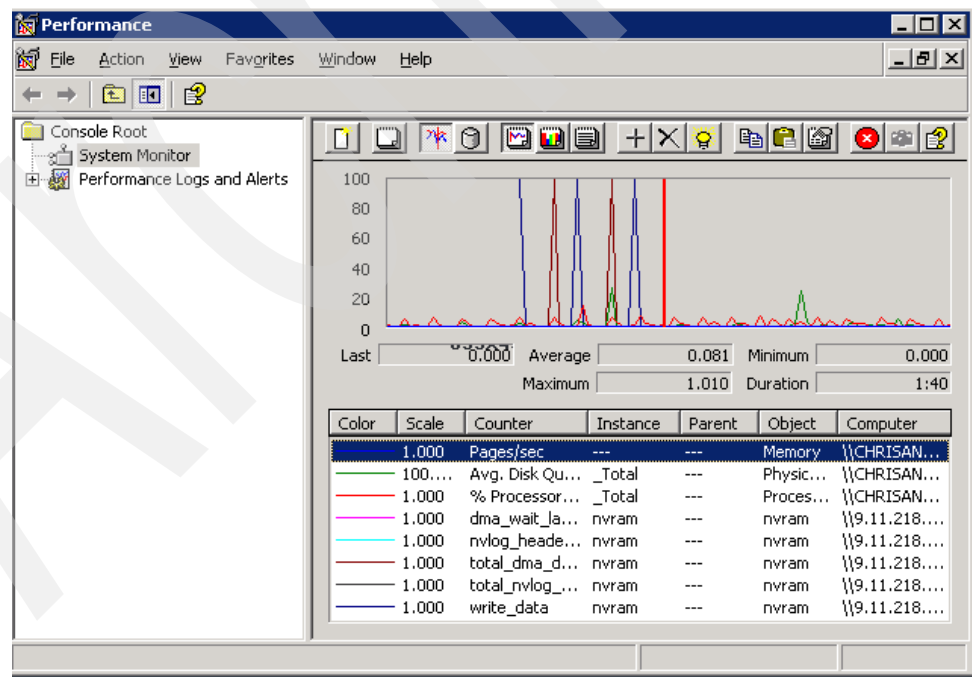


Figure 14-2 NVRAM counters

Stats command

The stats command is a way to collect or view statistical data on an N series product (Example 14-12).

Example 14-12 Stats syntax

```
itsotuc3*> stats
The following commands are available; for more information
type "stats help <command>"
explain          list          start          stop
help             show
stats help - Display command help.
stats list - List available objects, instances or counters in system.
stats explain - Explain available objects or counters in system.
stats show - Show all or selected statistics with formatting.
stats start - Start background statistic collection.
stats stop - Stop all background operations and discard results.
```

- ▶ stats show—current counter values are displayed as a snapshot
- ▶ stats show -i—repeating, in which counter values are displayed multiple times at a fixed interval. Example 14-13 shows repeating counters every five seconds.

Example 14-13 stats example

```
stats show -i 5
```

Use the `-p` option to use a specific XML file that was configured to show certain statistics in a certain format.

Using the `-l` option, you can use a specific identifier for your statistics collection. If you do not use the `-l` option, a unique identifier is automatically created.

You can use the `-r` or `-c` options to show the output of your statistics in rows (Example 14-14) or columns.

Example 14-14 Row format

```
stats show -r system
system:system:nfs_ops:0/s
system:system:cifs_ops:1/s
system:system:http_ops:0/s
system:system:dafs_ops:0/s
system:system:fcps_ops:0/s
system:system:iscsi_ops:0/s
system:system:net_data_recv:1KB/s
system:system:net_data_sent:0KB/s
system:system:disk_data_read:32KB/s
system:system:disk_data_written:0KB/s
system:system:cpu_busy:3%
system:system:avg_processor_busy:1%
system:system:total_processor_busy:5%
system:system:num_processors:4
system:system:time:1170200293s
system:system:uptime:1910405s
```

You can fine the object you are interested in by using the `stats list` command (Example 14-15 on page 301).

Example 14-15 stats list

stats list objects

Objects:

- prished
- priorityqueue
- vfiler
- qtree
- aggregate
- iscsi
- fcg
- cifs
- volume
- lun
- target
- nfsv3
- ifnet
- processor
- disk
- system

Conversely if you want to see available counters you can use the **stats list counters** command (Example 14-16) stats.

Example 14-16 stats list counters

stats list counters

Counters for object name: prished

- queued
- queued_max

Counters for object name: priorityqueue

- weight
- usr_weight
- usr_sched_total
- usr_pending
- avg_usr_pending_ms
- usr_queued_total
- sys_sched_total
- sys_pending
- avg_sys_pending_ms
- sys_queued_total
- usr_read_limit
- max_user_reads
- sys_read_limit
- max_sys_reads
- usr_read_limit_hit
- sys_read_limit_hit
- nvlog_limit
- nvlog_used_max
- nvlog_limit_full

Counters for object name: vfiler

- vfiler_cpu_busy
- vfiler_net_data_rcv
- vfiler_net_data_sent

vfiler_read_ops
vfiler_write_ops
vfiler_misc_ops
vfiler_read_bytes
vfiler_write_bytes

Counters for object name: qtree

nfs_ops
cifs_ops

Counters for object name: aggregate

total_transfers
user_reads
user_writes
cp_reads
user_read_blocks
user_write_blocks
cp_read_blocks

Counters for object name: iscsi

iscsi_ops
iscsi_write_data
iscsi_read_data

Counters for object name: fcp

fcp_ops
fcp_write_data
fcp_read_data

Counters for object name: cifs

cifs_ops
cifs_latency

Counters for object name: volume

avg_latency
total_ops
read_data
read_latency
read_ops
write_data
write_latency
write_ops
other_latency
other_ops

Counters for object name: lun

read_ops
write_ops
other_ops
read_data
write_data
queue_full
avg_latency
total_ops

Counters for object name: target

- read_ops
- write_ops
- other_ops
- read_data
- write_data
- queue_full

Counters for object name: nfsv3

- nfsv3_ops
- nfsv3_read_latency
- nfsv3_read_ops
- nfsv3_write_latency
- nfsv3_write_ops

Counters for object name: ifnet

- recv_packets
- recv_errors
- send_packets
- send_errors
- collisions
- recv_data
- send_data
- recv_mcasts
- send_mcasts
- recv_drop_packets

Counters for object name: processor

- processor_busy

Counters for object name: disk

- total_transfers
- user_read_chain
- user_reads
- user_write_chain
- user_writes
- cp_read_chain
- cp_reads
- guarenteed_read_chain
- guaranteed_reads
- guarenteed_write_chain
- guaranteed_writes
- user_read_latency
- user_read_blocks
- user_write_latency
- user_write_blocks
- cp_read_latency
- cp_read_blocks
- guarenteed_read_latency
- guaranteed_read_blocks
- guarenteed_write_latency
- guaranteed_write_blocks
- disk_busy

Counters for object name: system

```
nfs_ops
cifs_ops
http_ops
dafs_ops
fcg_ops
iscsi_ops
net_data_recv
net_data_sent
disk_data_read
disk_data_written
cpu_busy
avg_processor_busy
total_processor_busy
num_processors
time
uptime
```

If you want to send the output of your statistics to a file, use the `-o` option.

Using IBM System Storage N series with Oracle

Faint background watermark text: 'ARCHIVED'

Archived



Backup, restore, and disaster recovery of Oracle using N series

This chapter discusses backup, restore, and disaster recovery of Oracle using the IBM System Storage N series.

15.1 How to back up data from N series

Data that is stored on an IBM System Storage N series can be backed up to online storage, nearline storage, or tape. The protocol used to access data while a backup is occurring must always be considered. When NFS and CIFS are used to access data, Snapshot and SnapMirror can be used and will always result in consistent copies of the file system. They must coordinate with the state of the Oracle Database to ensure database consistency.

With Fibre Channel or iSCSI protocols, Snapshot copies and SnapMirror commands must always be coordinated with the server. The file system on the server must be blocked and all data flushed to the N series product before invoking the Snapshot command.

Data can be backed up within the same N series to another N series or to a tape storage device. See Figure 15-1. Tape storage devices can be directly attached to an IBM System Storage N series, or they can be attached to an Ethernet or Fibre Channel network, and the IBM System Storage N series can be backed up over the network to the tape device.

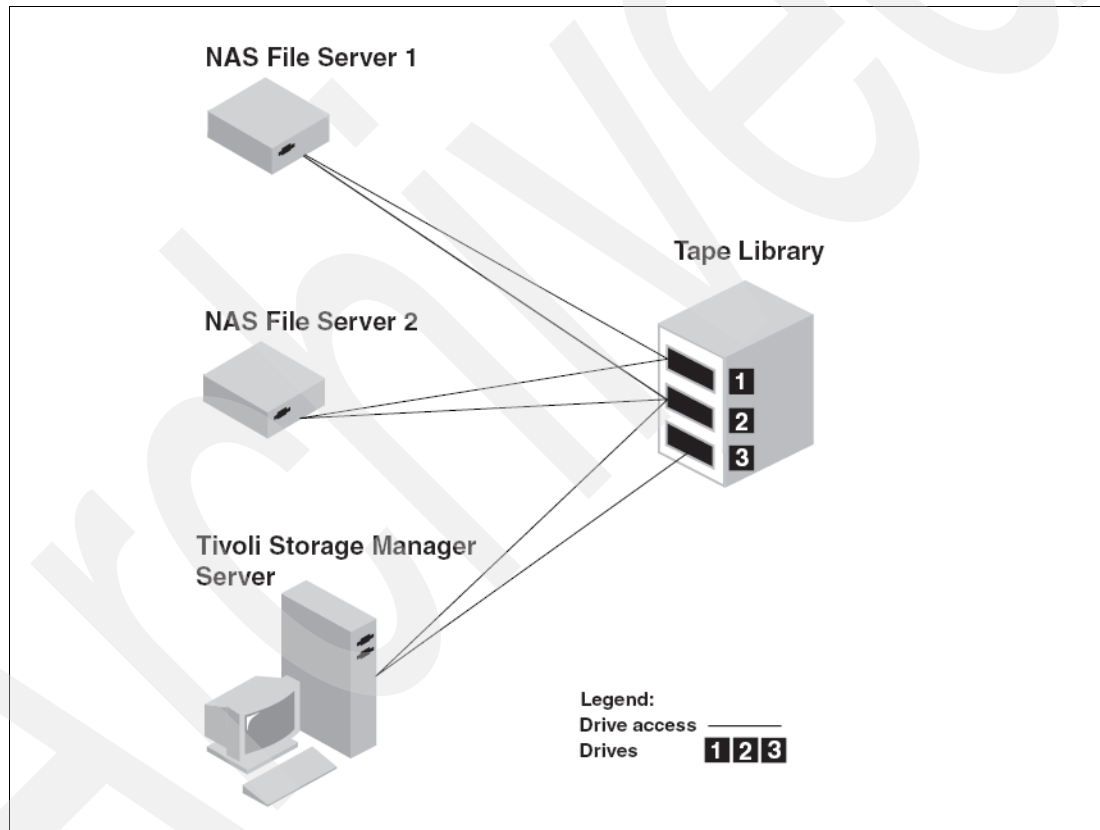


Figure 15-1 Backup to Tape Library

Possible methods for backing up data on IBM System Storage N series include the following:

- ▶ Use SnapManager for Oracle to create your online or offline backups
- ▶ Use automated Snapshot copies to create online backups
- ▶ Use scripts on the server that `rsh` to the IBM System Storage N series to invoke Snapshot copies to create online backups
- ▶ Use SnapMirror to mirror data to another IBM System Storage N series
- ▶ Use SnapVault to vault data to another IBM System Storage N series

- ▶ Use server operating system–level commands to copy data to create backups
- ▶ Use NDMP commands to back up data to an N series product
- ▶ Use NDMP commands to back up data to a tape storage device
- ▶ Use third-party backup tools to back up the IBM System Storage N series to tape or to other storage devices

15.1.1 Creating online backups using Snapshot copies

The IBM System Storage N series Snapshot technology makes extremely efficient use of storage by storing only block-level changes between creating each successive Snapshot copy. Since the Snapshot process is virtually instantaneous, backups are fast and simple. Snapshot copies can be automatically scheduled. They can be called from a script running on a server, or they can be created via SnapDrive or SnapManager.

Data ONTAP includes a scheduler to automate Snapshot backups. Use automatic Snapshot copies to back up non application data, such as home directories.

Database and other application data should be backed up when the application is in its backup mode. For Oracle Databases this means placing the database tablespaces into hot backup mode prior to creating a Snapshot copy.

Note: IBM System Storage N series recommends using Snapshot copies for performing cold or hot backup of Oracle Databases. No performance penalty is incurred for creating a Snapshot copy. We recommend that you turn off the automatic Snapshot scheduler and coordinate the Snapshot copies with the state of the Oracle Database.

15.1.2 Recovering individual files from a Snapshot copy

Individual files and directories can be recovered from a Snapshot copy by using native commands on the server, such as the UNIX `cp` command or dragging and dropping in Microsoft Windows. Data can also be recovered using the single-file **SnapRestore** command. Use the method that works most quickly.

15.1.3 Recovering data using SnapRestore

SnapRestore quickly restores a file system to an earlier state preserved by a Snapshot copy. SnapRestore can be used to recover an entire volume of data or individual files within that volume.

When using SnapRestore to restore a volume of data, the data on that volume should belong to a single application. Otherwise operation of other applications may be adversely affected.

The single-file option of SnapRestore allows individual files to be selected for restore without restoring all of the data on a volume. Be aware that the file being restored using SnapRestore cannot exist anywhere in the active file system. If it does, the appliance will silently turn the single-file SnapRestore into a copy operation. This may result in the single-file SnapRestore taking much longer than expected (normally the command executes in a fraction of a second) and also requires that sufficient free space exist in the active file system.

Note: IBM System Storage N series recommends using SnapRestore to instantaneously restore an Oracle Database. SnapRestore can restore the entire volume to a point-in-time in the past or can restore a single file. It is advantageous to use SnapRestore on a volume level, as the entire volume can be restored in minutes, and this reduces downtime while performing Oracle Database recovery. If using SnapRestore on a volume level, we recommend storing the Oracle log files, archive log files, and copies of control files on a separate volume from the main data file volume, and use SnapRestore only on the volume containing the Oracle data files.

15.1.4 Backup and recovery best practices

This section combines the IBM System Storage N series data protection technologies and products previously described into a set of best practices for performing Oracle hot backups (online backups) for backup, recovery, and archival purposes using primary storage (IBM System Storage N series with high-performance Fibre Channel disk drives) and nearline storage (IBM System Storage N series with low-cost, high-capacity ATA and SATA disk drives). This combination of primary storage for production databases and nearline disk-based storage for backups of the active data set improves performance and lowers the cost of operations. Periodically moving data from primary to nearline storage increases free space and improves performance, while generating considerable cost savings.

SnapVault and database backups

Oracle Databases can be backed up while they continue to run and provide service, but must first be put into a special hot backup mode.

Certain actions must be taken before and after a Snapshot copy is created on a database volume. Since these are the same steps taken for any other backup method, many database administrators probably already have scripts that perform these functions.

While SnapVault Snapshot schedules can be coordinated with appropriate database actions by synchronizing clocks on the N series product and database server, it is easier to detect potential problems if the database backup script creates the Snapshot copies using the SnapVault **snap create** command.

In this example, a consistent image of the database is created every hour, keeping the most recent five hours of Snapshot copies (the last five copies). One Snapshot version is retained per day for a week, and one weekly version is retained at the end of each week. On the SnapVault secondary software, a similar number of SnapVault Snapshot copies is retained.

Process for performing Oracle hot backups with SnapVault

Following is the high-level process for performing Oracle hot backups with SnapVault. The steps are further detailed into specific steps later in the section.

1. Set up the target IBM System Storage N series (itsotuc2) system to talk to the primary IBM System Storage N series (itsotuc1).
2. Set up the schedule for the number of Snapshot copies to retain on each of the storage devices using the script-enabled SnapVault schedule on both the primary and target N series.
3. Start the SnapVault process between the primary and target N series products.

4. Create shell scripts to drive Snapshot copies through SnapVault on both the primary and target N series product to perform Oracle hot backups.
5. Create a cron-based schedule script on the host to drive hot backup scripts for Snapshot copies driven by SnapVault, as previously described.

Step 1: Set up the target IBM System Storage N series (itsotuc2) to talk to the primary IBM System Storage N series (itsotuc1).

The example in this subsection assumes that the primary N series for database storage is named “itsotuc1”, and the N series for database archival is named “itsotuc2.”

The following steps occur on the primary N series, “itsotuc1”:

- a. License SnapVault, and enable it on N series, “itsotuc1”. See Example 15-1 and Figure 15-2.

Example 15-1 License add and option enablement

```
itsotuc1> license add ABCDEFG
itsotuc1> options snapvault.enable on
itsotuc1> options snapvault.access host=itsotuc2
```

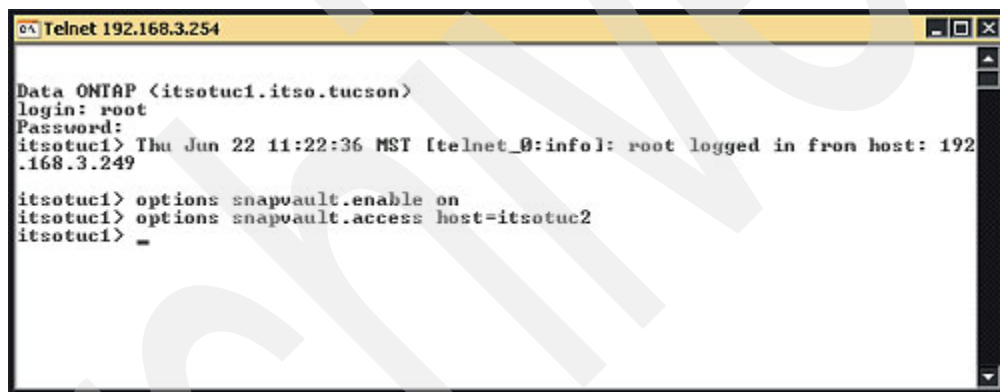
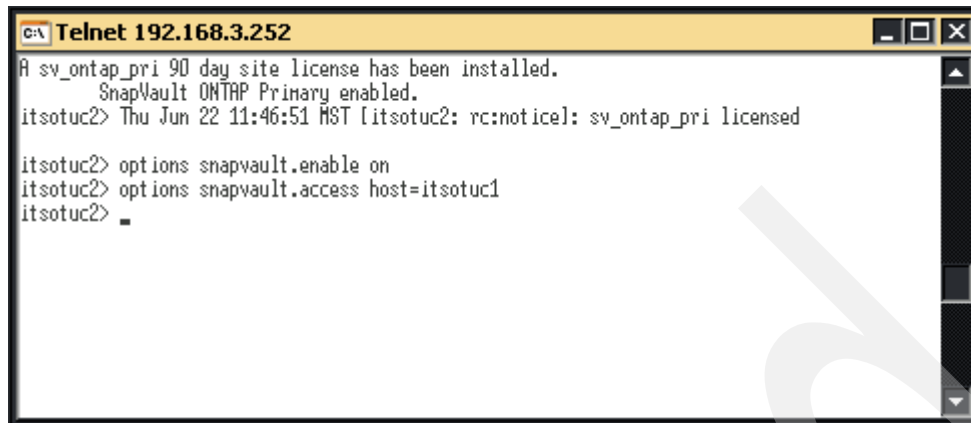


Figure 15-2 Options enable on primary N series

- b. License SnapVault, and enable it on the target N series, “itsotuc2”. See Example 15-2 and Figure 15-2 on page 311.

Example 15-2 Licensing

```
itsotuc2> license add ABCDEFG
itsotuc2> options snapvault.enable on
itsotuc2> options snapvault.access host=itsotuc1
```

A Telnet session window titled "Telnet 192.168.3.252" showing the configuration of SnapVault on a target N series. The output shows that a 90-day site license has been installed for SnapVault ONTAP Primary, and the primary is enabled. The user 'itsotuc2' is logged in. The user then enters the command 'options snapvault.enable on' and 'options snapvault.access host=itsotuc1'.

```
G:\ Telnet 192.168.3.252
A sv_ontap_pri 90 day site license has been installed.
  SnapVault ONTAP Primary enabled.
itsotuc2> Thu Jun 22 11:46:51 MST [itsotuc2: rc:notice]: sv_ontap_pri licensed

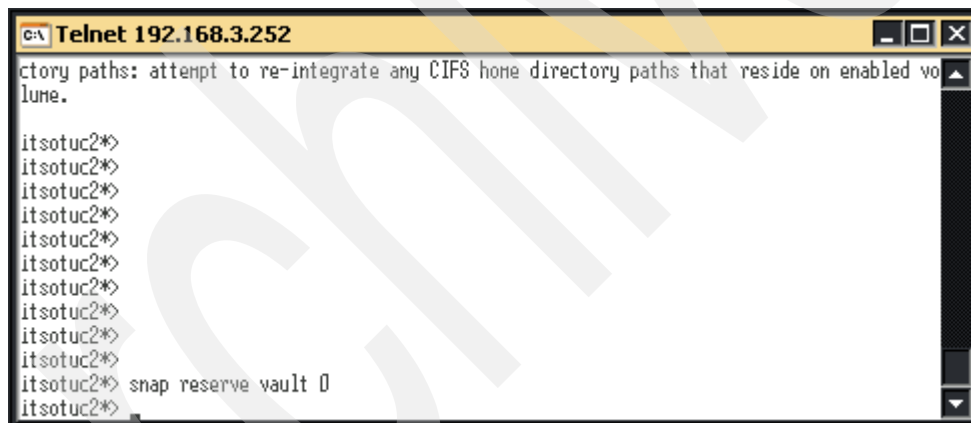
itsotuc2> options snapvault.enable on
itsotuc2> options snapvault.access host=itsotuc1
itsotuc2> _
```

Figure 15-3 Options enable on target N series

- c. Create a volume for use as a SnapVault destination in the target N series, "itsotuc2". See Example 15-3 and Figure 15-4.

Example 15-3 creating a volume

```
itsotuc2> vol create vault -r 10 10
itsotuc2> snap reserve vault 0
```

A Telnet session window titled "Telnet 192.168.3.252" showing the creation of a SnapVault volume. The output shows that the volume 'vault' has been created with a reservation of 10. The user 'itsotuc2' is logged in. The user then enters the command 'snap reserve vault 0'.

```
G:\ Telnet 192.168.3.252
ctory paths: attempt to re-integrate any CIFS home directory paths that reside on enabled vo
lume.

itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*>
itsotuc2*> snap reserve vault 0
itsotuc2*> _
```

Figure 15-4 snap reserve

Step 2: Set up schedules (disable automatic Snapshot copies) on both primary and target N series.

- a. Disable the normal Snapshot schedule on the primary and secondary IBM System Storage N series, which will be replaced by SnapVault Snapshot schedules. See Example 15-4, Figure 15-5 on page 313, Example 15-5 on page 313, and Figure 15-6 on page 313.

Example 15-4 Snapshot schedule

```
itsotuc1> snap sched oradata 0 0 0
```

```
Telnet 192.168.3.254
Data ONTAP <itsotuc1.itso.tucson>
login: root
Password:
itsotuc1> Thu Jun 22 14:51:44 MST [telnet_0:info]: root logged in from host: 192
.168.3.249

itsotuc1> snap sched oradata 0 0 0
itsotuc1> _
```

Figure 15-5 snap sched on primary

Example 15-5 snap schedule disable

```
itsotuc2> snap sched vault 0 0 0
```

```
Telnet 192.168.3.254
Data ONTAP <itsotuc1.itso.tucson>
login: root
Password:
itsotuc1> Thu Jun 22 14:51:44 MST [telnet_0:info]: root logged in from host: 192
.168.3.249

itsotuc1> snap sched oradata 0 0 0
itsotuc1> _
```

Figure 15-6 snap sched on target

- b. Set up a SnapVault Snapshot schedule to be script driven on N series, itsotuc1, for the “oradata” volume. See Figure 15-6. This command disables the schedule and also specifies how many of the named Snapshot copies to retain.

This schedule creates a Snapshot copy called sv_hourly and retains the most recent five copies, but it does not specify when to create the Snapshot copies specified by a cron script, described later in this procedure. See Example 15-6.

Example 15-6 Snapshot schedule

```
itsotuc1> snapvault snap sched oradata sv_hourly 5@-
```

Similarly, this schedule creates a Snapshot copy called sv_daily and retains only the most recent copy. It does not specify when to create the Snapshot copy. See Example 15-7.

Example 15-7 SnapVault sc

```
itsotuc1> snapvault snap sched oracle sv_daily 1@-
```

This schedule creates a Snapshot copy called sv_weekly and retains only the most recent copy. It does not specify when to create the Snapshot copy.

Example 15-8 Weekly schedule

```
itsotuc1> snapvault snap sched oracle sv_weekly 10-
```

- c. Set up the SnapVault Snapshot schedule to be script driven on the target N series, itsotuc2, for the SnapVault destination volume, "vault." This schedule also specifies how many of the named Snapshot copies to retain.

This schedule creates a Snapshot copy called sv_hourly and retains the most recent five copies, but does not specify when to create the Snapshot copies. That is done by a cron script, described later in this procedure see Example 15-9.

Example 15-9 Hourly schedule

```
itsotuc2> snapvault snap sched vault sv_hourly 50-
```

Similarly, this schedule creates a Snapshot copy called sv_daily (Example 15-10) and retains only the most recent copy. It does not specify when to create the Snapshot copy.

Example 15-10 Daily schedule

```
itsotuc2> snapvault snap sched vault sv_daily 10-
```

This schedule creates a Snapshot copy called sv_weekly (Example 15-11) and retains only the most recent copy. It does not specify when to create the Snapshot copy.


Example 15-11 Weekly schedule

```
itsotuc2> snapvault snap sched vault sv_weekly 10-
```

Step 3: Start the SnapVault process between primary and target N series products.

At this point, the schedules are configured on both the primary and secondary systems, and SnapVault is enabled and running. However, SnapVault does not know which volumes or qtrees to back up or where to store them on the secondary. Snapshot copies are created on the primary, but no data is transferred to the secondary.

To provide SnapVault with this information, use the SnapVault **start** command on the secondary, as shown in Figure 15-7:



```
GA Telnet 192.168.3.252
itsotuc2> snapvault start -s 192.168.3.254:/vol/oradata/- /vol/vault/oradata
Snapvault configuration for the qtree has been set.
Transfer started.
Monitor progress with 'snapvault status' or the snapmirror log.
itsotuc2>
```

Figure 15-7 snapvault start

Step 4: Create Oracle hot backup script enabled by SnapVault.

Example 15-12 shows the sample script defined in “/home/oracle/snapvault/sv-dohot-daily.sh”.

Example 15-12 Sample script

```
#!/bin/csh -f
# Place all of the critical tablespaces in hot backup mode.
$ORACLE_HOME/bin/sqlplus system/oracle @begin.sql
# Create a new SnapVault Snapshot copy of the database volume on the primary
IBM N series
rsh -l root itsotuc1 snapvault snap create oracle sv_daily
# Simultaneously 'push' the primary IBM N series Snapshot copy to the secondary
IBM N series system
rsh -l root itsotuc2 snapvault snap create vault sv_daily
# Remove all affected tablespaces from hot backup mode.
$ORACLE_HOME/bin/sqlplus system/oracle @end.sql
```

Note: The “@begin.sql” and “@end.sql” scripts contain sql commands to put the database’s tablespaces into hot backup mode (begin.sql) and then to take them out of hot backup mode (end.sql).

Step 5: Use cron script to drive Oracle hot backup script enabled by SnapVault from step 4.

A scheduling application such as cron on UNIX systems or the Windows task scheduler program on Windows systems is used to create an sv_hourly Snapshot copy each day at every hour, except 11:00 p.m., and a single sv_daily Snapshot copy each day at 11:00 p.m., except on Saturday evenings when an sv_weekly Snapshot copy is created instead.

Sample cron script for automation is in Example 15-13

Example 15-13 cron script

```
# sample cron script with multiple entries for Oracle hot backup
# using SnapVault, IBM N series (itsotuc1), and IBM N series(itsotuc2)
# Hourly Snapshot copy/SnapVault at the top of each hour 0 * * * *:
/home/oracle/snapvault/sv-dohot-hourly.sh
# Daily Snapshot copy/SnapVault at 2:00 a.m. every day except on Saturdays 0 2 * *
0-5:
/home/oracle/snapvault/sv-dohot-daily.sh
# Weekly Snapshot copy/SnapVault at 2:00 a.m. every Saturday 0 2 * * 6:
/home/oracle/snapvault/sv-dohot-weekly.sh;
```

In step 4 above, there is a sample script for daily backups, “sv-dohot-daily.sh.” The hourly and weekly scripts are identical to the script used for daily backups, except the Snapshot copy name is different (sv_hourly and sv_weekly, respectively).

15.1.5 SnapManager for Oracle—Backup and recovery best practices

SnapManager for Oracle is a host and client-based product recently released for IBM System Storage IBM N series that integrates with Oracle9i R2 and Oracle Database 10g R2. It allows the DBA or Storage Administrator to manage your database backup, restore, recovery, and

cloning while maximizing your storage utilization. SnapManager for Oracle utilizes the N series product's Snapshot, SnapRestore, and FlexClone technologies while integrating with the latest Oracle releases. SnapManager automates and simplifies complex and manual time consuming processes typically done by Oracle DBAs that improve aggressive backup and recovery service level agreements (SLAs).

SnapManager for Oracle is protocol agnostic and thus works seamlessly when used with NFS and iSCSI protocols. It also integrates with native Oracle technologies, such as RAC, ASM and RMAN.

In order to utilize the SnapManager for Oracle product you need to have the following databases, applications, and licenses enabled:

- ▶ SnapManager for Oracle
- ▶ Red Hat Enterprise Linux
- ▶ SnapDrive for UNIX V2.1 or later
- ▶ IBM N series Data ONTAP 7.1 or later
- ▶ Oracle 9iR2 or Oracle 10g R2 using NFS or iSCSI LUNs
- ▶ IBM N series Host Agent 2.2.1 or later
- ▶ FlexClone license
- ▶ NFS or iSCSI license
- ▶ SnapRestore license

Note: IBM recommends that you use SnapManager for Oracle if you are using Redhat Enterprise Linux 3.0 Update 4 (RHEL 3.0 Update 4) and greater and you wish to solve mission-critical enterprise issues as they relate to Oracle backup, recovery, and cloning.

15.1.6 SnapManager for Oracle—Automatic Storage Management-based backup and restore

SnapManager for Oracle (Figure 15-8 on page 317) provides capabilities that enable seamless backups of Oracle ASM based-databases, while deployed on IBM System Storage N series. This allows customers that are running Oracle 10gR2 while using the Automatic Storage Management (ASM)-based databases on an iSCSI LUN to leverage the use of IBM System Storage N series' Snapshot and SnapRestore technology through the SnapManager for Oracle software. This is accomplished while maintaining the same flexibility and simplification that an Oracle ASM database was designed to achieve.

SnapManager for Oracle Central Management - Graphical User Interface (Windows)

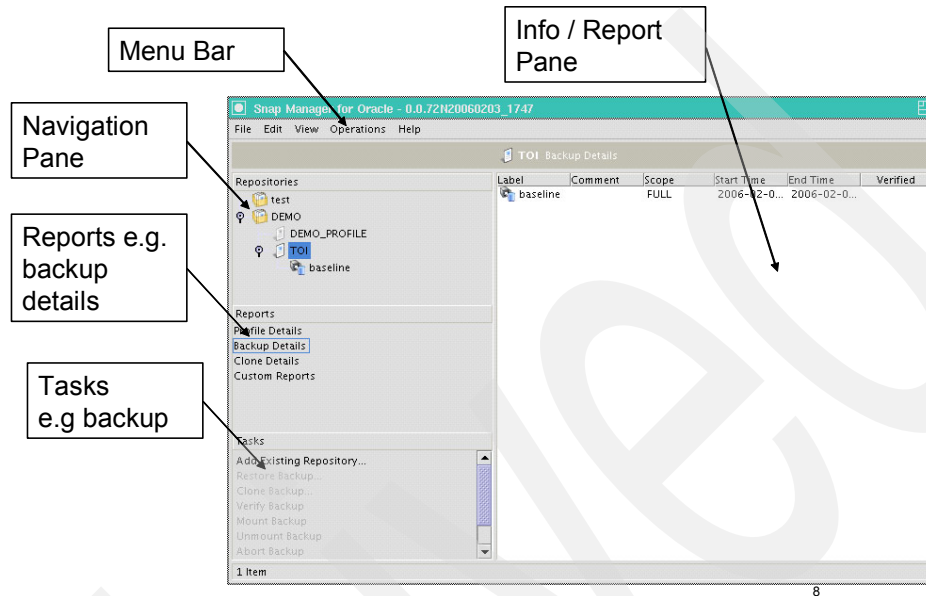


Figure 15-8 SnapManager for Oracle

Note: IBM requires that the ASMLib driver is used on Red Hat Linux Enterprise 3 when used with ASM and SnapManager for Oracle. The ASMLib driver is a dependency for SnapManager for Oracle and will not function without it.

15.1.7 SnapManager for Oracle—RMAN based backup and restore

SnapManager for Oracle provides integration with the Oracle RMAN architecture by allowing the functionality to register SnapManager based backups with the RMAN catalog. This allows the DBA to utilize the IBM System Storage N series' Snapshot and SnapRestore technology for database backups and recovery through the use of SnapManager, while still having access to the RMAN capabilities your DBA may have grown accustomed to using. Thus by allowing RMAN integration with the IBM System Storage N series' SnapManager product, the ability to do block level recovery using RMAN is not sacrificed.

Note: IBM requires that all datafiles, logfiles, and archive log files from the database to be backed up are stored on the IBM System Storage N series in a flexible volume for any backup or recovery to be completed.

15.1.8 SnapManager for Oracle—cloning

SnapManager for Oracle allows database cloning of existing Oracle 9iR2 and Oracle Database 10gR2. Database cloning is available when used with IBM System Storage N series' FlexClone technology, while running the SnapManager product (Figure 15-9). The database clone process is executed by providing both the old and new database SID name as well as a map file that allows the DBA or Storage Administrator to specify the new location of the files along with the new file names.

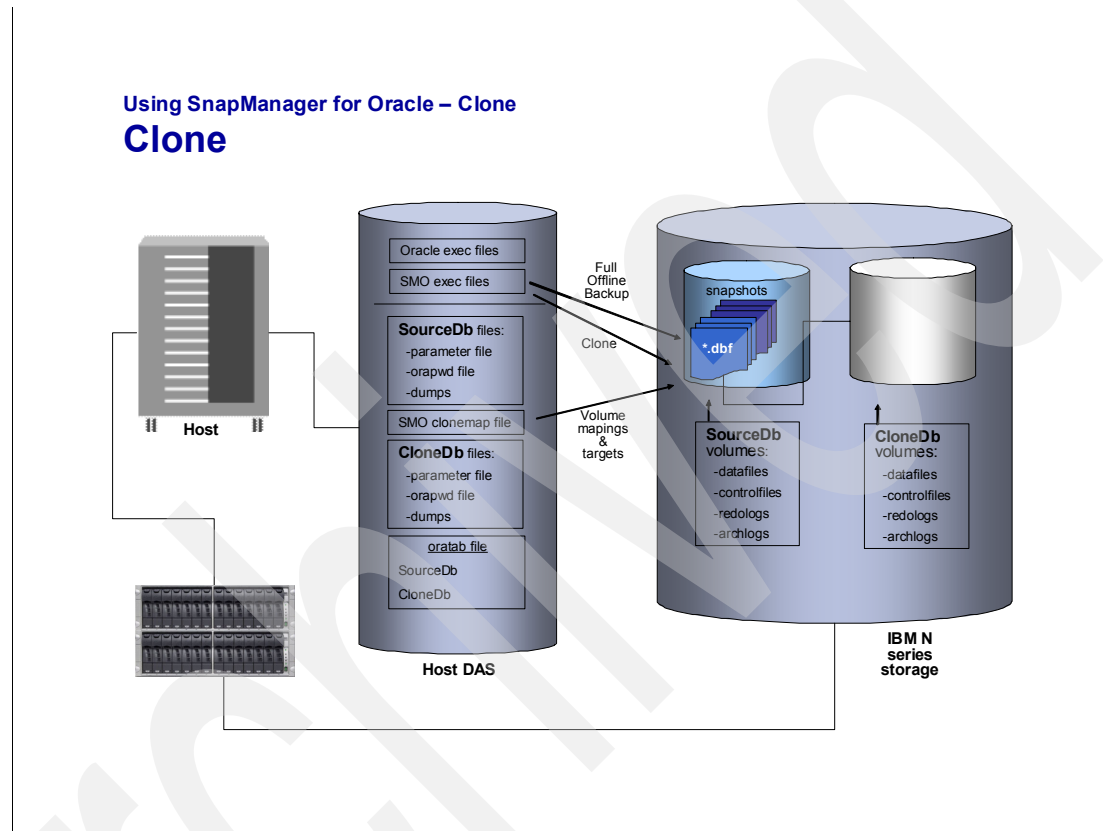


Figure 15-9 SnapManager for Oracle Clone

Note: IBM System Storage N series requires that a database clone is completed against a database backup that was taken when the database was in offline mode. Hot database cloning will be available in a future release of SnapManager for Oracle.



NFS performance tuning for N series and Oracle Database

This chapter discusses the operation of Relational Databases with IBM System Storage N series. Management implications and performance expectations for databases using IBM System Storage N series are presented in detail.

16.1 Introduction

Modern relational databases are growing at explosive rates. This database growth is accompanied by an increase in storage subsystem size and complexity, resulting in increased management challenges. These challenges can be effectively addressed using storage virtualization. The most common virtualization technique uses IBM System Storage N series and the Network File System (NFS). NFS is becoming a common and critical component in successful enterprise database deployments. NFS provides extremely effective storage virtualization, while utilizing existing infrastructure and management facilities.

This chapter provides a roadmap to successfully deploy a relational database management system (RDBMS) with NFS, based on the deployment of Red Hat Linux servers and IBM System Storage N series. Specific guidelines are provided for deploying Oracle on NFS along with tips and techniques for analyzing performance.

16.2 Focus on OLTP Workloads

Relational database workloads fall into two main categories: Online Transaction Processing (OLTP) and Decision Support Systems (DSS). OLTP environments consist of transactional workloads, for example order entry environments, and are critical for the day-to-day activities of many modern enterprises. DSS, also referred to as Data Mining or Data Warehouse systems, is typically used in an analytical context to execute what-if scenarios, perform trend analysis, and derive statistical information from transactional data. Commercial workloads typically consist of a mixture of OLTP and DSS workloads. A common scenario is one where OLTP is the dominant activity during business hours and DSS workloads prevail during non-business hours.

This chapter focuses on OLTP as a first step in migrating traditional database environments to IBM System Storage N series. OLTP workloads are typically more prevalent; thus, reducing the cost and complexity provides a more significant win. Although many of the concepts and recommendations discussed are also applicable to DSS environments.

16.3 Deploying databases with IBM System Storage N series and NFS

This section provides background on the technologies and concepts central to deploying databases on NAS and NFS including a discussion of the tradeoffs between NAS and SAN, the benefits of NFS in traditional applications, and the differences between database I/O versus that seen in traditional applications.

16.3.1 Network Attached Storage versus Storage Area Networks

There are two main storage connection paradigms: NAS and SAN. Network Attached Storage (NAS) is data storage that is connected to traditional networks such as Ethernet networks and accessible to clients on those networks via standard protocols. Such networks typically rely on data transmission protocols, such as TCP/IP, which have been widely used for many years. The most common NAS solutions use the Network File System (NFS). NFS provides network clients with a file-level interface to the data store.

Storage Area Networks (SANs) also provide network access to data storage, but with a notably different interconnect paradigm. Where NAS uses traditional networks, SANs relies on a set of storage-specific transport protocols: Small Computer System Interface (SCSI) and Fibre Channel Protocol (FCP). SANs provides a block-level interface to stored data.

The first and most obvious difference between the two storage paradigms is the interconnect used. SAN uses Fibre Channel and NAS uses Ethernet. This is an important distinction since, as shown in Figure 2-1 on page 32, Ethernet bandwidth is eclipsing Fibre Channel. In addition, most companies also have significant in-house expertise with Ethernet, simplifying NAS deployment and ongoing network management.

A second and more subtle difference between SAN and NAS (as typified by NFS) is that while the actual data storage is contained in a storage array in both paradigms, there is a significant difference in the software that runs on the host system. With NFS the only software component is the network transport stack, since the file system and volume management layers are run on the back end storage system. NFS provides file access to applications without the burden of additional system software.

With SAN, the entire file system code stack must run on the host. In addition, a Volume Manager is often needed to help manage the vast amount of storage. This software plus required data management tasks (backup, recovery, and so on) put an additional burden on the host system. Viewed as a whole, therefore, NAS offers significant advantages over SAN for database deployment.

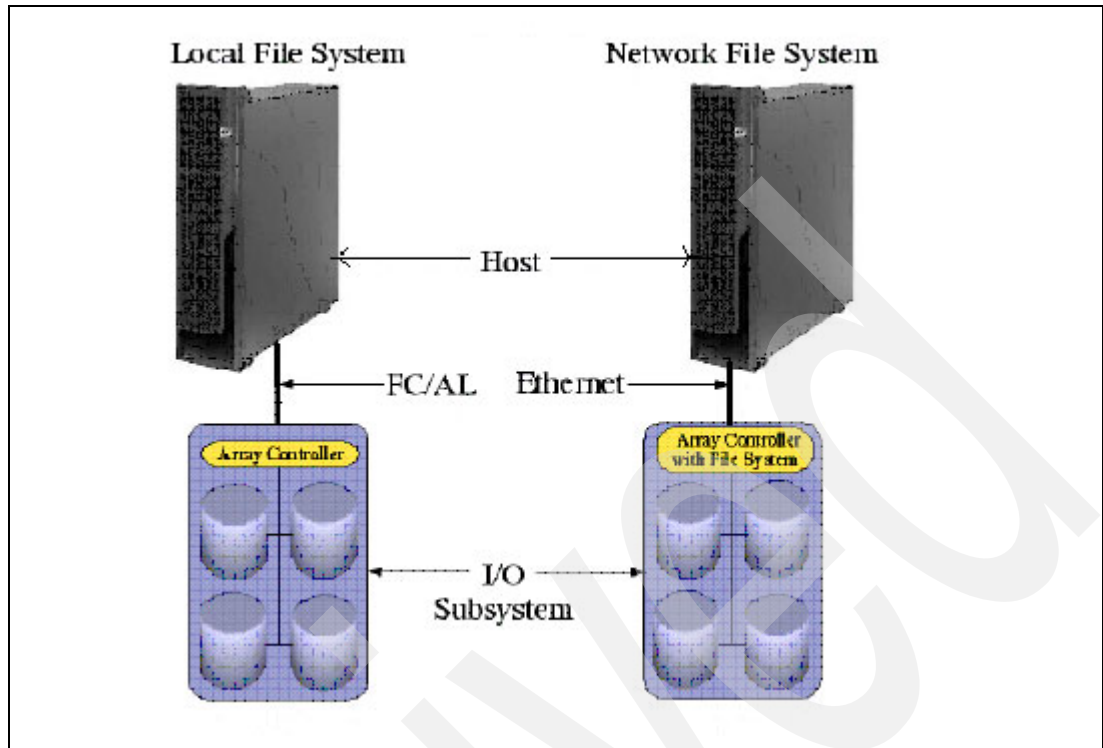


Figure 16-1 Comparison of a Local File System versus a Network File System

16.3.2 NAS Protocols

In addition to NFS, there are a number of other protocols that fall into the NAS category. Internet SCSI (iSCSI) and the Common Internet File System (CIFS) are the most common or widely talked about. The common denominator among these protocols is the use of Ethernet as the data transport layer. CIFS, which is predominantly used in Microsoft Windows environments, is not commonly found in database environments because CIFS lacks the appropriate file locking semantics required by databases. Internet SCSI offers a block-oriented protocol for Ethernet that is in some ways similar to FCP but with substantial cost advantages. This section focuses on NFS, and these alternative protocols are not discussed further.

16.3.3 Why NFS

A successful solution for database storage has several important properties:

- ▶ Storage virtualization: Storage configuration, management, and access must be handled almost completely by the storage system.
- ▶ Complexity and cost reduction: Controlling the cost and complexity of deploying and managing a storage solution is a primary concern in most IT environments.
- ▶ Rapid application deployment: Rapid deployment and simplified support for a wide variety of applications are essential.
- ▶ Grid support: There is a clear trend toward grid-aware applications that can take advantage of emerging server blade environments, such as Oracle10g RAC. A successful storage solution must accommodate current and future grid deployments.

NFS fulfills these requirements and provides a number of additional benefits over traditional storage access models. NFS not only simplifies data transport but also reduces the management complexity of large storage installations. NFS also offloads the physical I/O processing from the host system to NAS. This can be a real benefit over systems that require the host system to consume CPU cycles to handle all I/O functionality. This offloading can translate into more efficient resource usage.

16.3.4 Database I/O Patterns

Database I/O patterns differ from those seen in traditional NFS environments. Databases stress different aspects of both the NFS client and the NFS server. The principal differences lie in the following areas:

- Data Caching Mechanisms
- Data Integrity Model
- Asynchronous I/O (AIO)
- I/O pattern

Non-database applications rely on the underlying file system to cache I/O to increase performance. The file system caches data used by the application and re-uses the data on subsequent I/O requests to reduce the number of disk transactions. Databases, however, have their own internal caching mechanisms. This often leads to double caching (caching of the same data by both the file system and the application cache) and potentially negative interactions between the two caches, resulting in sub-optimal performance if not corrected.

Another significant difference between databases and other classes of applications is the data integrity model for I/O write operations. Non-database applications often allow the file system to defer writing data to disk until a later point-in-time determined by the operating system. This is often referred to as asynchronous write-back, or just write-back. Write-back reduces the number of disk transactions and thus increases I/O efficiency and performance. However, databases require that an I/O request be immediately written to disk to ensure data integrity. This increases the number of I/Os required and creates a serialized latency that non-database applications typically do not experience.

Asynchronous I/O (AIO) is a framework supplied by the operating system that allows the calling application to continue with other data processing while an I/O request is serviced in the background. Databases make extensive use of AIO to pipeline I/O requests to reduce CPU consumption and to provide maximum performance. AIO is especially useful for applications that control I/O read-ahead (pre-fetching of data that will be processed later) and write-back characteristics. Databases have an intimate knowledge of their internal I/O architecture and can effectively control data read-ahead and write-back behavior. Other applications typically do not use AIO due to the added complexity required to create an appropriate I/O architecture within the application. Non-database applications leave the read-ahead and write-back functionality to the file system layer.

The final major distinction between database environments and other application environments relates to the I/O access pattern. OLTP environments generate an I/O pattern of small, highly parallel, randomly distributed requests. Concurrent I/O (often to the same file) is vital to successful OLTP database operation.

The asymmetry between database I/O patterns and that of the other classes of NFS applications means NFS performance improvement efforts in the past have mostly neglected database issues. With NFS playing an increasing role in database deployments, there is a compelling need to better characterize I/O performance for these environments.

16.4 Tuning a Linux, N series product, Oracle environment

Configuring a complete NAS environment for a relational database is a reasonably straightforward undertaking. This section provides guidelines for configuring and tuning Oracle10g, Red Hat Linux, the N series product, and Gigabit Ethernet networks for best results. These specific guidelines can also apply in a general way to other similar solutions.

16.4.1 Oracle tuning

This section provides specific tuning considerations and configuration recommendations that affect Oracle/NFS performance. The recommendations deal specifically with Oracle10g; however, the principles apply to any RDBMS.

There are three main areas to consider:

- Software architecture: 64-bit versus 32-bit
- Direct I/O versus Cached I/O
- Asynchronous I/O

Each of these areas is discussed in detail.

Software architecture: 64-bit versus 32-bit Oracle

Oracle comes in two basic architectures: 64-bit and 32-bit. The number of address bits determines the maximum size of the virtual address space:

32-bits = 2^{32} = 4 GB maximum

64-bits = 2^{64} = 16777216 TB maximum

Database performance is highly dependent on available memory. In general, more memory increases caching, which reduces physical I/O to the disks. Hence, the 32-bit versus 64-bit decision is very important. For instance, Oracle's Shared Global Area (SGA) is a memory region in the application that caches database tables and other data for processing. With 32-bit software the SGA is limited to 4 GB.

Many customer databases are quite large, on the order of hundreds of gigabytes or even multiple terabytes. As a result, the database working set size is much larger than the 32-bit limit. As the ratio of working set size to SGA size grows, the probability of finding a data block in memory decreases, causing more I/O demand on the system and decreasing performance.

With 64-bit databases, the SGA can be sized to fit the majority of the physical memory available in the system. This decreases the ratio of working set size to SGA size and therefore improves performance. Larger SGAs minimize physical I/O and maximize transaction rates. Additionally, databases can manage the SGA more effectively than the OS manages the file system cache. In general, a 64-bit database should be given the majority of physical memory.

Unfortunately, many database applications still require 32-bit database software primarily due to application qualification and support. In this case, the system contains more physical memory than the SGA can utilize. In this situation the file system cache can be used in addition to the SGA, a technique known as *super-caching*. With super-caching, database blocks are cached not only in the SGA, but additional database blocks that do not fit inside SGA are cached in the file system buffer cache. This allows the database to fully utilize the system memory for database processing rather than leaving large portions of memory unused.

Super-caching in general helps 32-bit database environments but at some cost to optimal performance as the file system is now consuming additional CPU cycles for processing. For 64-bit environments, super-caching is not advised as the file system cache will typically prevent the system from achieving the optimal transaction rate.

For database consolidation environments in which multiple databases execute on the same system, the SGA size per database instance must be limited since the database instances share system memory. Super-caching can help in such a scenario.

Direct I/O versus OS Cached I/O

DirectIO is a mechanism available in modern file systems that delivers data directly to an application without caching the data in the file system buffer cache. This provides raw device-like non-caching behavior while preserving all other file system semantics. Avoiding the file system cache can provide a reduction in kernel code path execution lengths, which reduces CPU cycles per I/O. DirectIO also allows I/Os of arbitrary size rather than limiting each I/O to a page size multiple. Enabling DirectIO on NFS translates to an over-the-wire NFS operation for every I/O request. Additionally, I/O requests are passed directly to the network stack, bypassing some code layers. This behavior can be very beneficial for Oracle Log Writer performance, both in terms of throughput and latency. Typically the database application is responsible for enabling DirectIO. See Figure 16-2.

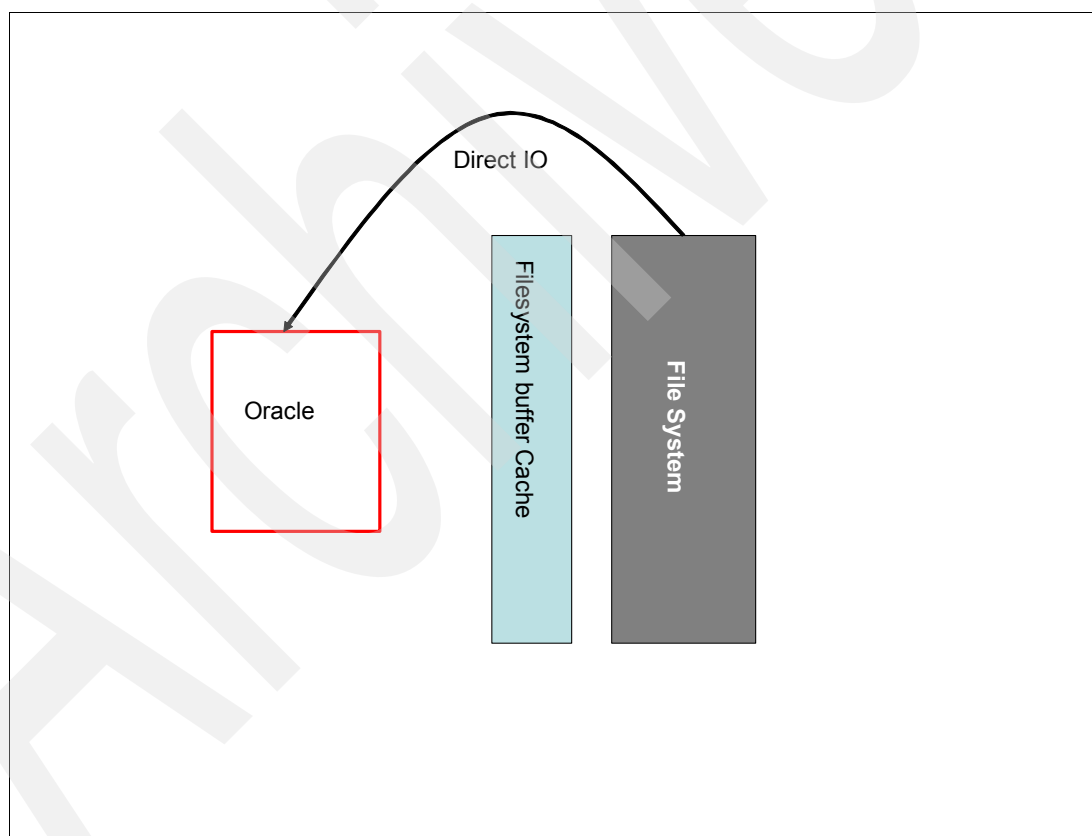


Figure 16-2 Direct IO

Enabling Direct I/O

Use the following procedure to enable uncached I/O:

1. Become root.
2. Edit /etc/modules.conf.

3. Add the following line anywhere:

```
options nfs nfs_uncached_io=1
```

4. § Set the filesystem io_options parameter in the oracle initialization parameter file to DIRECTIO (filesystemio_options = DIRECTIO).§ If the asynchronous I/O option is in use, then parameter filesystemio_options in the initialization parameter file should be set to SETALL.

Uncached I/O will take effect after you reboot your client. Only mount points that use the “noac” mount option will be effected by this change.

Note: Direct I/O support is not available and is not supported on Red Hat Enterprise Linux 2.1 and United Linux. It is available and is supported on Red Hat Enterprise Linux 3 and higher also over NFS, if the driver being used on the system supports varyio XE "varyio" .

Asynchronous I/O

With synchronous I/O, a process issues an I/O request and waits for it to complete before proceeding. Asynchronous I/O (AIO) however is less intuitive. With AIO a process issues an I/O request and, without waiting for it to complete, continues execution. At some point in the future (and through an alternative mechanism) the process is notified of the I/O completion. The AIO model fits well with databases. A list of I/Os can be submitted to the file system while the database proceeds with other useful work. When the database is ready to verify that its I/Os have completed, the appropriate mechanism is invoked.

The Oracle Log Writer and other Database Writer processes all use AIO. The database issues a large list of I/O requests (both random and sequential) rather than accessing one block at a time

Enabling Asynchronous I/O

Use the following steps to enable AIO

1. Verify that libaio and libaio-devel are installed in your Linux system.

Example 16-1 Verifying installation of libaio and libaio-devel on Linux

```
libaio-0.3.96-5.i386.rpm  
libaio-devel-0.3.96-5.i386.rpm
```

2. By default, asynchronous I/O (AIO) is disabled in Oracle. To use AIO with Oracle Databases, the Oracle binary must be relinked with AIO enabled, using the following commands:

```
cd $ORACLE_HOME/rdbms/lib  
make PL_ORALIBS=-laio -f ins_rdbms.mk asynch_on
```

These commands must be executed while logged in as the Oracle user with the instance shut down.

3. Set the following Oracle parameters in the Oracle initialization parameter file and start the instance and database:

```
disk_asynch_io=true  
filesystemio_options=asynch
```

16.5 Linux NFS client performance

If you use Linux NFS clients and the N series product together on an unrouted network, consider using jumbo frames to improve the performance of your application. Be sure to consult your switch's command reference to make sure it is capable of handling jumbo frames in your environment. If you experience unexpected performance slow downs when using jumbo frames, try reducing the MTU to, say, 8,960 bytes.

When using jumbo frames on more complex networks, ensure that every link in the network between your client and server support them and have the support enabled. If NFS over TCP is working with jumbo frames, but NFS over UDP is not, that may be a sign that some part of your network does not support jumbo frames.

16.5.1 Identifying Kernel Releases

IBM System Storage N series recommends that its Linux customers always use the latest actively maintained distributions available.

To obtain which kernel your clients run, you can issue the following command:

```
% uname -r
```

16.5.2 Choosing a Network Transport Protocol

We strongly recommend that you use TCP as the transport of choice for NFS on modern Linux distributions. To avoid IP fragmentation issues on both the client and the N series product, consider disabling NFS over UDP entirely on the N series product and explicitly specifying "tcp" on all your NFS mounts. In addition, we strongly recommend the explicit use of the "timeo=600" mount option on Linux to work around bugs in the mount command, which shorten the retransmit time out. If you must use UDP for NFS, ensure that you have properly sized the transport socket buffers, and explicitly set a large number of retransmissions with the "retrans=" mount option.

16.5.3 Linux—Kernel Patches

The uncached I/O patch was introduced in Red Hat Advanced Server 2.1, update 3, with kernel errata e35 and up. It is mandatory to use uncached I/O when running Oracle9i RAC with the N series product in an NAS environment. Uncached I/O does not cache data in the Linux file system buffer cache during read/write operations for volumes mounted with the **noac** mount option.

To enable uncached I/O, add the following entry to the /etc/modules.conf file, and reboot the cluster nodes:

```
options nfs nfs_uncached_io=1
```

The volumes used for storing Oracle Database files should still be mounted with the **noac** mount option for Oracle9i RAC databases. The uncached I/O patch was developed by Red Hat and tested by Oracle, IBM System Storage N series, and Red Hat.

16.5.4 Enlarging a client's Transport Socket Buffers

Enlarging the transport socket buffers that Linux uses for NFS traffic helps reduce resource contention on the client, reduces performance variance and improves maximum data and

operation throughput. In future releases of the client, the following procedure will not be necessary, as the client will automatically choose an optimal socket buffer size.

```
Become root on the client
cd into /proc/sys/net/core
echo 262143 > rmem_max
echo 262143 > wmem_max
echo 262143 > rmem_default
echo 262143 > wmem_default
Remount the NFS file systems on the client
```

This is especially useful for NFS over UDP and when using Gigabit Ethernet. Consider adding this to a system startup script that runs before the system mounts NFS file systems. **The recommended size (262,143 bytes) is the largest safe socket buffer size tested on Data ONTAP.** On clients with 16 MB of memory or less, leave the default socket buffer size setting to conserve memory.

Red Hat distributions after 7.2 contain a file called `/etc/sysctl.conf` where changes such as this can be added so they will be executed after every system reboot. Add the following lines to the `/etc/sysctl.conf` file on these Red Hat systems:

```
net.core.rmem_max = 262143
net.core.wmem_max = 262143
net.core.rmem_default = 262143
net.core.wmem_default = 262143
```

16.5.5 Other TCP enhancements

The following settings can help reduce the amount of work that clients and IBM System Storage N series do when running NFS over TCP:

```
echo 0 > /proc/sys/net/ipv4/tcp_sack
echo 0 > /proc/sys/net/ipv4/tcp_timestamps
```

These operations disable optional features of TCP to save a little processing time and network bandwidth. When building kernels, be sure that `CONFIG_SYNCOOKIES` is disabled. SYN cookies slow down TCP connections by adding extra processing on both ends of the socket. Some Linux distributors provide kernels with SYN cookies enabled.

Linux 2.2 and 2.4 kernels support large TCP windows (RFC 1323) by default. No modification is required to enable large TCP windows.

16.5.6 Linux networking—Gigabit Ethernet Network Adapters

If Linux servers are using high-performance networking (gigabit or faster), provide enough CPU and memory bandwidth to handle the interrupt and data rate. The NFS client software and the gigabit driver reduce the resources available to the application, so make sure resources are adequate. Most gigabit cards that support 64-bit PCI or better should provide good performance.

Note: Any database using N series should utilize Gigabit Ethernet on both the N series and database server to achieve optimal performance.

16.5.7 Linux networking—jumbo frames with GbE

Using jumbo frames can improve performance in environments where Linux NFS clients and IBM System Storage N series are together on an unrouted network. Be sure to consult the command reference for each switch to make sure it is capable of handling jumbo frames.

16.5.8 Special mount options

Consider using the **bg** option if your client system needs to be available even if it cannot mount some servers. This option causes mount requests to put themselves in the background automatically if a mount cannot complete immediately. When a client starts up and a server is not available, the client waits for the server to become available by default. The default behavior, which you can adjust with the **retry** mount option, results in waiting for almost a week before giving up.

The **fg** option is useful when you need to serialize your mount requests during system initialization. For example, you probably want the system to wait for /usr to become available before proceeding with multiuser boot. If you mount /usr or other critical file systems from an NFS server, you should consider using fg for these mounts. The **retry** mount option has no effect on foreground mounts. A foreground mount request will fail immediately without any retransmission if any problem occurs.

For security, you can also use the **nosuid** mount option. This causes the client to disable the special bits on files and directories. The Linux main page for the mount command recommends disabling or removing the **suidperl** command when using this option. Note that the IBM System Storage N series also has a **nosuid** export option, which does roughly the same thing for all clients accessing an export. Interestingly, the IBM System Storage N series' **nosuid** export option also disables the creation of special devices. If you notice programs that use special sockets and devices (such as "panel") behaving strangely, check for the **nosuid** export option on your N series product.

To enable Kerberos authentication on your NFS mounts, you can specify the **sec=krb** mount option. In addition to Kerberos authentication, you can also choose to enable authentication with request integrity checking (**sec=krb5i**), or authentication with privacy (**sec=krb5p**). Note that most Linux distributions do not yet support krb5p or other advanced security flavors such as SPKM3 or lipkey as of the current writing.

For some servers or applications, it is necessary to prevent the Linux NFS client from sending Network Lock Manager requests. You can use the **no1ock** mount option to prevent the Linux NFS client from notifying the server's lock manager when an application locks a file. Note, however, that the client still flushes its data cache and uses more restrictive write back semantics when a file lock is in effect. The client always flushes all pending writes whenever an application locks or unlocks a file.

Oracle and OS version	NFS mount options
Oracle 9i non-RAC Solaris	rw,bg,hard,intr,rsize=32768, wsize=32768,[forcedirectio or llock],tcp,vers=3
Oracle 10g non- RAC - Solaris	rw,bg,hard,intr,rsize=32768, wsize=32768,[forcedirectio or llock],tcp,vers=3
Oracle 9i non-RAC RHAS 2.1	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600;
Oracle 10g non-RAC - RHAS 2.1	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600;
Oracle 10g non-RAC - RHEL 3.0	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600
Oracle 10g non-RAC - RHEL 4.0	Oracle parameter filesystemio_options=directio rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600
Oracle 9i non-RAC - SLES 8/9	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600
Oracle 10g non-RAC - SLES 8/9	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600

Figure 16-3 Special mount options

16.5.9 IBM System Storage N series tuning

This section focuses on general database/file layout and specific N series product configurations. The general layout recommendations are intended as guidelines that may vary depending on the deployment environment. The specific N series product settings are recommended to ensure optimal performance from the storage subsystem.

There are several areas of consideration for organizing the database storage on the IBM N series:

- Volume layout
- Placement of Oracle home directory
- Placement of Oracle data files
- Placement of Oracle log files

16.5.10 Volume layout

A *volume* is a virtual unit of storage in an IBM System Storage N series. A volume consists of one or more *raid groups*, which in turn consist of multiple physical disk drives. See Figure 16-4 on page 331. In terms of Oracle Database deployment, a volume has two important properties:

- ▶ **Disk performance load distribution:** All disks in a volume share equally in the I/O load. Volumes with more disks will have greater I/O performance capacity than volumes with a smaller number of disks. All files in a volume benefit from each disk's throughput capability, providing automatic load balancing and maximum throughput across the disks. In a database deployment where certain data sets are more frequently used than others, but the active data set changes over time, this load balancing property ensures good performance. For this reason, the recommendation is to use fewer volumes with more disks per volume when feasible.
- ▶ **Snapshot and SnapRestore:** In a database deployment, Snapshot copies are typically created on a per volume basis. For these reasons, database logs needed for failure recovery are typically stored on a volume separate from database files. With this arrangement, database corruption can be resolved by first doing a snaprestore" on the database volume, and then applying the rollback logs. A database can typically be

returned to operation in a matter of minutes using this technique, versus the hours or days typically needed to restore from tape.

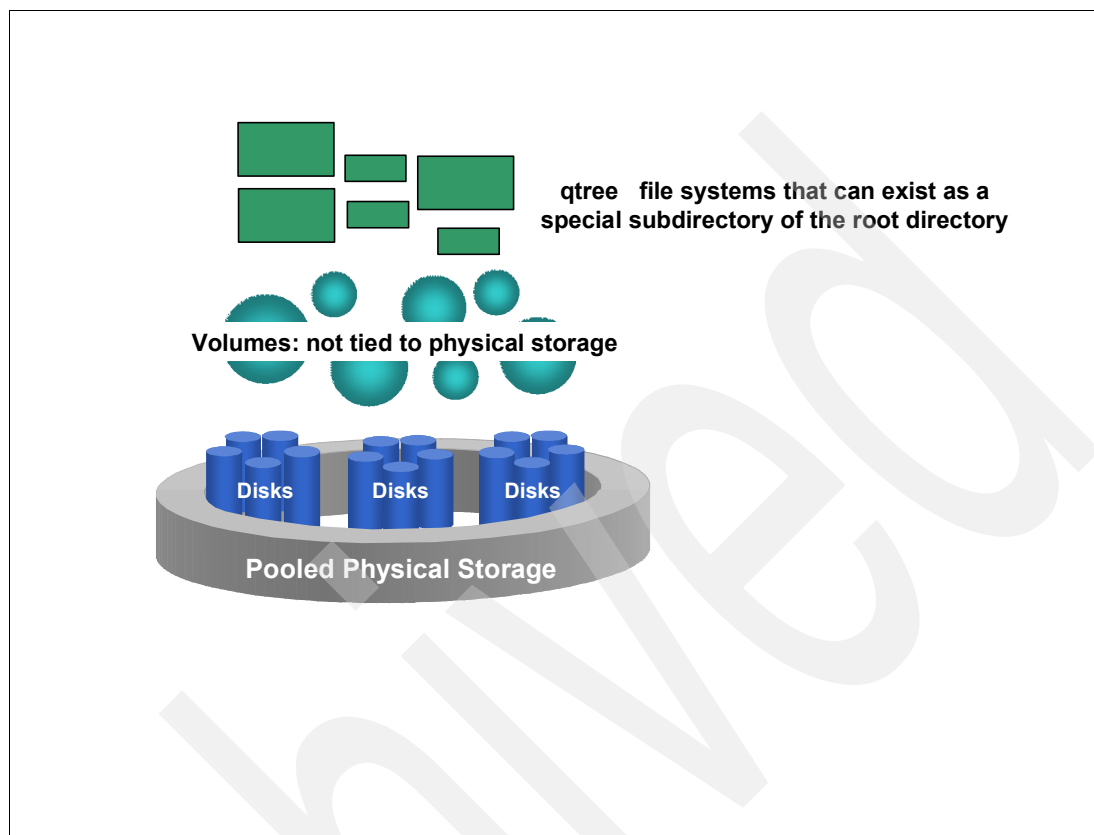


Figure 16-4 IBM System Storage N series volumes

16.5.11 Placement of Oracle home directory

The Oracle home directory (typically /export/home/oracle) can be located either on the N series product or on storage attached locally to the database server. The recommendation is to store Oracle Home on the IBM System Storage N series. However, several issues must be considered and managed with such a configuration.

Placing Oracle Home on an IBM System Storage N series has the following advantages:

- ▶ Easy to configure a manual fail-over server.
- ▶ Easy to set up multiple versions of Oracle on the same server, and then switch between them. This is ideal in a lab environment, where multiple versions of Oracle must be tested.
- ▶ The ability to create a Snapshot copy of Oracle Home before installing patches or upgrades ensures a safe, quick recovery if the procedure fails or has to be backed out.

Placing Oracle Home on an IBM System Storage N series creates the following issues:

- ▶ Creates an additional point of failure for the Oracle installation. However, since Oracle Database files are stored on the IBM System Storage N series, this is not significant since loss of the N series product would temporarily bring down the database anyway.
- ▶ In the event of an IBM System Storage N series outage, the error logs that Oracle normally keeps in the "\${ORACLE_HOME}/rdbms/logs" directory will not be accessible. Thus, there is a chance that information required to diagnose a problem might not be

available. For this reason, these files should be relocated onto local storage or onto a separate N series products.

Note: Oracle binaries cannot be currently executed via a forcedirectio mount point. A separate mount point is required. Future revisions of the Solaris OS will remove this restriction.

16.5.12 Placement of Oracle data and log Files

There are many advantages to storing Oracle data files on an IBM System Storage N series:

- ▶ Data availability is improved. For example, the loss of a disk has no impact to the host.
- ▶ Management of the storage space is simplified through the ability to create Snapshot copies, easy capacity expansion, and effective backup strategies.
- ▶ Performance demands are automatically balanced across all the disks in a volume.
- ▶ The shared file system paradigm enables access to the data files from more than one host.

If Archive Logging is enabled as part of the operation of the Oracle Database, it is a good idea to place the log data files in a separate file system. Separation of archive logs from data files provides better protection in case of database corruption. If the volumes containing the data files are damaged or corrupted, the most recent backup can be restored and a point-in-time recovery using the Archive Logs can be performed.

Having the Redo Logs in a separate volume offers an additional level of protection. Placing Redo Logs in a separate volume also simplifies management of these files for backup and recovery purposes. Because the Redo Logs contain a continuous stream of update records, they will often grow at a faster pace than the rest of the data files. As a result, a more frequent backup schedule may be required to ensure adequate data retention and subsequent cleanup of the oldest archive log files. By placing the logs in a separate volume, creating Snapshot copies or integration with backup tools is simplified.

Finally, if the log volume fills to capacity, there is a chance that the database will stop processing. By placing the logs in a separate volume, monitoring disk space usage is simplified because the total space is allocated to a single purpose. Procedures can be created for emergency handling of near full situations. These procedures can be automated by integrating with a host-based backup tool.

16.5.13 NFS settings

There are only a few IBM System Storage N series specific NFS variables. Verify that the following IBM System Storage N series settings by typing options **nfs.tcp** on the IBM System Storage N series console.

```
nfs.tcp.enable on
nfs.tcp.recvwindowsize 65536
nfs.tcp.xfersize 65536
```

The first option verifies that the N series product will accept TCP connections. The next two options set the “receive window size” and “NFS transfer size” to the maximum values.

16.5.14 Volume options

There are several per volume options that affect database performance. These include the use of Snapshot copies, pre-fetching, and NFS file attribute updates.

Automatic Snapshot copies should be turned off on an N series product used to store an Oracle Database. To do this, issue the following command for each volume on which Oracle data is stored:

```
vol options <volname> nosnap on
```

To make the .snapshot directory invisible to the client, issue the following command for each volume on which Oracle data is stored:

```
vol options <volname> nosnapdir on
```

In order for Snapshot copies to be effectively used with Oracle, they must be coordinated with the Oracle backup process. A complete description of backup and restore options can be found in Chapter 15, “Backup, restore, and disaster recovery of Oracle using N series” on page 307.

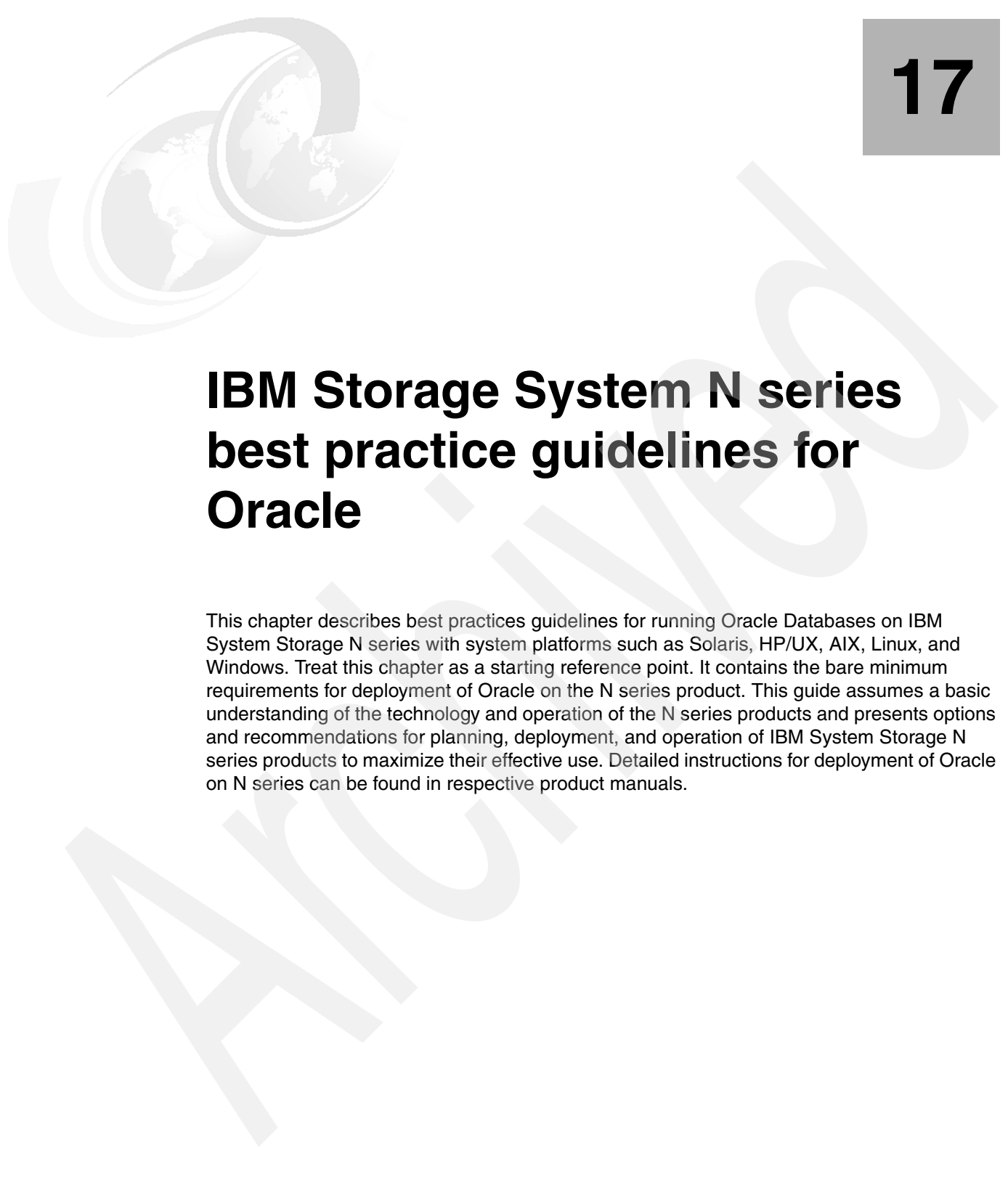
The option **minra** controls data pre-fetch or read ahead properties on a per volume basis. Minra is short for “minimize read ahead”. So setting **minra=on** indicates that minimal read ahead should be performed on the volume. OLTP database workloads are mostly random access, so pre-fetching is ineffective and can waste critical system resources. By default **minra** is off. To turn **minra** on, do the following:

```
vol options <volname> minra on
```

Additional IBM System Storage N series performance can be gained by disabling the update of access times on inodes. Consider turning this option on if your environment has heavy read traffic AND can tolerate “last access times” on files being incorrect. This is the case with databases that maintain their own internal file data and time information.

```
vol options <volname> no_atime_update on
```

Archived



IBM Storage System N series best practice guidelines for Oracle

This chapter describes best practices guidelines for running Oracle Databases on IBM System Storage N series with system platforms such as Solaris, HP/UX, AIX, Linux, and Windows. Treat this chapter as a starting reference point. It contains the bare minimum requirements for deployment of Oracle on the N series product. This guide assumes a basic understanding of the technology and operation of the N series products and presents options and recommendations for planning, deployment, and operation of IBM System Storage N series products to maximize their effective use. Detailed instructions for deployment of Oracle on N series can be found in respective product manuals.

17.1 IBM System Storage N series system configuration

This section covers network settings with regards to N series and databases.

17.1.1 IBM System Storage N series settings

When configuring network interfaces for new systems, it is best to run the set up command to automatically bring up the interfaces and to update the `/etc/rc` file and `/etc/hosts` file. The set up command requires a reboot to take effect. However, if a system is in production and cannot be rebooted, network interfaces can be configured with the `ifconfig` command. If a NIC is currently online and needs to be reconfigured, it must first be brought down. To minimize downtime on that interface, a series of commands can be entered on a single command line separated by the semicolon (;) symbol.

Example 17-1 ifconfig

```
nseries>ifconfig e0 down;ifconfig e0 'hostname'-e0 mediatype auto netmask  
255.255.255.0 partner e0
```

Note: When configuring or reconfiguring NICs or VIFs in a cluster, it is imperative to include the appropriate partner <interface> name or VIF name in the configuration of the cluster partner's NIC or VIF to ensure fault tolerance in the event of cluster takeover. Please consult your IBM System Storage N series support representative for assistance. A NIC or VIF being used by a database should not be reconfigured while the database is active. Doing so can result in a database crash.

17.1.2 Ethernet—Gigabit Ethernet, auto negotiation, and Full Duplex

Any database using IBM System Storage N series should utilize Gigabit Ethernet on both the N series product and the database server.

IBM System Storage N series Gigabit II, III, and IV cards are designed to auto negotiate interface configurations and can intelligently self-configure themselves if the auto negotiation process fails.

Note: IBM System Storage N series recommends that Gigabit Ethernet links on clients, switches, and IBM System Storage N series systems be left in their default auto negotiation state unless no link is established, performance is poor, or other conditions arise that might warrant further troubleshooting.

By default, flow control should be set to “full” on the IBM System Storage N series in its `/etc/rc` file. You can do this by including the entry in Example 17-2 (assuming the Ethernet interface is `e5`):

Example 17-2 Flow control

```
ifconfig e5 flowcontrol full
```

If the output of the `ifstat -a` command does not show full flow control, then the switch port will also have to be configured to support it. (The `ifconfig` command on the N series product

will always show the requested setting. `ifstat` shows what flow control was actually negotiated with the switch.)

17.2 Volume, aggregate setup and options

In this section we cover set up parameters and options for volume and aggregates as they relate to databases.

17.2.1 Databases

There is currently no empirical data to suggest that splitting a database into multiple physical volumes enhances or degrades performance. Therefore, the decision on how to structure the volumes used to store a database should be driven by backup, restore, and mirroring requirements. A single database instance should not be hosted on multiple unclustered IBM System Storage N series because a database with sections on multiple N series products makes maintenance that requires IBM System Storage N series downtime even for short periods hard to schedule and increases the impact of downtime. If a single database instance must be spread across several separate N series for performance, care should be taken during planning so that the impact of IBM System Storage N series maintenance or backup can be minimized. Segmenting the database so the portions on a specific N series can periodically be taken offline is recommended whenever feasible.

17.2.2 Aggregates and FlexVol volumes or traditional volumes

IBM System Storage N series supports pooling of a large number of disks into an aggregate, and then building virtual volumes (FlexVol volumes) on top of those disks.

For Oracle Databases we recommend that you pool all your disks into a single large aggregate (see Figure 17-1), and use FlexVol volumes for your database datafiles and logfiles. This provides the benefit of much simpler administration, particularly for growing and reducing volume sizes without affecting performance.

Note: In the following few sections, when we refer to volume size, it can be either traditional volumes or Flex™ Volumes.

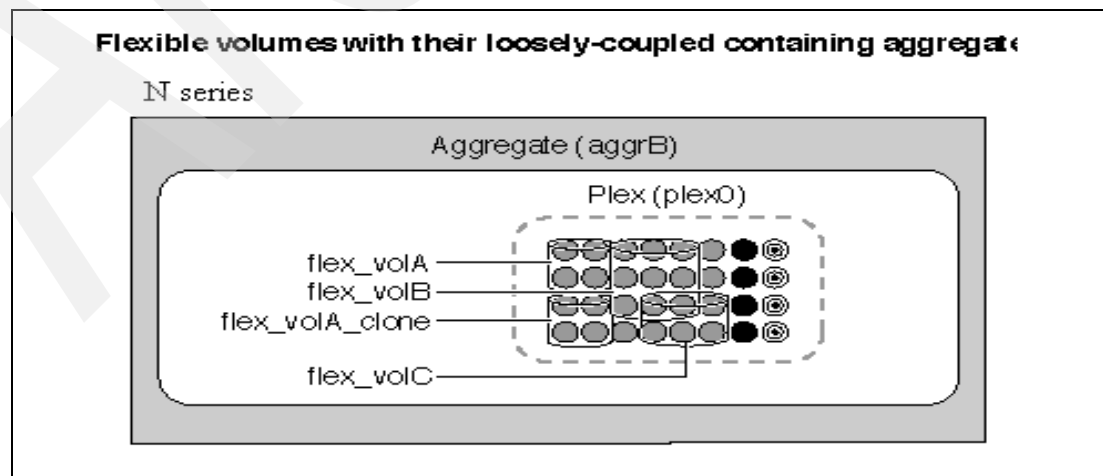


Figure 17-1 Aggregate with volumes

17.2.3 Volume Size

While the maximum supported volume size on an IBM System Storage N series system is 16TB, the N series product discourages customers from configuring individual volumes larger than 3TB.

IBM System Storage N series recommends that the size of a volume be limited to 3TB or smaller for the following reasons:

- ▶ Reduced per volume backup time
- ▶ Individual grouping of Snapshot copies, qtrees, and so on.
- ▶ Improved security and manageability through data separation.
- ▶ Reduced risk from administrative mistakes, hardware failures, and so on.

17.2.4 Recommended volumes for Oracle Database files and logfiles

Based on testing, we found the layouts in Table 17-1 adequate for most scenarios. The general recommendation is to have a single aggregate containing all the flexible volumes containing database components.

Table 17-1 Volume recommendations

Database binaries	Dedicated FlexVol volume	
Database config files	Dedicated FlexVol volume	Multiplex with transaction logs
Transaction log files	Dedicated FlexVol volume	Multiplex with config files
Archive logs	Dedicated FlexVol volume	Use SnapMirror
Data files	Dedicated FlexVol volume	
Temporary datafiles	Dedicated FlexVol volume	Do not make Snapshot copies of this volume
Cluster related files	Dedicated FlexVol volume	

17.2.5 Oracle Optimal Flexible Architecture (OFA) on the N series product

Distribute files on multiple volumes on physically separate disks to achieve I/O load balancing:

- ▶ Separate high I/O Oracle files from system files for better response times.
- ▶ Ease backup and recovery for Oracle data and log files by putting them in separate logical volumes.
- ▶ Ensure fast recovery from a crash to minimize downtime.
- ▶ Maintain logical separation of Oracle components to ease maintenance and administration.
- ▶ OFA architecture works well with a multiple Oracle home (MOH) layout (Figure 17-2 on page 339).

For more information about Oracle OFA for RAC or Non-RAC and Oracle9i versus Oracle10g, visit the following links:

- OFA for Non-RAC:

http://download-west.oracle.com/docs/html/B14399_01/app_ofa.htm#i633126

For RAC, OFA for ORACLE_HOME changes:

http://download-west.oracle.com/docs/cd/B19306_01/install.102/b14203/apa.htm#CHDCDGFE

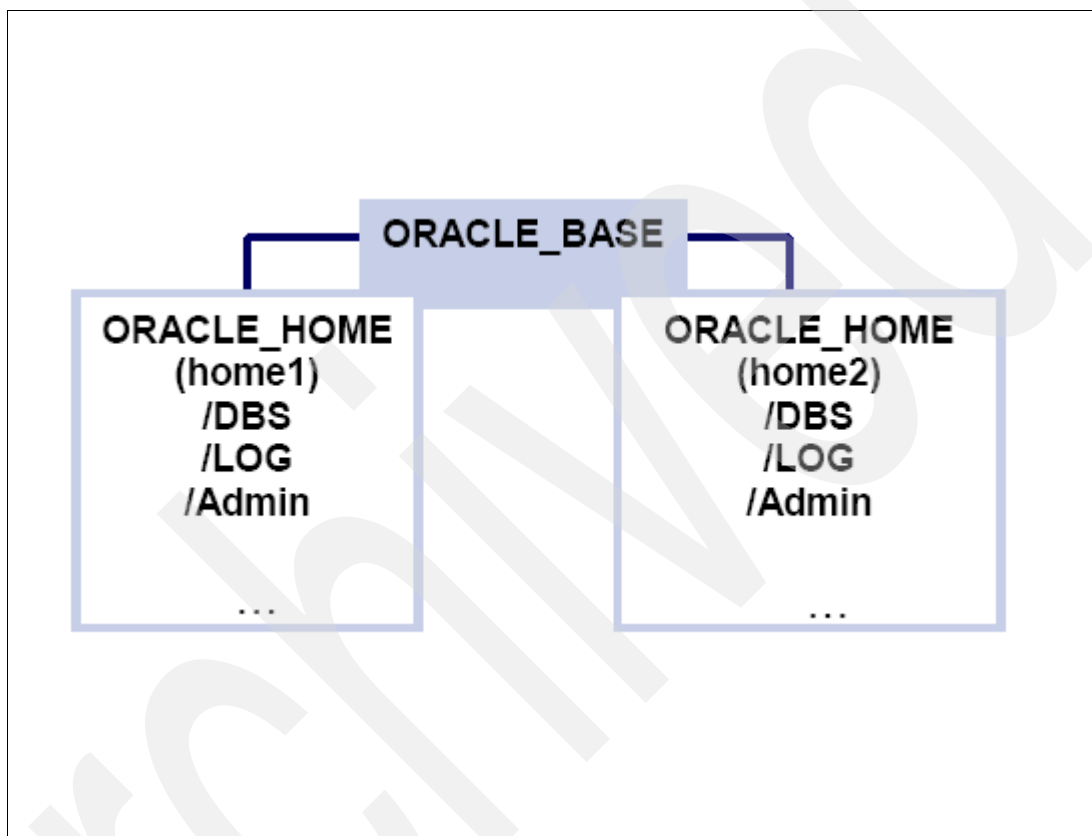


Figure 17-2 MOH layout

17.2.6 Oracle home location

OFA structure is flexible enough where ORACLE_HOME can reside either on the local file system, on an NFS mounted volume, or in a SAN environment (Figure 17-3 on page 340). For Oracle 10g, ORACLE_HOME can be shared for a specific RAC configuration where a single set of Oracle binaries and libraries are shared by multiple instances of the same database. Some details about shared ORACLE_HOME are discussed in the next sections.

Type of Files	Description	OFA Compliant Mount point	Location
ORACLE_HOME	Oracle libraries and binaries	/u01/app/oracle/product/9.2.0/ or /u01/app/oracle/product/10.1.0/db_unique_name	Local filesystem or storage system
Database files	Oracle database files	/u02/oradata	NFS mount on storage subsystem
Log Files	Oracle redo archive logs	/u03/oradata	NFS mount on storage subsystem
CRS_HOME (For 10.1.x.x RAC)	Oracle CRS HOME	/u01/app/oracle/product/10.1.0/crs_1	NFS mount on storage subsystem
CRS_HOME (For 10.2.x.x RAC)	Oracle CRS HOME	/u04/crs/product/10.2.0/app/ (Oracle 10g™ R2)	NFS mount on storage subsystem

Figure 17-3 Database locations

What is a shared ORACLE_HOME

- ▶ A shared ORACLE_HOME is an ORACLE_HOME directory that is shared by two or more hosts. This is a software install directory and typically includes the oracle binaries, libraries, network files (listener, tnsnames, and so on), oraInventory, dbs, and so forth.
- ▶ A shared ORACLE_HOME is a term used to describe an Oracle software directory that is mounted from a NFS server and access is provided to two or more hosts from the same directory path.
- ▶ An ORACLE_HOME directory looks similar to the following (/u01/app/oracle/product/10.2.0/db_1) according to the OFA.

What does Oracle support on Oracle 10g

- ▶ Single Instance (Oracle 10g) support using an NFS mounted ORACLE_HOME to a single host.
- ▶ Oracle RAC Instance (Oracle 10g) support using an NFS mounted ORACLE_HOME to one or more hosts.

What are the advantages of sharing the ORACLE_HOME in Oracle 10g

- ▶ Redundant copies are no longer needed for multiple hosts. This is extremely efficient in a testing type of environment where quick access to the Oracle binaries from a similar host system is necessary.
- ▶ Disk space savings.
- ▶ Patch application for multiple systems can be completed more rapidly. For example, if testing ten systems that you want to all run on the exact same Oracle DB versions, this is beneficial.
- ▶ It is easier to add nodes.

What are disadvantages of sharing the ORACLE_HOME in Oracle 10g

- ▶ By patching one ORACLE_HOME directory, all databases using the same home need to be bounced as well.

- In a high availability environment, having a shared ORACLE_HOME could cause downtime to a greater number of servers if impacted.

What does the N series product support regarding sharing the ORACLE_HOME

- We DO support a shared ORACLE_HOME in a RAC environment.
- We DO support a shared ORACLE_HOME for single instance Oracle when mounted to a single host system.
- We DO NOT support using a shared ORACLE_HOME in a production environment that requires high availability for a single instance Oracle set up. In other words, multiple databases should not share a single NFS mounted ORACLE_HOME while any of the database are running in production mode.

17.2.7 Best Practices for control and log files on N series

In this section we discuss recommended practices for Oracle on IBM System Storage N series.

Online redo log files

Multiplex your log files, using the following recommendations:

1. Create a minimum of two online redo log groups, each with three members.
2. Put the first online redo log group on one volume and the next on another database volume (Figure 17-4). The LGWR instance process (Log Writer) flushes the REDO Log Buffer, which contains both committed and uncommitted transactions to all members of the current online redo log group, and when the group is full it performs a log switch to the next group, and LGWR writes to all members of that group until the group fills up, and so on. Checkpoints do not cause log switches, in fact many checkpoints can occur while a log group is being filled. A checkpoint occurs when a log switch occurs.
3. Use the following suggested layout:
 Redo Grp 1: \$ORACLE_HOME/Redo_Grp1 (on IBM N series volume /vol/oralog)
 Redo Grp 2: \$ORACLE_HOME/Redo_Grp2 (on IBM N series volume /vol/oralog)

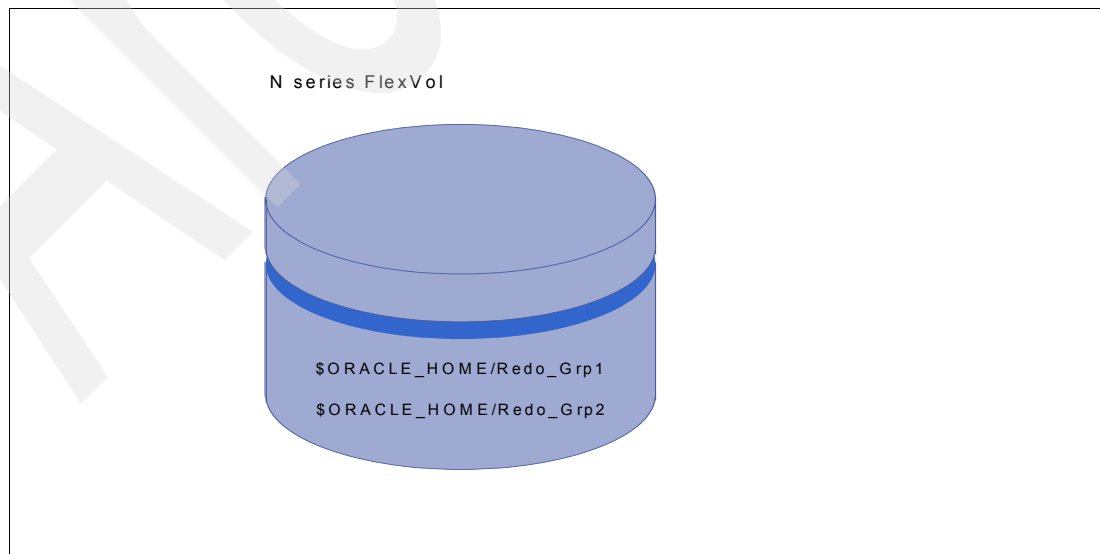


Figure 17-4 Redo logs on IBM System Storage N series volume

Archived log files

Set your init parameter, ARCHIVE_LOG_DEST, to a directory in the log volume such as \$ORACLE_HOME/log/ArchiveLog (on the N series product volume /vol/oralog).

Control files

Multiplex your control files by setting your init parameter, CONTROL_FILE_DEST, to point to destinations on at least two different N series product volumes:

Dest 1: \$ORACLE_HOME/Control_File1 (on local filesystem or on the N series product volume /vol/oralog)

Dest 2: \$ORACLE_HOME/log/Control_File2 (on the N series product volume /vol/oradata)

17.3 RAID Group Size

When the reconstruction rate (the time required to rebuild a disk after a failure) is an important factor, use smaller RAID groups.

17.3.1 RAID-DP

With RAID-DP, each RAID group is allocated an additional parity disk. Given this additional protection, the likelihood of data loss due to a double disk failure is nearly eliminated; therefore, larger RAID group sizes can be supported (Figure 17-5).

Note: RAID group sizes up to 16 disks can be safely configured using RAID-DP. However we recommend the default RAID group size of 16 for RAID-DP.

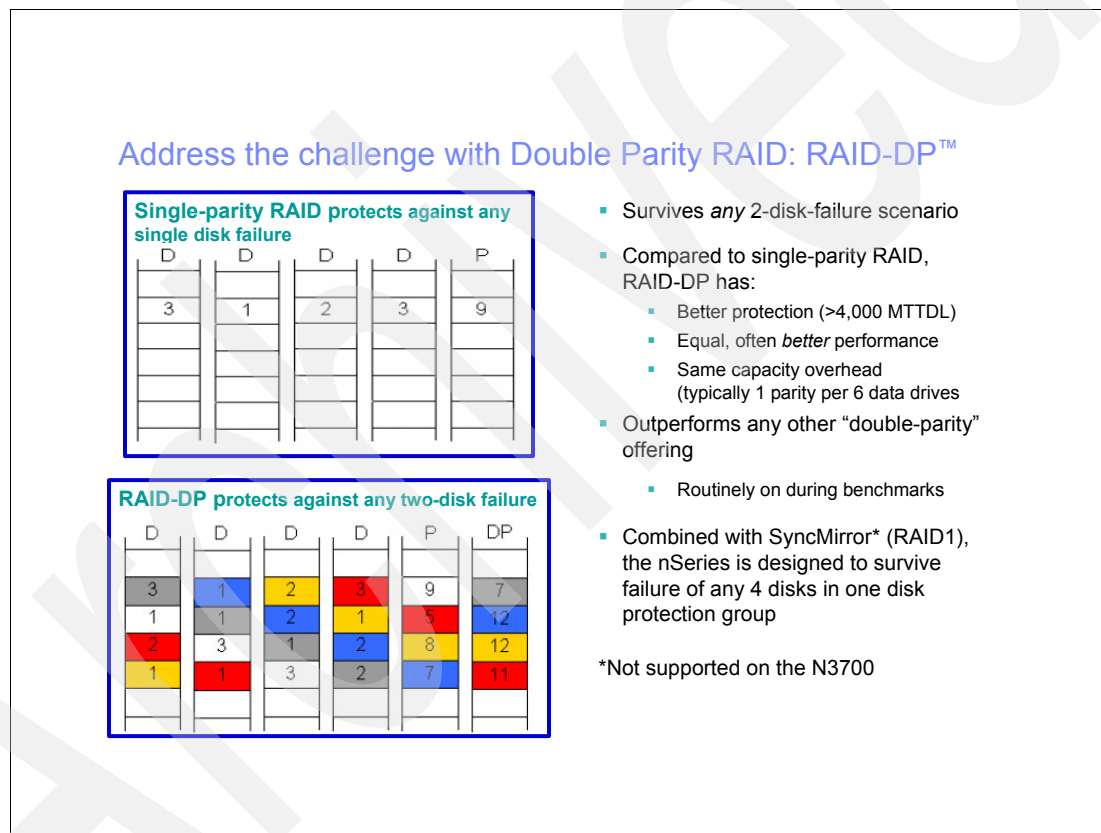


Figure 17-5 RAID-DP

17.4 Snapshot and SnapRestore

IBM System Storage N series strongly recommends using Snapshot and SnapRestore for Oracle Database backup and restore operations. Snapshot provides a point-in-time copy of the entire database in seconds without incurring any performance penalty, while SnapRestore can instantly restore an entire database to a point-in-time in the past.

In order for Snapshot copies to be effectively used with Oracle Databases, they must be coordinated with the Oracle hot backup facility. For this reason, IBM System Storage N series

recommends that automatic Snapshot copies be turned off on volumes that are storing data files for an Oracle Database.

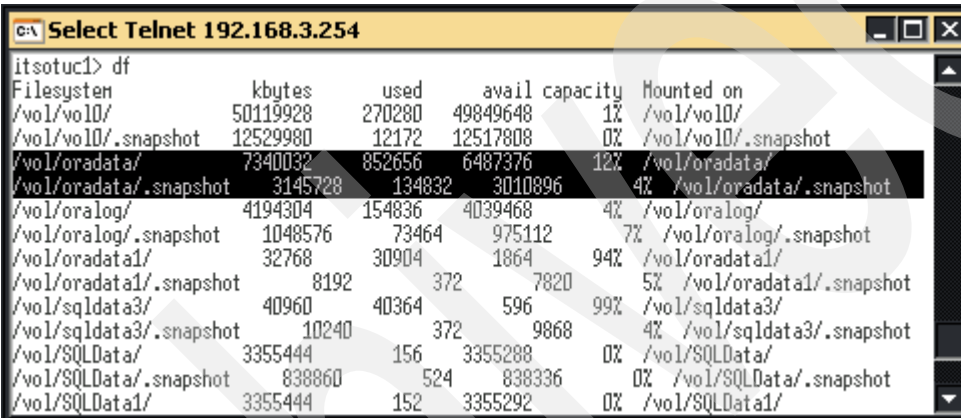
To turn off automatic Snapshot copies on a volume, issue the following command:

```
vol options <volname> nosnap on
```

If you want to make the “.snapshot” directory invisible to clients, issue the following command:

```
vol options <volname> nosnapdir on
```

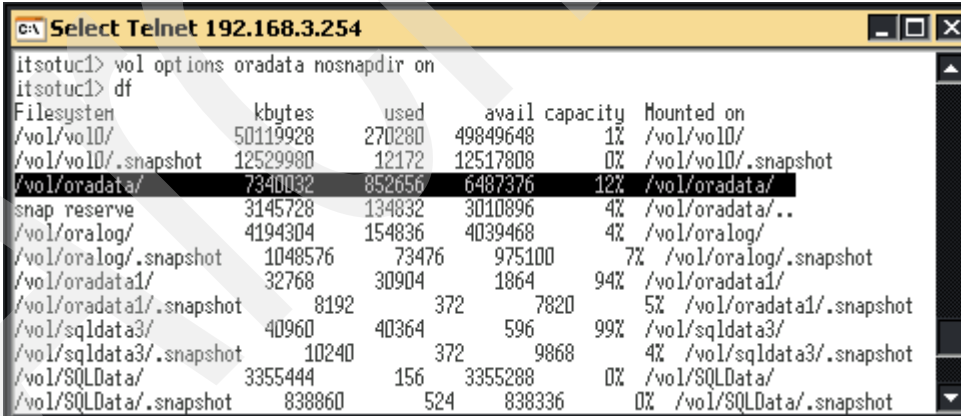
Figure 17-5 on page 343 and Figure 17-6 are examples of the before and after execution of the command.



```
itsotuc1> df
```

Filesystem	kbytes	used	avail	capacity	Mounted on
/vol/vol0/	50119928	270280	49849648	1%	/vol/vol0/
/vol/vol0/.snapshot	12529980	12172	12517808	0%	/vol/vol0/.snapshot
/vol/oradata/	7340032	852656	6487376	12%	/vol/oradata/
/vol/oradata/.snapshot	3145728	134832	3010896	4%	/vol/oradata/.snapshot
/vol/oralog/	4194304	154836	4039468	4%	/vol/oralog/
/vol/oralog/.snapshot	1048576	73464	975112	7%	/vol/oralog/.snapshot
/vol/oradata1/	32768	30904	1864	94%	/vol/oradata1/
/vol/oradata1/.snapshot	8192	372	7820	5%	/vol/oradata1/.snapshot
/vol/sqldata3/	40960	40364	596	99%	/vol/sqldata3/
/vol/sqldata3/.snapshot	10240	372	9868	4%	/vol/sqldata3/.snapshot
/vol/SQLData/	3355444	156	3355288	0%	/vol/SQLData/
/vol/SQLData/.snapshot	838860	524	838336	0%	/vol/SQLData/.snapshot
/vol/SQLData1/	3355444	152	3355292	0%	/vol/SQLData1/

Figure 17-6 Output of df before nosnapdir on



```
itsotuc1> vol options oradata nosnapdir on
itsotuc1> df
```

Filesystem	kbytes	used	avail	capacity	Mounted on
/vol/vol0/	50119928	270280	49849648	1%	/vol/vol0/
/vol/vol0/.snapshot	12529980	12172	12517808	0%	/vol/vol0/.snapshot
/vol/oradata/	7340032	852656	6487376	12%	/vol/oradata/
snap reserve	3145728	134832	3010896	4%	/vol/oradata/..
/vol/oralog/	4194304	154836	4039468	4%	/vol/oralog/
/vol/oralog/.snapshot	1048576	73476	975100	7%	/vol/oralog/.snapshot
/vol/oradata1/	32768	30904	1864	94%	/vol/oradata1/
/vol/oradata1/.snapshot	8192	372	7820	5%	/vol/oradata1/.snapshot
/vol/sqldata3/	40960	40364	596	99%	/vol/sqldata3/
/vol/sqldata3/.snapshot	10240	372	9868	4%	/vol/sqldata3/.snapshot
/vol/SQLData/	3355444	156	3355288	0%	/vol/SQLData/
/vol/SQLData/.snapshot	838860	524	838336	0%	/vol/SQLData/.snapshot

Figure 17-7 Output of df after nosnapdir on

Note: With automatic Snapshot copies disabled, regular Snapshot copies are created as part of the Oracle backup process when the database is in a consistent state.

17.5 Snap Reserve

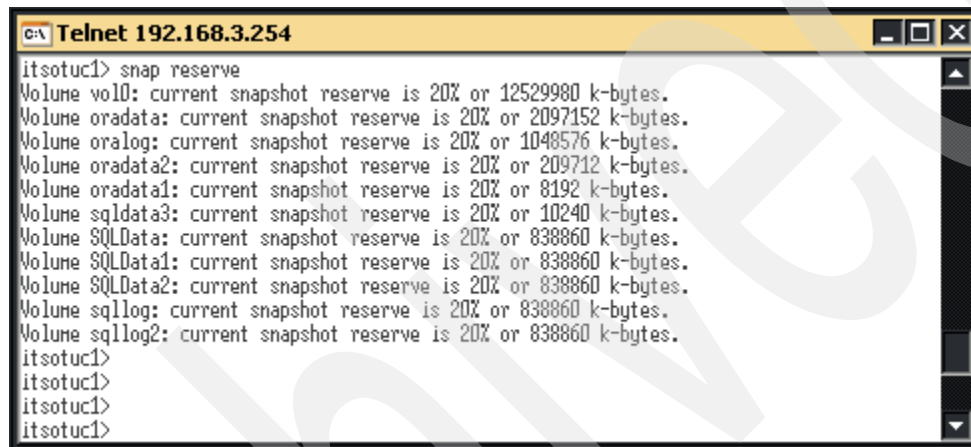
Setting the snap reserve on a volume sets aside part of the volume for the exclusive use of Snapshot copies.

Note: Snapshot copies may consume more space than allocated with snap reserve, but user files may not consume their reserved space.

To see the snap reserve size on a volume, issue the following command:

snap reserve

An example is provided in Figure 17-8.



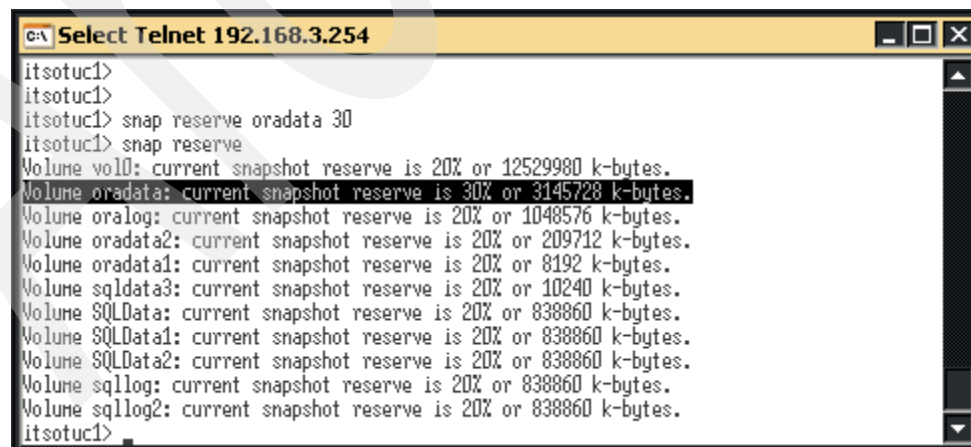
```
Telnet 192.168.3.254
itsotuc1> snap reserve
Volume vol0: current snapshot reserve is 20% or 12529980 k-bytes.
Volume oradata: current snapshot reserve is 20% or 2097152 k-bytes.
Volume oralog: current snapshot reserve is 20% or 1048576 k-bytes.
Volume oradata2: current snapshot reserve is 20% or 209712 k-bytes.
Volume oradata1: current snapshot reserve is 20% or 8192 k-bytes.
Volume sqldata3: current snapshot reserve is 20% or 10240 k-bytes.
Volume SQLData: current snapshot reserve is 20% or 838860 k-bytes.
Volume SQLData1: current snapshot reserve is 20% or 838860 k-bytes.
Volume SQLData2: current snapshot reserve is 20% or 838860 k-bytes.
Volume sqllog: current snapshot reserve is 20% or 838860 k-bytes.
Volume sqllog2: current snapshot reserve is 20% or 838860 k-bytes.
itsotuc1>
itsotuc1>
itsotuc1>
itsotuc1>
```

Figure 17-8 Snap reserve output

To set the volume snap reserve size (the default is 20%), issue the following command:

snap reserve <volume> <percentage>

An example is provided in Figure 17-9.



```
Select Telnet 192.168.3.254
itsotuc1>
itsotuc1>
itsotuc1> snap reserve oradata 30
itsotuc1> snap reserve
Volume vol0: current snapshot reserve is 20% or 12529980 k-bytes.
Volume oradata: current snapshot reserve is 30% or 3145728 k-bytes.
Volume oralog: current snapshot reserve is 20% or 1048576 k-bytes.
Volume oradata2: current snapshot reserve is 20% or 209712 k-bytes.
Volume oradata1: current snapshot reserve is 20% or 8192 k-bytes.
Volume sqldata3: current snapshot reserve is 20% or 10240 k-bytes.
Volume SQLData: current snapshot reserve is 20% or 838860 k-bytes.
Volume SQLData1: current snapshot reserve is 20% or 838860 k-bytes.
Volume SQLData2: current snapshot reserve is 20% or 838860 k-bytes.
Volume sqllog: current snapshot reserve is 20% or 838860 k-bytes.
Volume sqllog2: current snapshot reserve is 20% or 838860 k-bytes.
itsotuc1>
```

Figure 17-9 Output of snap reserve percentage

Do not use a percent sign (%) when specifying the percentage.

Adjust the snap reserve to reserve slightly more space than the Snapshot copies of a volume consume at their peak. The peak Snapshot copy size can be determined by monitoring a system over a period of a few days when activity is high.

The snap reserve may be changed at any time. Do not raise the snap reserve to a level that exceeds free space on the volume; otherwise, client machines may abruptly run out of storage space.

For IBM System Storage N series, we recommend that you observe the amount of snap reserve being consumed by Snapshot copies frequently. Do not allow the amount of space consumed to exceed the snap reserve. If the snap reserve is exceeded, consider increasing the percentage of the snap reserve or delete Snapshot copies until the amount of space consumed is less than 100%. The N series' Operations Manager can aid in this monitoring.

17.6 The N series storage system options

This section discusses the options on N series that need to be addressed for database operations.

17.6.1 The minra option

When the minra option is enabled, it minimizes the number of blocks that are prefetched for each read operation. By default, minra is turned off, and the system performs aggressive read ahead on each volume. The effect of read ahead on performance is dependent on the I/O characteristics of the application. If data is being accessed sequentially, as when a database performs full table and index scans, read ahead increases I/O performance. If data access is completely random, read ahead should be disabled, since it may decrease performance by prefetching disk blocks that are never used, thereby wasting system resources.

The following command is used to enable minra on a volume and to turn read ahead off:

```
vol options <volname> minra on
```

You can check the status of the volume with the **vol status** command. See Figure 17-10.

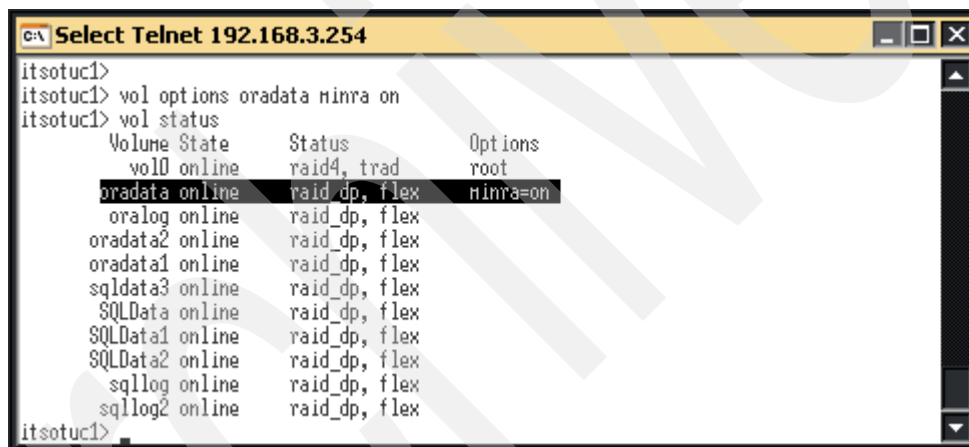


Figure 17-10 Output of vol status

Generally, the read ahead operation is beneficial to databases, and the minra option should be left alone. However for N series, we recommend that you experiment with the minra option to observe the performance impact, since it is not always possible to determine how much of an application's activity is sequential versus random. This option is transparent to client access and can be changed at will without disrupting client I/O. Be sure to allow two-to-three minutes for the cache on the appliance to adjust to the new minra setting before looking for a change in performance.

17.6.2 File access time update

Another option that can improve access time is file access time update. If an application does not require or depend upon maintaining accurate access times for files, this option can be disabled. Use this option only if the application generates heavy read I/O traffic. The following command disables file access time updates:

```
vol options <volname> no_atime_update on
```

You can check the status of this option with the **vol status** command. See Figure 17-11.

```

itsotuc1> vol options oradata no_atime_update on
itsotuc1> vol status
  Volume State      Status      Options
  -----
  vol0 online      raid4, trad root
  oradata online    raid_dp, flex minra=on,
                    no_atime_update=on
  oralog online     raid_dp, flex
  oradata2 online    raid_dp, flex
  oradata1 online    raid_dp, flex
  sqldata3 online    raid_dp, flex
  SQLData online     raid_dp, flex
  SQLData1 online    raid_dp, flex
  SQLData2 online    raid_dp, flex
  sqllog online      raid_dp, flex
  sqllog2 online     raid_dp, flex
itsotuc1>

```

Figure 17-11 Output of vol status

17.6.3 NFS settings

IBM System Storage N series recommends the use of TCP as the data transport mechanism with the current NFS V3.0 client software on the host. If it is not possible to use NFS V3.0 on the client, then it may be necessary to use UDP as the data transport mechanism. When UDP is configured as the data transport mechanism, the following NFS option should be configured on the IBM System Storage N series:

```
options nfs.udp.xfersize 32768
```

Verify your settings with the **options nfs.udp** command shown in Example 17-3.

Example 17-3 Checking nfs options

```

itsotuc2> options nfs.udp
nfs.udp.enable           on
nfs.udp.xfersize         32768

```

This sets the NFS transfer size to the maximum. There is no penalty for setting this value to the maximum of 32,768. However, if xfersize is set to a small value and an I/O request exceeds that value, the I/O request is broken up into smaller chunks, resulting in degraded performance.

17.7 Operating systems

This section covers recommended options for the OS and the DB environment.

17.7.1 Linux recommended versions

The various Linux operating systems are based on the underlying kernel. With all the distributions available, it is important to focus on the kernel to understand features and compatibility.

Kernel 2.4

The NFS client in this kernel has many improvements over the 2.2 client, most of which address performance and stability problems. The NFS client in kernels later than 2.4.16 has significant changes to help improve performance and stability.

There have been recent controversial changes in the 2.4 branch that prevented distributors from adopting late releases of the branch. Although there were significant improvements to the NFS client in 2.4.15, Torvalds also replaced parts of the VM subsystem, making the 2.4.15, 2.4.16, and 2.4.17 kernels unstable for heavy workloads. Many recent releases from Red Hat and SUSE include the 2.4.18 kernel.

The use of 2.4 kernels on hardware with more than 896 MB should include a special kernel compile option known as CONFIG_HIGHMEM, which is required to access and use memory above 896 MB. The Linux NFS client has a known problem in these configurations in which an application or the whole client system can hang at random. This issue was addressed in the 2.4.20 kernel, but still haunts kernels contained in distributions from Red Hat and SUSE that are based on earlier kernels.

Linux Kernel recommendation

IBM System Storage N series tested many kernel distributions, and those based on 2.6 are currently recommended. Recommended distributions include Red Hat Enterprise Linux Advanced Server 3.0 and 4.0 as well as Suse Enterprise Linux 9.0 (SLES9). See Table 17-2.

Table 17-2 Current recommended levels

Manufacturer	Version	Tested	Recommended
Red Hat	Advanced Server 2.1	Yes	No
Red Hat	Advanced Server 3.0	Yes	Yes
Red Hat	Advanced Server 4.0	Yes	Yes
SUSE	7.2	Yes	No
SUSE	SLES 8	Yes	No
SUSE	SLES 9	Yes	Yes

Linux Kernel patches

In all circumstances, the kernel patches recommended by Oracle for the particular database product being run should be applied first. In general, those recommendations will not conflict with the ones here, but if a conflict does arise, check with Oracle or IBM System Storage N series customer support for a resolution before proceeding. The uncached I/O patch was introduced in Red Hat Advanced Server 2.1, update 3, with kernel errata e35 and up. It is mandatory to use uncached I/O when running Oracle9i RAC with the N series product in a NAS environment. Uncached I/O does not cache data in the Linux file system buffer cache during read/write operations for volumes mounted with the "noac" mount option. To enable uncached I/O, add the following entry to the /etc/modules.conf file, and reboot the cluster nodes:

```
options nfs nfs_uncached_io=1
```

The volumes used for storing Oracle Database files should still be mounted with the "noac" mount option for Oracle9i RAC databases.

The uncached I/O patch was developed by Red Hat and tested by Oracle, IBM System Storage N series, and Red Hat.

17.7.2 Linux OS settings

This section covers those options specific to the Linux OS.

Enlarging a client's transport socket buffers

Enlarging the transport socket buffers that Linux uses for NFS traffic helps reduce resource contention on the client, reduces performance variance, and improves maximum data and operation throughput. In future releases of the client, the following procedure will not be necessary, as the client will automatically choose an optimal socket buffer size.

```
su to root on the client
cd into /proc/sys/net/core
echo 262143 > rmem_max
echo 262143 > wmem_max
echo 262143 > rmem_default
echo 262143 > wmem_default
```

Remount the NFS file systems on the client.

This is especially useful for NFS over UDP and when using Gigabit Ethernet. Consider adding this to a system startup script that runs before the system mounts NFS file systems. **The recommended size (262,143 bytes) is the largest safe socket buffer size that has been tested at the time of this publication.** On clients with 16 MB of memory or less, leave the default socket buffer size setting to conserve memory.

Red Hat distributions after 7.2 contain a file called `/etc/sysctl.conf` where changes such as this can be added so they are executed after every system reboot. Add these lines to the `/etc/sysctl.conf` file on these Red Hat systems.

```
net.core.rmem_max = 262143
net.core.wmem_max = 262143
net.core.rmem_default = 262143
net.core.wmem_default = 262143
```

Figure 17-12 is an example of the lines added to the `sysctl.conf` file.

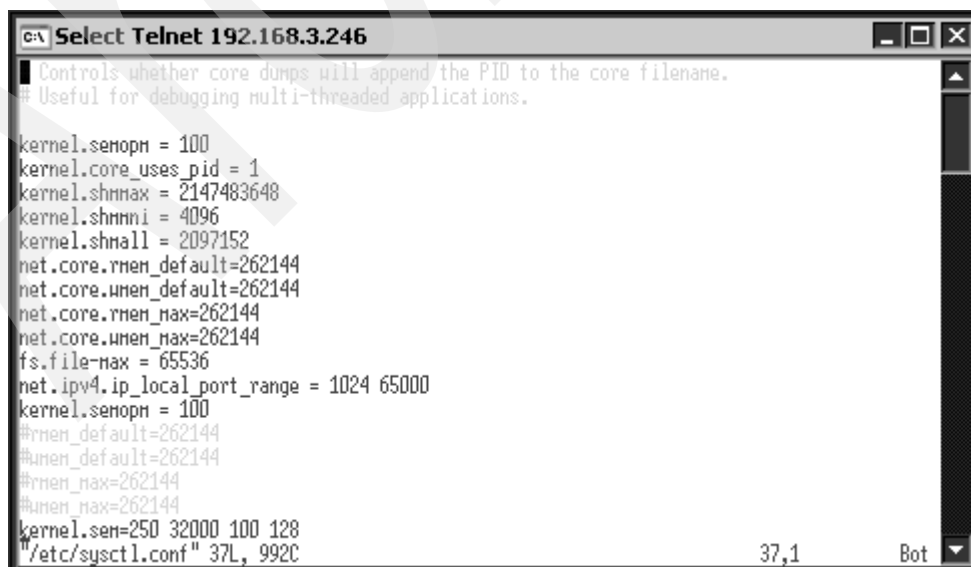


Figure 17-12 Example of `/etc/sysctl.conf` file in Linux

Other TCP enhancements

The following settings can help reduce the amount of work clients and IBM System Storage N series do when running NFS over TCP:

```
echo 0 > /proc/sys/net/ipv4/tcp_sack
echo 0 > /proc/sys/net/ipv4/tcp_timestamps
```

These operations disable optional features of TCP to save a little processing time and network bandwidth. When building kernels, be sure that CONFIG_SYNCOOKIES is disabled. SYN cookies slow down TCP connections by adding extra processing on both ends of the socket. Some Linux distributors provide kernels with SYN cookies enabled.

Linux 2.2 and 2.4 kernels support large TCP windows (RFC 1323) by default. No modification is required to enable large TCP windows.

Linux Networking—full duplex and auto negotiation

Most network interface cards use auto negotiation to obtain the fastest settings allowed by the card and the switch port to which it attaches. Sometimes, chipset incompatibilities may result in constant renegotiation or negotiating half duplex or a slow speed. When diagnosing a network problem, be sure the Ethernet settings are as expected before looking for other problems. Avoid hard coding the settings to solve auto negotiation problems, because it only masks a deeper problem. Switch and card vendors should be able to help resolve these problems.

Linux Networking—Gigabit Ethernet network adapters

If Linux servers are using high-performance networking (gigabit or faster), provide enough CPU and memory bandwidth to handle the interrupt and data rate. The NFS client software and the gigabit driver reduce the resources available to the application, so make sure resources are adequate. Most gigabit cards that support 64-bit PCI or better should provide good performance.

Note: Any database using IBM System Storage N series should utilize Gigabit Ethernet on both the N series product and database server to achieve optimal performance.

Linux networking—jumbo frames with Gigabit Ethernet

All of the cards described above support the jumbo frames option of Gigabit Ethernet. Using jumbo frames can improve performance in environments where Linux NFS clients and IBM System Storage N series systems are together on an unrouted network. Be sure to consult the command reference for each switch to make sure it can handle jumbo frames. There are some known problems in Linux drivers and the networking layer when using the maximum frame size (9000 bytes). If unexpected performance slow downs occur when using jumbo frames, try reducing the MTU to 8960 bytes.

Linux NFS protocol—mount options

Table 17-3 on page 352 summarizes the list of recommended NFS client side mount options for various Oracle versions and OS platform permutations.

Table 17-3 mount options

Oracle and OS version	NFS mount options
Oracle 9i non-RAC Solaris	rw,bg,hard,intr,rsize=32768, wsize=32768,[forcedirectio or llock],tcp,vers=3
Oracle 10g non- RAC - Solaris	rw,bg,hard,intr,rsize=32768, wsize=32768,[forcedirectio or llock],tcp,vers=3
Oracle 9i non-RAC RHAS 2.1	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600;
Oracle 10g non-RAC - RHAS 2.1	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600;
Oracle 10g non-RAC - RHEL 3.0	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600
Oracle 10g non-RAC - RHEL 4.0	Oracle parameter filesystemio_options=directio rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600
Oracle 9i non-RAC - SLES 8/9	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600
Oracle 10g non-RAC - SLES 8/9	rw,bg,hard,nointr,rsize=32768, wsize=32768,tcp,vers=3, timeo=600

iSCSI initiators for Linux

iSCSI support for Linux is just now becoming available in a number of different forms. Both hardware and software initiators are starting to appear but have not reached a level of adoption to merit a great deal of attention. Testing is insufficient to recommend any best practices at this time. This section will be revisited in the future for any recommendations or best practices for running Oracle Databases on Linux with iSCSI initiators.

FCP SAN initiators for Linux

The N series product supports Fibre Channel storage access for Oracle Databases running on a Linux host. Connections to IBM System Storage N series can be made through a Fibre Channel switch (SAN) or direct-attached. The N series product currently supports Red Hat Enterprise Linux 2.1 and 3.0 and SUSE Enterprise Server 8 on a Linux host with N series.

IBM System Storage N series recommends using Fibre Channel SANs for Oracle Databases on Linux, where there is an existing investment in Fibre Channel infrastructure or the sustained throughput requirement for the database server is greater than 1GB per second (~110MB per second).

17.8 Oracle Database settings

This section describes settings that are made to the Oracle Database application, usually through settings contained in the “init.ora” file. We assume that you have an existing knowledge of how to correctly set these settings and an idea of their effect. The settings described here are the ones most frequently tuned when using IBM System Storage N series with Oracle Databases.

17.8.1 DISK_ASYNCH_IO

Enables or disables Oracle asynchronous I/O. Asynchronous I/O allows processes to proceed with the next operation without having to wait for an issued write operation to complete, therefore, improving system performance by minimizing idle time. This setting may improve performance depending on the database environment. If the DISK_ASYNCH_IO parameter is set to TRUE, then DB_WRITER_PROCESSES and

DB_BLOCK_LRU_LATCHES (Oracle versions prior to 9i) or DBWR_IO_SLAVES must also be used, as described below. The calculation looks as follows:

$$\text{DB_WRITER_PROCESSES} = 2 * \text{number of CPUs}$$

Oracle Database 10g does not install asyncio by default. To enable asyncio, you need to relink oracle. After the installation process successfully completes, execute the following procedure to enable asyncio.

```
Linux20:~> cd $ORACLE_HOME/rdbms/lib
Linux20:~/oradbf/O10g/rdbms/lib> make -f ins_rdbms.mk async_on
Linux20:~/oradbf/O10g/rdbms/lib> make -f ins_rdbms.mk ioracle
```

Attention: Use `make -f ins_rdbms.mk async_off` to switch back, in case of problems.

After we completed the previously mentioned steps, we made changes to the `init.ora`. We included the following two parameters in our `init.ora` to enable asyncio and Direct I/O

```
DISK_ASYNC_IO=TRUE
FILESYSTEMIO_OPTIONS=SETALL
```

There is a combination of parameters to use, depending on what you want to do.

- ▶ `DISK_ASYNC_IO=TRUE` is all that is needed for asyncio for raw devices.
- ▶ `FILESYSTEMIO_OPTIONS=ASYNC` is required for asyncio only for file systems.

17.8.2 DB_FILE_MULTIBLOCK_READ_COUNT

Determine the maximum number of database blocks read in one I/O operation during a full table scan. The number of database bytes read is calculated by multiplying `DB_BLOCK_SIZE * DB_FILE_MULTIBLOCK_READ_COUNT`. The setting of this parameter can reduce the number of I/O calls required for a full table scan, thus improving performance. Increasing this value may improve performance for databases that perform many full table scans but degrade performance for OLTP databases where full table scans are seldom (if ever) performed.

Setting this number to a multiple of the NFS READ/WRITE size specified in the mount will limit the amount of fragmentation that occurs in the I/O subsystem. Be aware that this parameter is specified in “DB Blocks,” and the NFS setting is in “bytes,” so adjust as required. As an example, specifying a `DB_FILE_MULTIBLOCK_READ_COUNT` of 4 multiplied by a `DB_BLOCK_SIZE` of 8 kB will result in a read buffer size of 32 kB.

Note: It is recommended that `DB_FILE_MULTIBLOCK_READ_COUNT` should be set from 1 to 4 for an OLTP database and from 16 to 32 for DSS.

17.8.3 DB_BLOCK_SIZE

For best database performance, DB_BLOCK_SIZE should be a multiple of the OS block size. For example, if the Solaris page size is 4096:

$$\text{DB_BLOCK_SIZE} = 4096 * n$$

The NFS rsize and wsize options specified when the file system is mounted should also be a multiple of this value. Under no circumstances should it be smaller. For example, if the Oracle DB_BLOCK_SIZE is set to 16 kB, the NFS read and write size parameters (rsize and wsize) should be set to either 16 kB or 32 kB, never to 8 kB or 4 kB

17.8.4 DBWR_IO_SLAVES and DB_WRITER_PROCESSES

DB_WRITER_PROCESSES is useful for systems that modify data heavily. It specifies the initial number of database writer processes for an instance. If DBWR_IO_SLAVES is used, only one database writer process is allowed, regardless of the setting for DB_WRITER_PROCESSES. Multiple DBWRs and DBWR IO slaves cannot coexist. We recommend that one or the other be used to compensate for the performance loss resulting from disabling DISK_ASYNC_IO. Metalink note 97291.1 provides guidelines on usage.

The first rule of thumb is to always enable DISK_ASYNC_IO if it is supported on that OS platform. Next, check to see if it is supported for NFS or only for block access (FC/iSCSI). If supported for NFS, then consider enabling async I/O at the Oracle level and at the OS level and measure the performance gain. If performance is acceptable, then use async I/O for NFS. If async I/O is not supported for NFS or if the performance is not acceptable, then consider enabling multiple DBWRs and DBWR IO slaves as described next. Multiple DBWRs and DBWR IO slaves cannot coexist. We recommend that one or the other be used to compensate for the performance loss resulting from disabling DISK_ASYNC_IO. Metalink note 97291.1 provides guidelines on usage. The recommendation is that DBWR_IO_SLAVES be used for single CPU systems and that DB_WRITER_PROCESSES be used with systems having multiple CPUs.

Note: We recommend that DBWR_IO_SLAVES be used for single-CPU systems and that DB_WRITER_PROCESSES be used with systems having multiple CPUs.

17.8.5 DB_BLOCK_LRU_LATCHES

The number of DBWRs cannot exceed the value of the DB_BLOCK_LRU_LATCHES parameter:

$$\text{DB_BLOCK_LRU_LATCHES} = \text{DB_WRITER_PROCESSES}$$

Starting with Oracle9i, DB_BLOCK_LRU_LATCHES is obsolete and need not be set.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redbooks publication.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 356. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IBM System Storage N series File System Design for an NFS File Server, REDP-4086*
- ▶ *IBM System Storage N Series Implementation of RAID Double Parity for Data Protection, REDP-4169*
- ▶ *N Series SnapManager with Microsoft SQL, REDP-4174*
- ▶ *Multiprotocol Data Access with IBM System Storage N series, REDP-4176*

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM System Storage N series Data ONTAP 7.1 Upgrade and Reinstallation Guide, GC26-7791-00*
- ▶ *IBM System Storage N series Block Access Management Guide for FCP and iSCSI, GA32-0528-01*
- ▶ *IBM System Storage N series Data ONTAP 7.2 Data Protection Online Backup and Recovery Guide, GC26-7967-00*

Online resources

The following Web sites are also relevant as further information sources:

- ▶ Support for IBM System Storage and TotalStorage products
<http://www-304.ibm.com/jct01004c/systems/support/supportsite.wss/storageselectproduct?brandind=5000029&familyind=0&oldfamily=0>
- ▶ Support for Data ONTAP
<http://www-304.ibm.com/jct01004c/systems/support/supportsite.wss/supportresources?brandind=5000029&familyind=5329797&taskind=1>
- ▶ IBM System Storage N series and TotalStorage NAS interoperability matrixes
<http://www-03.ibm.com/systems/storage/nas/interophome.html>

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, IBM Redpapers publications, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks publications or CD-ROMs, at the following Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

2864-A10 13
2864-A20 13
2865-A10 13
2865-A20 13
2865-G20 24
2866-A10 15
2866-A20 15
2866-G10 24
2866-G20 24
2867-A10 15
2867-A20 15, 24
2867-G10 24
2868 - A10 13
2868 -A20 13
3U 12
6U 17

A

aggr create -r -t 114
aggregates 114
ARCH 32

B

business practices 20

C

CJQn 33
CKPT 32
clone database
 verifying 201
clone volumes
 mounting 177, 197
clone volumes using Snapshot copies
 creating 204
cloned database
 configuring 179, 186, 198, 207
cloned volumes
 mounting 184
clones of the FlexVol volumes
 creating 194
Compact flash 12
Compliance 3
conventional storage 20
CPU 13, 15

D

Data ONTAP 3–4
database 2
 verifying 181, 188
database into a consistent state (suspend writes)
 bringing 182

database server
 different DB2 UDB version
 selecting 173
db2
 "catalog database mydb as mydb on /mnt/dbdata"
 181
DB2 UDB Database in the N series Environment
 cloning 171
DBDir
 179
DBWn 31
DestinationStorageSystem 192
Direct Access Storage 19
disk failure with a hot spare disk 132
Disk sanitization 3
Dynamic SGA 26

E

ESH2 13
Ethernet 12
expansion shelf 12
export entry for the clone volume
 creating 176
exportfs -p rw=,root= 113

F

Fast commit 32
FC 10
FCP 3, 19
Fibre Channel adapters 12
file access 19
FilerView 1, 113
FilerView™ 176
FileSystem 178
FlexVol volumes
 cloning 175
FlexVolName 177

G

Gateway 3, 18, 20
Gateways 18
gateways 18

H

hot plug 12

I

IBM N series A models 4
IBM N series G Models 5
IBM System Storage IBM N series 1
infrastructure 13
InstanceName

179
InstanceOwner 178
InstanceOwnerGroup 178
iSCSI 3

J

Job coordinator 33

K

vol create size 114

L

LCKn 33
LGWR 32
license add 111
LockVault 3
LUN 4, 24
LUNs 4

M

MIPS 12
MountPoint 178
Multiproctocol 1
multiprotocol 10

N

N3700 2, 11
N5000 2–4, 13
N5200 14
N5500 13
N7000 2, 15–16, 23–24
N7600 15, 18, 24
N7800 15, 24
NAS 3
Nearline™ 3
NewInstanceName 180
NFS export entries for the cloned volumes
 creating 197, 205
non production server without DB2 UDB installed
 selecting 173
normal database operations (resume writes)
 183, 203

O

OEM 35
offline database
 to a remote storage system
 cloning 188
OldInstanceName 180
online database
 on the same storage system
 cloning 182
 to a remote storage system
 cloning 202
options rsh.enable on 112
Oracle

Administration tools 33
Background processes 31
Commit record 32
Control file 30
Data blocks 27
Data dictionary cache 26
Data segments 28
Database buffer cache 25
Datafiles 29
Dedicated server 31
Execution plan 26
Export utilities 34
Extents 27
Import utilities 34
Incremental extent 27
Initial extent 27
Instance 25
Logical Storage Structures 27
LRU list 25
Memory Structures 24
Multithreaded server 31
Oracle Enterprise Manager (OEM) 35
PGA 24
Physical Storage Structures 29
Private SQL areas 26
Processes 30
Redo log buffer 25–26
Redo logs 29
Rollback segments 28
Schema objects 29
Segment 27
Serializable transactions 28
Server Manager 34
Server processes 31
Session 30
SGA 24
Shared pool 25
Shared pool area 26
Shared SQL areas 26
SQL*Loader 34
SQLJ 33
System change number (SCN) 32
Tablespaces 29
Temporary segments 28
Transaction-level read consistency 28
User processes 30
Oracle commands
 exp 34
 imp 34
Oracle Parallel Server 37
Oracle parameters
 log_buffer 26
Oracle processes
 Archiver (ARCH) 32
 Checkpoint Process (CKPT) 32
 Database Writer (DBWn) 31
 Dispatcher(Dnnn) 33
 LOCK (LCKn) 33
 Log Writer (LGWR) 32
 Process Monitor (PMON) 33

Recover (RECO) 33
System Monitor (SMON) 32

P

Parallel databases 23
ParentSnap 175
ParentVol 175
PGA 24, 26
PMON 33
Program Global Areas (PGA) 24
protection solutions 20
provisioning 20

R

RAID 1, 4
RAID0 21–22
RAID4 3
RECO 33
Redbooks Web site 294
 Contact us xiii
Redundant Cooling 12

S

SAN 3, 19
SATA 11, 13, 15
set write
 suspend for database 182
SGA 24
SID 25
SMON 32
snap create dbdata dbdata_cl_snp01 174
SnapLock 3
SnapMirror 24
 configuring and initializing 189, 202
 initialize command 193
 updating destination 203
Snapshot copies
 database FlexVol volumes
 creating 182
 FlexVol volumes
 creating 203
Snapshot copy 176
SnapVault 25–26
source database
 cataloging 181, 201
 starting 175
SourceStorageSystem 192
storage management 20
storage systems 19
storage utilization 20
StorageSystemName 177
synchronous 24
System Global Area (SGA) 24

V

virtualization 18

vol clone create dbdata_cl -s volume -b dbdata
dbdata_cl_snp.01 175
vol status dbdata_cl 175
VolumeName 192

W

WAFL 3

Archived



Using the IBM System Storage N Series with Databases

(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



Using the IBM System Storage N Series with Databases



Redbooks

IBM DB2 UDB and N series

Oracle and N series

Microsoft SQL Server and N series

This IBM® Redbooks® publication discusses how to optimize the IBM System Storage™ N series products with some of the major commercially available databases available to customers today. Topics include installation, performance, monitoring, and management when using the IBM System Storage N series with IBM DB2®, Microsoft® SQL, and Oracle®. We also cover best practices and tips for using the IBM System Storage N series with these major database applications.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7292-00

ISBN 0738489166