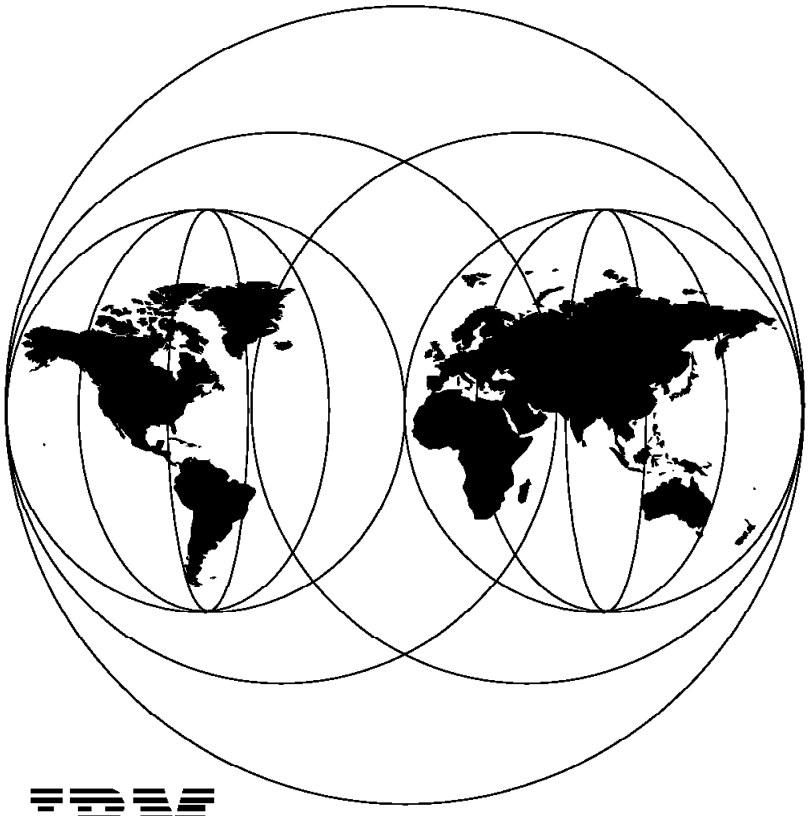
SG24-4826-00

Database Parallelism on the AS/400

December 1996



IBM

International Technical Support Organization
Rochester Center

International Technical Support Organization

SG24-4826-00

Database Parallelism on the AS/400

December 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 141.

First Edition (December 1996)

This edition applies to Version 3 Release 1, Version 3 Release 2 and Version 3 Release 7 of DB2 Symmetric Multiprocessing for OS/400. It also applies to Version 3 Release 2 and Version 3 Release 7 of DB2 Multisystem for OS/400. Both products are features of the corresponding versions of the OS/400 operating system.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

Contents

Figures vii

Tables ix

Preface xi
How This Redbook Is Organized xi
The Team That Wrote This Redbook xii
Comments Welcome xiii

Chapter 1. Database Parallelism on the AS/400 System 1
1.1 Two Flavors of Parallelism: Tightly Coupled and Loosely Coupled 1
1.2 Introducing Symmetric Multiprocessing 2
1.3 Introducing DB2 Multisystem 3
1.3.2 When To Use DB2 Multisystem 5

Chapter 2. Introduction to Symmetric Multiprocessing 7
2.1 What Does Symmetric Multiprocessing Do? 7
2.1.1 Symmetric Multiprocessing and I/O Parallelism 7
2.1.2 Symmetric Multiprocessing and Work Management 7
2.2 Supported Interfaces 8
2.2.1 Activating Symmetric Multiprocessing 10
2.2.2 Activating SMP in Client/Server Environments 11
2.3 Symmetric Multiprocessing Algorithms: An Overview 13
2.3.1 Considerations on Ordering Result Sets 14
2.4 Will Symmetric Multiprocessing Help? 15
2.4.1 Macroscopical Indicators 15
2.4.2 Balancing System Resources for Symmetric Multiprocessing 16
2.4.3 Gathering Some More Information 17
2.4.4 Symmetric Multiprocessing Inhibitors and Limitations 19

Chapter 3. Parallel Table Scan 21
3.1 How Parallel Table Scan Works 21
3.2 When Parallel Table Scan Is Beneficial 21
3.3 Examples 22

Chapter 4. Parallel Index Scan 25
4.1 How Parallel Index Scan Works 25
4.2 When to Use Parallel Index Scan 25
4.3 Examples 27

Chapter 5. Parallel Key Positioning 31
5.1 How Parallel Key Positioning Works 31

Chapter 6. Index Only Access 33
6.1 How Index Only Access Works 33
6.2 Examples 34

Chapter 7. Hash Group By 37
7.1 How Hash Group By Works 37
7.2 When to Use Hash Group By 37
7.3 Examples 38

Chapter 8. Hash Join . . . 43
8.1 How Hash Join Works . . . 43
8.2 When to Use Hash Join . . . 43
8.3 Examples . . . 44
8.4 Hash Join and Subquery Processing . . . 46

Chapter 9. Setting Up and Implementing DB2 Multisystem . . . 49
9.1 Prerequisites to Activating DB2 Multisystem . . . 49
9.1.1 Communications Considerations . . . 49
9.1.2 Security . . . 51
9.1.3 Partition Map . . . 51
9.1.4 Partitioning Key . . . 52
9.1.5 Determine What Files To Distribute . . . 54
9.2 Activating DB2 Multisystem . . . 55
9.2.1 Creating A Node Group . . . 55
9.2.2 Displaying A Node Group . . . 56
9.2.3 Changing A Node Group . . . 58
9.3 Monitoring Your Distributed Files . . . 59
9.4 Control Language Commands and Distributed Files . . . 59
9.5 CCSIDs and SRTSEQ Tables . . . 59

Chapter 10. How To Distribute . . . 61
10.1 How To Choose A Partitioning Key . . . 61
10.1.1 Distribution of Workload Among Systems . . . 61
10.1.2 Logical Separation of Data . . . 63
10.1.3 Visibility Node . . . 64
10.2 Working With Node Groups and Partitioning Files . . . 66
10.2.1 Working With Node Groups . . . 66
10.2.2 Working With Partitioning Files . . . 68
10.3 Monitoring for Distribution . . . 70

Chapter 11. Managing Distributed Files . . . 73
11.1 Availability . . . 73
11.2 Journaling . . . 74
11.3 Commitment Control . . . 76
11.4 Disaster Recovery Planning . . . 79
11.4.1 How to Save Distributed Files . . . 80
11.4.2 How to Restore Distributed Files . . . 81
11.5 Security Considerations . . . 85
11.6 National Language Support . . . 90
11.7 Managing Jobs . . . 91
11.7.1 Managing Temporary Result Writer Jobs . . . 95
11.8 Redistribution Considerations . . . 99

Chapter 12. Distributed Queries . . . 101
12.1 Distributed Optimization . . . 101
12.1.1 Communications Considerations . . . 103
12.2 Single File Queries . . . 106
12.3 Record Ordering . . . 108
12.4 UNION and DISTINCT Clauses . . . 108
12.5 Distributed Join . . . 110
12.5.1 Co-located join . . . 111
12.5.2 Directed Join . . . 113
12.5.3 Re-Partitioned Join . . . 116
12.5.4 Broadcast Join . . . 120

12.6 Implementation of Grouping 123
12.6.1 One-Step Grouping 123
12.6.2 Two-Step Grouping 124
12.6.3 Grouping and Joins 126
12.7 Improving Performance of Distributed Queries 126
12.8 Parallel Support for Distributed Queries 127
12.8.1 Temporary Result Writers 127
12.8.2 Parallel Interaction with Remote Nodes 129
12.8.3 Symmetric Multiprocessing 129
12.8.4 CHGQRYA Command 132
12.8.5 Combining Parallel Options 133

Appendix A. Database Files Used in Examples 135
A.1 File LINEITEM 135
A.2 File ORDERS 135

Appendix B. Hash Function Table 137

Appendix C. PTF List for DB2 Multisystem for OS/400 Feature 139
C.1 OS/400 Version 3 Release 2 Modification 0 139
C.2 OS/400 Version 3 Release 7 Modification 0 139

Appendix D. Special Notices 141

Appendix E. Related Publications 143
E.1 International Technical Support Organization Publications 143
E.2 Redbooks on CD-ROMs 143
E.3 Other Publications 143

How To Get ITSO Redbooks 145
How IBM Employees Can Get ITSO Redbooks 145
How Customers Can Get ITSO Redbooks 146
IBM Redbook Order Form 147

List of Abbreviations 149

Index 151

Figures

1. Symmetric Multiprocessing Architecture . . . 9
2. Work with Registration Information . . . 12
3. A Generic LIKE Clause . . . 21
4. Query Optimizer Message for Parallel Table Scan . . . 23
5. Query Optimizer Message for Parallel Index Scan . . . 28
6. Number of Tasks Messages for Parallel Index Scan . . . 28
7. Index Only Access with Record Selection . . . 35
8. Index Only Access with Join . . . 36
9. A Group By Query . . . 39
10. Query Optimizer Message for Hash GROUP BY . . . 40
11. Combining ORDER BY and Hash Group By . . . 41
12. Query Optimizer Message for Hash Join . . . 45
13. Subquery Implemented as a Join . . . 46
14. A Join Query Equivalent to the Subquery . . . 46
15. Messages Issued During the Execution of the Modified Subquery . . . 46
16. Example of Relational Database Directory Entries . . . 50
17. Basis For Node Group and Elements of Default Partitioning Map . . . 51
18. Example of SQL Display Using Hash Function . . . 53
19. Example of Creation of a Node Group . . . 56
20. Example of Display Node Group . . . 57
21. Example of Display Node Group Partitioning Data . . . 57
22. Example of Change Node Group Attributes Partition Number . . . 58
23. Distribution of Records Between Nodes for Distributed File . . . 63
24. RSTOBJ Message for Node Group Object . . . 67
25. Displaying Partitioning Map for a Node Group Object . . . 69
26. Sample DDS for a Partitioning File . . . 69
27. Displaying Description of a Distributed File . . . 71
28. Estimating Record Migration When Redistributing File . . . 72
29. Starting Journaling for Distributed File . . . 74
30. Ending Journaling for a Distributed File . . . 75
31. Forcing ROLLBACK for Commitment Definition . . . 78
32. Cancelling Resynchronization for Commitment Definition . . . 79
33. Restoring Distributed File to Incorrect Node . . . 82
34. Deleting Distributed File Object When Node Not Accessible . . . 85
35. QCMN Subsystem Communications Entries . . . 87
36. APPN Local Location List . . . 88
37. APPN Remote Location List . . . 88
38. Changing Relational Database Directory Entry . . . 89
39. Distributed File on Systems with Different CCSIDs . . . 91
40. DDM Jobs Supporting Distributed Environment . . . 92
41. WRKACTJOB Display for Query Originating Node . . . 93
42. WRKACTJOB Display for Remote Node for Query . . . 94
43. DDM Jobs Started for CHGPF Command . . . 95
44. Temporary Result Writer Jobs on WRKACTJOB Display . . . 96
45. Autostart Job Entries for QSYSWRK Subsystem . . . 97
46. QQQTEMPS Job Description . . . 98
47. Query Optimizer Messages for Distributed Query . . . 102
48. Distributed Query over Three-Node Network . . . 104
49. Configuring APPC Mode for Distributed File Processing . . . 105
50. Determining Nodes for Distributed Query . . . 106
51. Query Optimizer Messages for Distributed Query with DISTINCT . . . 109

52. Query Optimizer Messages for Distributed Query with DISTINCT and ORDER BY 110
53. Query Optimizer Messages for Query with a Co-Located Join 112
54. Query Optimizer Message with Details about Co-Located Join 112
55. A Directed Join Example 113
56. Query Optimizer Messages for Query with Directed Join 114
57. Temporary Distributed Result File for Directed Join 114
58. Query Optimizer Messages for Re-Partitioned Join 117
59. Temporary Distributed Result File for First File in a Re-Partitioned Join 118
60. Temporary Distributed Result File for Second File in a Re-Partitioned Join 119
61. Query Optimizer Messages for Broadcast Join 121
62. Temporary Distributed Result File for Broadcast Join 122
63. Query Optimizer Messages for One-Step Grouping 124
64. Query Optimizer Messages for Two-Step Grouping 125
65. SMP Support for Distributed Query on Coordinator Node 130
66. SMP Support for Distributed Query on Remote Node 131
67. SMP Support for Distributed Query on Another Remote Node 131
68. CHGQRYA Command Parameters 132

Tables

1. Parallel Table Scan - Uni-Processor 23
2. Parallel Table Scan - Multiprocessor 24
3. Parallel Index Scan - Uni-Processor 27
4. Index Only Access 34
5. Hash GROUP BY 39
6. Hash Join 45
7. LINEITEM Database File Structure 135
8. ORDERS Database File Structure 135
9. Partitioning Key to Partition Number Conversion 137
10. Partition Number to Partitioning Key Conversion 137

Preface

This redbook discusses the new options for implementing database parallelism on the AS/400 platform. DB2 Symmetric Multiprocessing for OS/400 and DB2 Multisystem for OS/400 are covered in this document. Both the functional aspects of the products and the performance benefits they provide to the users are described here.

This book is primarily addressed to system and database administrators, consultants, and technical IT specialists who are looking at the AS/400 system as a database engine, especially for data warehousing implementations.

Through a series of examples and practical cases, the reader can understand how to improve query performance with DB2 Symmetric Multiprocessing and how to consolidate a network of AS/400 systems into a single DB2 for OS/400 database using DB2 Multisystem.

How This Redbook Is Organized

This redbook contains 157 pages. It is organized as follows:

- Chapter 1, “Database Parallelism on the AS/400 System”

  This chapter provides an overview of both Symmetric Multiprocessing and DB2 Multisystem.

- Chapter 2, “Introduction to Symmetric Multiprocessing”

  This chapter introduces the Symmetric Multiprocessing functions and describes how you need to operate to activate Symmetric Multiprocessing on your system.

- Chapter 3, “Parallel Table Scan”

  This chapter discusses the advantages introduced by Parallel Table Scan. Examples and performance tests are also included.

- Chapter 4, “Parallel Index Scan”

  This chapter discusses the advantages introduced by Parallel Index Scan. Examples and performance tests are also included.

- Chapter 5, “Parallel Key Positioning”

  This chapter discusses the advantages introduced by Parallel Key Positioning.

- Chapter 7, “Hash Group By”

  This chapter discusses the advantages introduced by Hash Group By. Examples and tests are also included.

- Chapter 8, “Hash Join”

  This chapter discusses the advantages introduced by Hash Join. Examples and tests are also included.

- Chapter 9, “Setting Up and Implementing DB2 Multisystem”

  This chapter introduces DB2 Multisystem. Examples of configuring an DB2 Multisystem environment are provided here.

- Chapter 10, “How To Distribute”

This chapter goes into details about the design guidelines you should follow when you implement a solution based on a DB2 Multisystem.

- Chapter 11, “Managing Distributed Files”

  This chapter shows how to manage a working DB2 Multisystem solution. Save/restore, availability, and commitment control are among the issues discussed here.

- Chapter 12, “Distributed Queries”

  This chapter provides guidelines on how you can take advantage of DB2 Multisystem to obtain good query performance in a distributed environment.

- Appendix A, “Database Files Used in Examples”

  This chapter provides a description of the database tables used in most of the examples throughout the book.

- Appendix B, “Hash Function Table”

  This chapter provides a table that you can use if you intend to control directly how your data is distributed across a maximum of 32 nodes.

- Appendix C, “PTF List for DB2 Multisystem for OS/400 Feature”

  This chapter provides a list of PTFs that you should install before you use DB2 Multisystem for OS/400.

The Team That Wrote This Redbook

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

Lynda Bovinette is an IT consultant who cooperates with the **International Management Systems** company in Altadena, California. She has wide experience in application development and database design, especially on the AS/400 platform.

Michele Chilanti is a System Engineer at the International Technical Support Organization, Rochester Center. He writes extensively and teaches IBM classes worldwide on AS/400 database, application development and object orientation. Before joining the ITSO 3 years ago, Michele worked in the Field Support Center in Milan, Italy as an AS/400 specialist.

Alexei Pytel is an AS/400 Product Unit Advisory in IBM Russia. He has more than 5 years of experience in this field. He holds a degree in Cybernetics obtained at the Moscow Institute of Physical Engineering. He is an internationally recognized expert on all areas of the AS/400 technology.

Thanks to the following people for their invaluable contributions to this project:

Randy Egan
Rochester Development Laboratory - IBM USA

Tom McKinley
Rochester Development Laboratory - IBM USA

Mark Megerian
Rochester Development Laboratory - IBM USA

Kent Milligan
Rochester Development Laboratory - IBM USA

Carol Ramler
Rochester Development Laboratory - IBM USA

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Database Parallelism on the AS/400 System

This chapter is a general introduction to the subjects discussed in this book. It includes a high level description of the functions that implement database parallelism on the AS/400 platform. This chapter also provides a market positioning for database parallelism.

1.1 Two Flavors of Parallelism: Tightly Coupled and Loosely Coupled

The OS/400 system integrates one of the most successful and functionally competitive database management systems in the industry: DB2 for OS/400. Each AS/400 system installed in the world runs DB2 for OS/400 and virtually every company running its business on an AS/400 system stores its data into DB2 for OS/400 database files.

A recent trend in the computer industry looks at databases in a perspective of building **data warehouses**, that are collections of data specially structured to provide business analysts, sales representatives, marketing specialists and executives prompt access to business information. Among the technical implications of creating efficient data warehouses, we want to focus on the following aspects:

1. Query Performance
2. Scalability

As the size of databases increases, **Query performance** may become a real critical aspect of data processing. Data warehousing environments, for instance, are often based on extremely large databases. Building these warehouses and also retrieving information from them are processes that demand good performance from the query engine.

Scalability can also become a serious issue in modern database environments. Customers who are looking at implementing large data warehouses may be facing the challenge of outgrowing the capacity of a single AS/400 system or the need to run their data warehousing software on the computing power of multiple systems for a single data warehouse.

These two requirements suggest that **parallel database technology** can be considered a valid solution to the needs of those companies that intend to embrace data warehousing in their IT department.

The AS/400 system offers two varieties of database parallelism:

- Symmetric Multiprocessing, which is parallel query processing across multiple CPUs in a multi-processor AS/400 system;
- Loosely Coupled Parallelism, which is parallel database processing across multiple AS/400 systems in a network.

This chapter aims at introducing and positioning these two options that are explained in detail throughout the book.

1.2 Introducing Symmetric Multiprocessing

The main idea behind Symmetric Multiprocessing is to enable the AS/400 query engine to logically split a query into multiple independent tasks which can be carried out in parallel. AS/400 systems with multiple processors are meant to be the primary beneficiaries of Symmetric Multiprocessing, even though parallelism may benefit even single processor systems.

All the available data access interfaces that use the AS/400 query engine may take advantage of Symmetric Multiprocessing. These interfaces are:

- Dynamic or static SQL
- OPNQRYF
- ODBC access to DB2 for OS/400
- Query/400
- Query Management and Query Manager
- Query/38

Native access to database data is not influenced by Symmetric Multiprocessing.

1.2.1.1 Product Packaging and Functions

Symmetric Multiprocessing was first made available as a separate product for Version 3 Release 1 of OS/400. V3R1 users are requested to load an appropriate set of PTFs and then install the Licensed Product 5733-SP1. For more information on installing Symmetric Multiprocessing on systems running V3R1 of OS/400, refer to the Technical Newsletter *Installing Feature Code 1698*, SC41-0609-01.

For V3R2 and for V3R7, Symmetric Multiprocessing becomes a separate chargeable feature of OS/400 (Option 26 of OS/400) that can be installed by using the usual commands for installing AS/400 licensed products.

There is a substantial difference between the contents of the Symmetric Multiprocessing feature for the IMPI releases (V3R1 and V3R2) and for the RISC releases (V3R7). In the IMPI releases, the Symmetric Multiprocessing feature includes the following functions:

- Parallel Table Scan
- Parallel Index Scan
- Parallel Key Positioning
- Hash Group By
- Hash Join
- Index Only Access

We will describe in detail each of these functions later on. At this stage, you may notice that the list includes some functions that are not directly related to parallelism, such as Hash Group By, Hash Join, and Index Only Access. These functions are going to be made available at no extra charge (free) to all the AS/400 users running Version 3 Release 7 through a set of PTFs (SF36173, SF36216, and SF36215). Therefore, the functions packaged into Symmetric Multiprocessing for V3R7 are the following:

- Parallel Table Scan
- Parallel Index Scan

- Parallel Key Positioning

1.2.1.2 Improving Query Performance

Symmetric Multiprocessing has been specifically designed to speed up the execution of some important categories of queries. The first part of this book is meant to illustrate in detail what kind of queries and, therefore, what kind of customers are expected to benefit from Symmetric Multiprocessing. See Chapter 3, “Parallel Table Scan” on page 21 to Chapter 8, “Hash Join” on page 43 to get a full understanding of how to make the most of each individual Symmetric Multiprocessing function.

From a high level perspective, we can affirm that Symmetric Multiprocessing is not meant to speed up every single type of query or SQL statement. In particular, it is very unlikely that transactional applications with embedded SQL will experience any improvement at all. Parallelism and most of the Symmetric Multiprocessing algorithms are much more likely to improve performance of long running queries and of query intensive environments that involve processing of large files like, for example, in data warehousing scenarios. To get more specific guidelines on how to determine whether Symmetric Multiprocessing is beneficial in your environment, refer to Chapter 2, “Introduction to Symmetric Multiprocessing” on page 7.

1.3 Introducing DB2 Multisystem

The DB2 Multisystem feature provides a simple way to distribute your database across multiple AS/400 systems. Access to the distributed database can be done without modifying your current applications and queries.

DB2 Multisystem addresses two different requirements:

- Providing customers additional scalability and performance options for implementing their data warehousing solutions on the AS/400 system.
- Providing a way to consolidate corporate data spread across multiple locations into a single database.

Up to 32 AS/400 systems can be connected in an APPC network and become a single DB2 for OS/400 database. A query can be initiated on any of the systems. DB2 Multisystem broadcasts the query across the network and has it executed in parallel: each of the participating systems execute the query on its own portion of the data. This parallel aspect of DB2 Multisystem can improve the performance of many complex queries.

Scalability is guaranteed first of all by the wide range of models that belong to the AS/400 family. Horizontal scalability is another advantage of DB2 Multisystem. Adding a system to the DB2 Multisystem network is a matter of using a few commands and can be done in a short time. Redistributing your data across the new configuration is also a simple process from an operational point of view.

DB2 Multisystem is supported across all the communications options that support APPC conversation. Local area networks (token-ring and Ethernet), optical connection through OptiConnect/400 and wide area networks are equally supported. As a matter of principle, corporate customers can integrate their multi-site operations into a single corporate-wide database.

Transparency is one of the major advantages provided by DB2 Multisystem: distributed files behave exactly as if they were local database files. Users can run their queries and applications can access the files as if the data were local.

We expect that the majority of the DB2 Multisystem customers are using this product on a local, very fast network and are looking for scalability and performance. Data warehousing is a typical environment where these requirements have a high priority.

Implementing a corporate-wide view of the data in a geographically distributed network is also made possible by DB2 Multisystem. However, we see that this path demands detailed and careful planning to be effectively implemented.

1.3.1.1 Product Packaging and Functions

DB2 Multisystem for OS/400 is a chargeable separate option of the operating system (Option 27). It is available on Version 3 Release 2 for the CISC AS/400 systems and Version 3 Release 7 for the RISC AS/400 systems.

We will talk extensively throughout the book about the way DB2 Multisystem works and about the steps needed to implement solutions based on this product. Here we provide a quick overview of the product architecture.

Before you start distributing your data, you need to connect the systems that are involved into an APPC/APPN network. You then need to create a *Node Group*, that is, an OS/400 object containing:

- References to the participating AS/400 systems.
- A *partition map* that influences the way a database file is distributed across the network. See Section 9.1.3, “Partition Map” on page 51 for more information on partitioning.

The maximum number of systems allowed in a node group is 32. Each system in the network is called a node within the node group.

In the same network, you can define many different node groups and the same system can belong to more than one node group at the same time.

When you create a distributed file, you indicate the node group across which the records are spread. The file description is created on every node in the node group and every node contains a portion of the distributed file.

Since you can create as many node groups as you need, you can distribute different files according to different criteria (using different partition maps) across the same number of physical systems. When you create the distributed file, the partition map is copied into the file description. If you need to change the way the records are distributed, that can be done in two simple steps:

- Change the partition map in the node group.
- Change the physical file (using *CHGPF* for instance) and specifying the node group name in the appropriate parameter.

  This step triggers the redistribution of records.

When planning for DB2 Multisystem, you need to consider several factors:

- Communications and physical connections
- AS/400 size - uni-processors versus multiprocessors

- Amount of memory
- Available auxiliary storage
- The distribution of the files
- Structure and complexity of queries
- Access method of the files
- Others

We elaborate on the impact these factors have against processing in the following chapters.

1.3.2 When To Use DB2 Multisystem

There are several reasons why you may want to use a DB2 Multisystem:

- If your data warehouse contains a large amount of data and you have reached maximum capacity, DB2 Multisystem delivers a simple solution for expanding your system horizontally.
- To improve performance of complex queries through parallelism. Not every query, of course, is going to improve. Remember that your systems are connected with a communication link and even OptiConnect introduces an overhead. If your queries are primarily performing summarizations over large files (processing a lot of records to return a relatively small result set), that is where DB2 Multisystem can greatly help with performance. Many data warehousing implementations have exactly this type of requirement.
- To facilitate user access to corporate-wide distributed files for specific reporting purposes. For instance, the headquarters can process sales reports valid across the entire corporate database.
- When there is a need to add a new remote location, the process of adding a new AS/400 system to your configuration is greatly simplified. Minimal changes, if any, in your applications are necessary.

Chapter 2. Introduction to Symmetric Multiprocessing

This chapter introduces the various functions provided by the Symmetric Multiprocessing for OS/400 feature. Some techniques on evaluating the benefits of installing SMP on your system are also discussed here.

2.1 What Does Symmetric Multiprocessing Do?

The execution of some database queries can be logically subdivided into multiple tasks that can be carried out at the same time. The main function provided by Symmetric Multiprocessing is to enable the AS/400 query optimizer to implement those queries through parallel processing. With Symmetric Multiprocessing the optimizer has the ability of scheduling multiple VMC tasks that can work in parallel to implement the execution of a single database query. The results collected by the various tasks are consolidated into a coherent result set by a coordinator task and given back to the end user or to the requesting application.

2.1.1 Symmetric Multiprocessing and I/O Parallelism

In order to better understand what Symmetric Multiprocessing really is, we compare it with I/O parallelism, a function of OS/400 introduced in Version 3 Release 1 and available to all of the AS/400 customers.

I/O parallelism can be used during the execution of queries to improve performance in bringing the necessary records into main storage. For example, if a query requires a complete scan of a large table, the various disk extents where the table is spread can be brought into memory in parallel. Once the extents are located in main storage, the real query processing, such as record selection, column derivation, and so on, can take place. This processing can still take place serially. No parallelism is involved at this level. The only parallelism consists of populating main storage with data to be processed.

With Symmetric Multiprocessing, parallelism is extended to the actual data processing. Not only can different data extents be transferred into main storage in parallel, but at the same time they are also processed by multiple parallel tasks. Each task produces a fragment of the result set. A coordinating task pulls everything together into a single result set. The major difference with I/O parallelism is that Symmetric Multiprocessing enables parallel processing to also perform operations such as:

- Record selection
- Calculations of derived fields
- Calculations of column functions

2.1.2 Symmetric Multiprocessing and Work Management

Once the Symmetric Multiprocessing feature is installed on your system, the IPL process starts up to 128 tasks that are ready to accept work from the query optimizer whenever parallel implementation is chosen. The tasks are prestarted and waiting on some sort of work queue, so that the user′s job does not experience any overhead for activating the tasks.

Using the *WRKSYSACT* command of the Performance Tools Licensed Program, you can monitor the activity of these tasks that can be identified by their name (DBL3xx). See also Figure 1 on page 9 for a graphical representation of the task scheduling mechanism.

It is important you understand that there is a single set of DBL3xx tasks on the system and not one per job. Application jobs use some of the available tasks by posting their requests on the work queue. We see later on how you can control how many tasks are used by a job when the optimizer chooses parallel implementation. The total number of available tasks for a system may vary depending on the size of the main storage, the CPU model, the number, and type of DASD actuators.

These tasks inherit the priority of the job for which they are currently working. They do not inherit other characteristics, such as the time slice. In fact, time slicing does not apply at all to the DBL3xx tasks. This means that once one of these tasks is allowed to occupy the CPU, it will not release it until it either has to wait for an I/O operation to complete or it has completed its execution. This consideration has to be taken into account to evaluate the impact of parallel query processing on other jobs. See also Chapter 3, “Parallel Table Scan” on page 21 for more information on this subject.

2.2 Supported Interfaces

Every data access interface that uses the AS/400 query optimizer to access database data may take advantage of Symmetric Multiprocessing. In detail, these interfaces are:

- Imbedded SQL (static or dynamic)
- Interactive SQL
- Open Query File (*OPNQRYF*)
- Query Management
- Query Manager
- Query/400
- Query/38
- Incoming DRDA requests
- Incoming ODBC requests
- Remote SQL (Client Access)

Many products available on the AS/400 platform use some of these interfaces to access data on the system. For example, OfficeVision/400 runs queries to search for documents, Client Access can run queries on the AS/400 system when the user executes a file transfer request, and so on. These functions may be carried out by Symmetric Multiprocessing algorithms if you install Symmetric Multiprocessing on the system and you activate it on a system wide basis. Keep this in mind when you plan for using Symmetric Multiprocessing on your system and when you are evaluating the impact of Symmetric Multiprocessing on the overall system performance.

In Figure 1 on page 9, you can look at the architecture of the Symmetric Multiprocessing implementation on the AS/400 system.

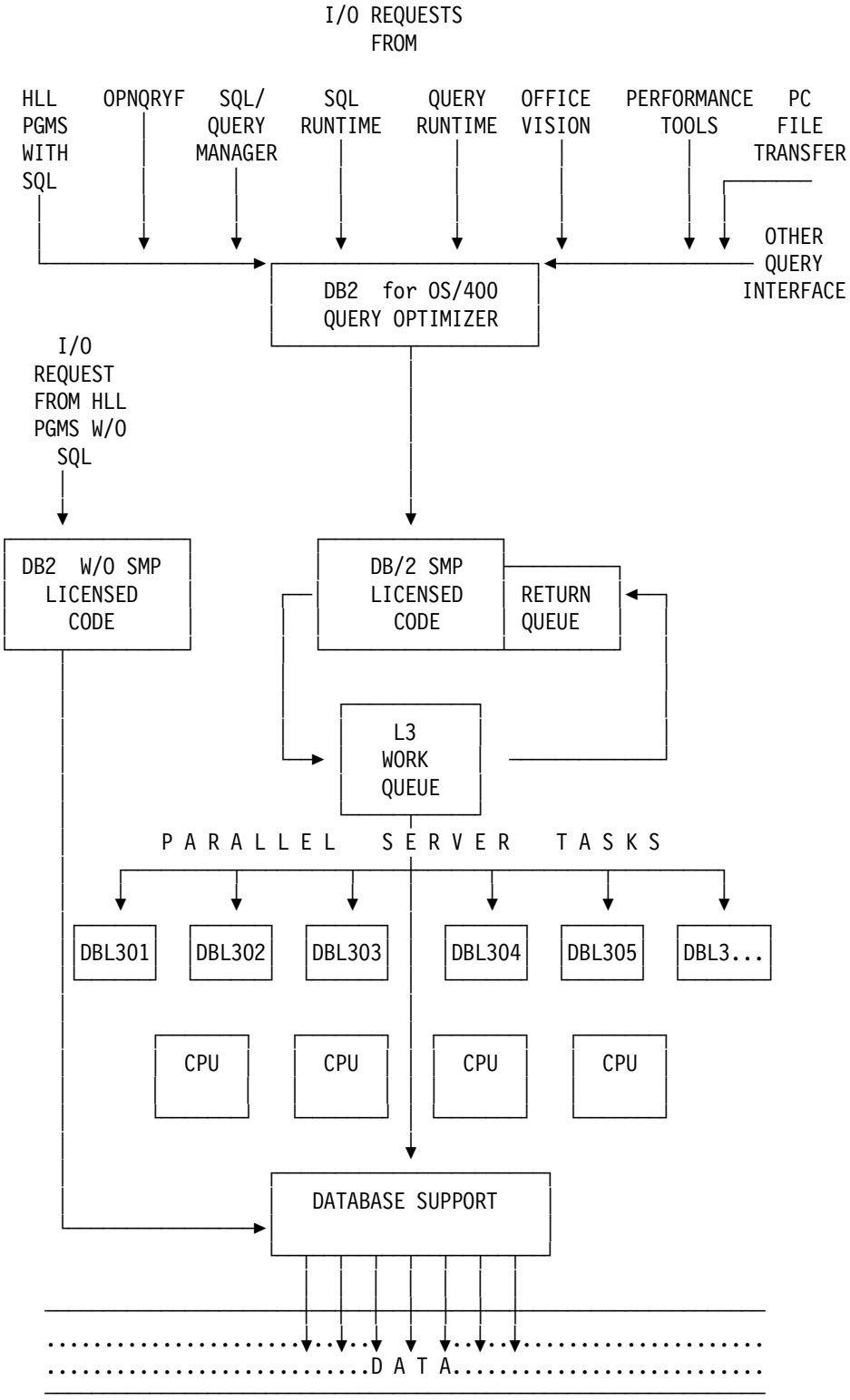


Figure 1. Symmetric Multiprocessing Architecture

2.2.1 Activating Symmetric Multiprocessing

Users have direct control on whether or not the optimizer is allowed to consider parallelism when determining the data access method for a query. However, it is important for you to remember that you cannot force the optimizer towards utilizing a specific access method. The query optimizer always makes decisions based on cost estimates. When you install and activate Symmetric Multiprocessing on your system, you only enable the optimizer to consider the Symmetric Multiprocessing algorithms as a way of executing a query.

Once Symmetric Multiprocessing is installed on an AS/400 system running either V3R1 or V3R2, the optimizer starts considering *Hash Group By*, *Hash Join*, and *Index Only Access* as possible ways to implement a query. These algorithms become part of OS/400 itself in V3R7.

Users can control the *degree of parallelism* with which queries can be implemented. More precisely, users can influence the maximum number of tasks that the optimizer is allowed to schedule to resolve queries on the system. You either have the option of controlling parallelism at the system level or at the job level.

Parallelism at the **system level** can be controlled by setting the appropriate value for the **QQRYDEGREE** system value. By default, all jobs inherit the degree of parallelism specified for the system value.

The degree of parallelism can also be tailored on a **per job basis** by using the *CHGQRYA* (Change Query Attribute) CL command. This command allows to specify the degree of parallelism in its *DEGREE* parameter.

The values that are allowed for the QQRYDEGREE system value are in the following list with an explanation of their meaning.

- ***NONE**: No parallelism is allowed. The optimizer can still use Hash Group By, Hash Join, and Index Only Access, but no parallel tasks are scheduled.
- ***OPTIMIZE**: Parallelism is enabled and the optimizer determines the number of parallel tasks and the amount of memory that can be utilized to execute a query. This value tells the optimizer to use parallelism without heavily affecting other jobs on the system. In particular, the optimizer looks at the size of the memory pool where the query is running and reserves a chunk of that pool equal to the entire pool divided by the activity level. The number of parallel tasks to be scheduled depends on many factors among which the size of the memory, the disk configuration, the processor performance and the number of processors play a significant role.
- ***MAX**: Parallelism is enabled and the optimizer is allowed to be ″more aggressive″. In practice, the optimizer acts to double the amount of resources (memory and number of tasks) that it would allocate to the query if the value was **OPTIMIZE*.
- ***IO**: This value enables what we call I/O parallelism. This type of parallelism is not a part of the Symmetric Multiprocessing feature and has been available as part of the operating system since V3R1.

For the DEGREE parameter of CHGQRYA, you can also specify:

- ***NBRTASKS**: This value allows you to directly specify the number of tasks that you want the optimizer to use for parallel processing. The maximum number of tasks you can specify here is 99. It must be said that it is

extremely difficult to predict what the effect of choosing a certain number would be without actually running a test. This option might be used to force the optimizer to be even more aggressive than with *MAX or to set an intermediate degree of parallelism between *MAX and *OPTIMIZE. Remember that even when you specify DEGREE(*MAX), the optimizer still takes into some account the total memory size the activity level of the memory pool where the query will run. Therefore, the number of tasks that is selected may not result aggressively enough for your needs.

There are a couple of additional considerations that you should keep in mind if you choose to specify a number of tasks. First, if you specify a high number of tasks or a number of tasks that are significantly higher than the one chosen by the optimizer with *MAX, you increase the probability that the various tasks start competing with each other to acquire resources such as disk actuators. In this case, you risk defeating the purpose of parallelism and achieve poorer performance than with a lower number of tasks.

Secondly, if too many tasks are specified, there is an increased probability that a number of them are not really used or that their job completes quickly, resulting in additional overhead for very little gain in performance.

We recommend you use *NBRTASKS for selected queries and only after testing them on the system with various numbers of tasks to make sure you achieve your objectives.

- ***SYSVAL**: The job picks the value specified in the system value.

It is crucial you understand the implications of using the QQRYDEGREE system value to activate Symmetric Multiprocessing. In an environment where queries and transactional jobs coexist, enabling parallelism on a system wide basis may result into privileging query activities over users performing interactive transactions. Also keep in mind that the DBL3xx tasks are not time-sliced as we explained in Section 2.2, “Supported Interfaces” on page 8. These considerations must be kept in focus especially if you choose to set the system value to *MAX.

In addition, remember that Symmetric Multiprocessing may apply to products and interfaces that normally are not associated to query activities, such as OfficeVision/400 or the Client Access file transfer functions. Using the system value propagates the enablement to parallel query processing to these functions also and you need to be sure that you really want to privilege them over other activities happening on the system.

When you buy and install Symmetric Multiprocessing, you may be tempted to unleash its power by setting QQRYDEGREE to *MAX. Before you operate this way, make sure that your environment can tolerate occasional situations where system resources are heavily utilized by query activities.

2.2.2 Activating SMP in Client/Server Environments

In a client/server scenario, where the AS/400 system plays the role of the server, you may face the need of activating parallelism for queries that are initiated by the client system. In this section, we point out some techniques that allow you to control Symmetric Multiprocessing in some of the most popular database serving architectures. In any client/server scenario, the major challenge related to Symmetric Multiprocessing is that you may need to execute the **CHGQRYA** command for a specific server job if you just cannot let every job refer to the QQRYDEGREE system value.

2.2.2.1 Symmetric Multiprocessing and ODBC

The ODBC driver provided with Client Access allows your ODBC queries to take advantage of Symmetric Multiprocessing algorithms. If you are using some non-IBM ODBC driver, check with your provider to verify if the driver meets the requirements to support Symmetric Multiprocessing.

In order to tailor the degree of parallelism for the various ODBC clients, you need to execute the CHGQRYA command for the ODBC server jobs (QZDAINIT). The AS/400 registration information facility provides a way to do that at connect time. By using the *WRKREGINF* on the AS/400 system you can register an exit program for the ODBC "exit point".

```
                      Work with Registration Information

Type options, press Enter.
  5=Display exit point   8=Work with exit programs

                            Exit
     Exit                   Point
Opt  Point                  Format    Registered  Text
__   QIBM_QTMT_WSG          QAPP0100     *YES     WSG Server Sign-On Valid
__   QIBM_QVP_PRINTERS      PRNT0100     *YES     Original Virtual Print S
__   QIBM_QWT_PREATTNPGMS   ATTN0100     *YES     Preattention program exi
__   QIBM_QWT_SYSREQPGMS    SREQ0100     *YES     Presystem request pgm ex
__   QIBM_QZCA_ADDC         ZCAA0100     *YES     Add Client exit point
__   QIBM_QZCA_REFC         ZCAF0100     *YES     Refresh Client Informati
__   QIBM_QZCA_RMVC         ZCAR0100     *YES     Remove Client exit point
__   QIBM_QZCA_SNMPTRAP     ZCAT0100     *YES     SNMP trap routing exit p
__   QIBM_QZCA_UPDC         ZCAU0100     *YES     Update Client Informatio
__   QIBM_QZDA_INIT         ZDAI0100     *YES     Database Server - entry
__   QIBM_QZDA_NDB1         ZDAD0100     *YES     Database Server - data b
                                                                    More...
Command
===> ____________________________________________________________________
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
```

Figure 2. Work with Registration Information

In Figure 2, you can see all of the available exit points for which you can define an exit program. *QIBM_QZDA_INIT* is related to ODBC and the correspondent exit program is executed at connect time. You can write and register an appropriate exit program that includes an invocation of CHGQRYA with the desired DEGREE parameter. The exit program can recognize the name of the client and the user ID of the connecting job and act accordingly. For more information on how to write and register an exit program, you can refer to *OS/400 Server Concepts and Administration Manual*, SC41-3740. No changes to the client code are required in order to use this facility.

2.2.2.2 Symmetric Multiprocessing and DRDA

If your clients are using DRDA to access DB2/400 data, you can use stored procedures to activate parallelism at the server side. A stored procedure is a program that can be called by the client by means of the SQL CALL statement and that are executed on the server. The stored procedure contains the appropriate invocation of the CHGQRYA command.

This technique implies that you take the explicit action of calling the stored procedure to enable parallelism on the AS/400 system and maybe suitable only when the queries are run from within a program. This technique has the drawback of not providing an automatic mechanism of activation for Symmetric Multiprocessing, but it has the advantage of letting the requester determine the degree of parallelism. ODBC applications can also use stored procedures.

2.3 Symmetric Multiprocessing Algorithms: An Overview

As we already mentioned, the objective of Symmetric Multiprocessing is to improve the run time of some specific categories of queries by using new data access methods. Your queries should meet some important requirements to be suitable for being implemented by Symmetric Multiprocessing access methods. In this section, we discuss those requirements and we also provide an overview of the Symmetric Multiprocessing access methods. We recommend you read *DB2 for OS/400 SQL Programming*, SC41-4611, for a detailed description of all the available access methods.

Symmetric Multiprocessing introduces the following access methods:

- Parallel Data Space Scan
- Parallel Key Row Positioning
- Parallel Key Row Selection
- Index Only Access (free (no charge) with an OS/400 PTF in V3R7)
- Hash Group By (free (no charge) with an OS/400 PTF in V3R7)
- Hash Join (free (no charge) with an OS/400 PTF in V3R7)

The chapters immediately following this one describe the Symmetric Multiprocessing access methods in detail.

These access methods can only be used if queries have the following characteristics:

- Backward scrolling is not required.
- Live data is not necessary.
- You want to minimize the total run time and you are not interested in minimizing the time it takes to provide the ″first page of results″.
- The query or SQL statement are read-only.

In order to ensure that the previous conditions are met, follow these guidelines:

- Always specify *ALWCPYDTA(*OPTIMIZE)* when you create your SQL programs, when you use Interactive SQL, or when using OPNQRYF. This parameter setting lets the optimizer take a copy of the result set into a temporary file in order to achieve better performance. The default value is *YES and that means that the optimizer can create a temporary result file only if it is strictly necessary for resolving the query. An implicit consequence of choosing *OPTIMIZE is that your result set might contain live data.
- When using OPNQRYF, specify *OPTIMIZE(*ALLIO)*; when using SQL, include the *OPTIMIZE FOR n ROWS* clause in your queries. Both these techniques tell the optimizer that it should try to minimize the **total** query run time versus the time it takes to bring up the first page of results.

- If you are using Interactive SQL, set the *REFRESH* parameter to **FORWARD*. With this setting, you tell Interactive SQL that when you scroll back, you do not need to get live data. In this case, Interactive SQL copies the result set as it is produced and when you request to scroll back, you get the results as they are in the temporary file.
- If you need to take advantage of Symmetric Multiprocessing to speed up query processing in applications running under commitment control, limit the isolation level to ***CHG** and use the option **ALWBLK(*ALLREAD)**. See Section 2.4.4, “Symmetric Multiprocessing Inhibitors and Limitations” on page 19 for more information on these parameters.

2.3.1 Considerations on Ordering Result Sets

Even before Symmetric Multiprocessing was available, SQL and the query engine do not guarantee that records are returned in any particular sequence unless some ordering criteria is specified. As a matter of fact, on a system where Symmetric Multiprocessing is not installed, you may notice that queries accessing the files sequentially return results ordered according to the physical sequence of the records in the files. In addition, queries with a *Group By* clause require an index and the result set is ordered according to the columns specified in the Group By clause.

If Symmetric Multiprocessing algorithms are used, the chances that this still happens are really low. When parallelism is used, every parallel task returns portions of the result set. The order with which these portions are returned is unpredictable. You should **never** assume that query results are in any particular sequence unless you explicitly specify an ordering clause.

On the other hand, if you install the Symmetric Multiprocessing feature, you may notice that *ORDER BY* type of clauses tend to bias the optimizer towards using an existing index, if one is available. This implementation does not allow parallelism to be used. We, therefore, recommend that you specify ordering criteria only if it is strictly necessary.

If the optimizer cannot locate a permanent index that has been built over the ordering columns, it estimates whether it is more convenient to build a temporary index or whether a table scan followed by a sort is faster. If the file is large, the optimizer probably declines to build an index. In this case, the smaller the result set, the more effective and beneficial Symmetric Multiprocessing will be. If the result set is large, the most time is spent in sorting the result set and Symmetric Multiprocessing cannot improve that piece of the processing.

In other words, queries such as:

```
SELECT LASTNAME, FIRSTNAME, SSNUMBER
 FROM CITIZENS
  ORDER BY LASTNAME
```

probably do not have much benefit from Symmetric Multiprocessing because the final sort is going to be expensive, whereas queries with more specific selection criteria, such as:

```
SELECT LASTNAME, FIRSTNAME, SSNUMBER
 FROM CITIZENS
  WHERE BIRTHDATE = '1963-07-27'
   ORDER BY LASTNAME
```

may improve significantly.

2.4 Will Symmetric Multiprocessing Help?

Symmetric Multiprocessing may improve your query performance thanks to the new access methods and to parallel data processing.

However, you should keep in mind that Symmetric Multiprocessing is applicable to certain categories of queries and that the extent of the benefit may vary broadly, depending on some unique characteristics of each database environment.

This section provides an overview of the tools and the methods that you can use to determine whether or not Symmetric Multiprocessing may be beneficial in a certain environment. Based on our experience in testing Symmetric Multiprocessing, we feel that we are not in a suitable position for defining a methodology to determine what the extent of the improvement is. The system offers several indicators and tools to establish whether Symmetric Multiprocessing can help with query performance.

2.4.1 Macroscopical Indicators

It is worth reminding you once again that Symmetric Multiprocessing only applies to query processing. If you only use native database I/O, you can disregard Symmetric Multiprocessing as a solution for your performance problems.

Multiprocessor systems are the best candidates for taking advantage of parallel processing. On a multiprocessor system, more than one task at a time can be running to carry out the execution of a single query. On a single processor system, the advantages of parallelism are limited to those situations where a task that is waiting for an I/O operation can leave the CPU to another task which is ready to process the data.

In other words, on a uni-processor system you can hope that Symmetric Multiprocessing transforms an **I/O-bound query into a CPU-bound query**, but you cannot hope for large improvements if your queries are already CPU-bound. On a system with multiple processors, Symmetric Multiprocessing can be beneficial in both cases. Actually, one of the major indications that you should consider Symmetric Multiprocessing is given by the CPU utilization in a multi-processor system. If you notice that the CPU workload tends to concentrate on a single processor while heavy queries are executed, that is a definite sign that Symmetric Multiprocessing may help in optimizing the usage of your system resources. See Section 2.4.3, “Gathering Some More Information” on page 17 to understand how to investigate the utilization of your system resources.

In general, Symmetric Multiprocessing requires that your system have some CPU cycles available to accommodate parallel processing. On a fully utilized system there is little hope that Symmetric Multiprocessing can be beneficial. There are two exceptions to this consideration:

1. On a fully utilized system, you may want to privilege some query activities over other jobs. For example, at night your system runs four different batch jobs in parallel. One of them is supposed to print an important report by running a sequence of complex queries. Even if the overnight CPU utilization is close to 100%, you can still use Symmetric Multiprocessing to speed up the execution of the query intensive batch job. In this case, you are likely to experience a longer execution time for the remaining three batch jobs.

2. On an IMPI system (V3R1 or V3R2), Symmetric Multiprocessing might be beneficial not only for its parallel algorithms, but also because it introduces Hash Group By, Hash Join, and Index Only Access. Using these algorithms instead of the traditional ones might lower the CPU time required to execute some categories of queries.

The Symmetric Multiprocessing feature mainly targets **long running queries**. If users run queries whose duration frequently exceeds one minute, that is a good indication that you may need to install Symmetric Multiprocessing. It is unlikely that Symmetric Multiprocessing can produce substantial improvements if the average run time of your queries remains well below one minute.

If your queries are directed towards **large database files**, it generally is a good indication that Symmetric Multiprocessing may be of great help. The definition of large file depends on the size of your system, but, in general, you should start considering Symmetric Multiprocessing when the size of your files exceeds 100 000 records.

2.4.2 Balancing System Resources for Symmetric Multiprocessing

A **balanced hardware configuration** is also important if you want to take full advantage of Symmetric Multiprocessing. You should make sure that your DASD devices can feed the CPU fast enough with fresh data extents; otherwise your queries always tend to remain I/O bound. An empirical test you can run on your system to verify if it has adequate DASD devices on your system is the following test:

1. Create two physical files by issuing the following commands:

   ```
   CRTPF FILE(filein) RCDLEN(100)
   CRTPF FILE(fileout) RCDLEN(100)
   ```

2. Initialize the input file (*filein*) with a relatively large number of records, as in the following example:

   ```
   INZPFM filein TOTRCDS(100000)
   ```

3. Force the entire file out of main storage:

   ```
   SETOBJACC filein *FILE POOL(*PURGE)
   ```

4. Activate parallelism:

   ```
   CHGQRYA DEGREE(*MAX)
   ```

5. Now issue the following command:

   ```
   OPNQRYF FILE(filein) QRYSLT('1=2') ALWCPYDTA(*OPTIMIZE)
   ```

   Clearly, this query does not return any records, but the optimizer must scan the entire file sequentially.

6. The following command causes the query to be executed and the parallel table scan algorithm is used:

   ```
   CPYFRMQRYF filein TOFILE(fileout)
   ```

7. While the previous step is being performed, monitor the CPU utilization using *WRKSYSSTS*. The test should run when the system is lightly loaded or, even better, in dedicated mode.

8. If you notice that the CPU utilization averages around 80% during the query execution, you can conclude that your system configuration is well balanced between CPU speed and DASD efficiency. If you notice that the CPU utilization is significantly lower than 80%, you should consider either

upgrading your disks to faster models or increase the number of actuators. If the CPU utilization is well above 80%, possibly you cannot fully benefit from Symmetric Multiprocessing because your CPU might not be fast enough to process the data provided by the DASD peripherals in an effective way.

An **even distribution** of the data belonging to large database files is also essential to obtain the most benefits from Symmetric Multiprocessing. If a large database file is spread across a limited number of actuators, there is not much that parallelism can do to improve queries performance. In fact, you might even experience some performance degradation caused by the fact that the parallel tasks tend to compete with each other to gain control of the same DASD actuator. This phenomenon may result in a sharp increase of the DASD service time and into worse performance.

There is no direct way or command by which users can check how a database file is distributed across the various disks. You can resort to using simple tests to infer this information. For example, you can run an extensive table scan and monitor the activity of your disk drives during the test by using the *WRKDSKSTS* command. The test should run in dedicated mode and you should notice that most actuators show the same degree of activity during the table scan. If you notice that most I/O operations are performed by a limited subset of your actuators, probably your data is distributed unevenly. These situations are common especially after you added new DASD to your system and at the same time some of your files grew significantly as a result of your applications′ activities.

Saving and restoring a large library or a set of libraries causes the system to redistribute the objects and you are likely to achieve a more even distribution.

2.4.3 Gathering Some More Information

The access methods the optimizer selects to execute queries depend on many factors, ranging from the number of records in the files, the estimated size of the result set, the estimated distribution of key values in the existing access paths, grouping, ordering and joining criteria, and so on. In general, it is not easy to predict what the optimizer′s choice is and it is not possible to force the optimizer to make a certain decision. It is, therefore, not easy to predict whether and how much Symmetric Multiprocessing helps on your system. In the previous section, we discussed some high-level indicators that can point you towards installing Symmetric Multiprocessing. In this section, we discuss some of the tools you can use to gather information that can help you determine if your query performance can improve by installing Symmetric Multiprocessing.

2.4.3.1 Performance Monitor and Performance Tools

The Performance Monitor is a function of OS/400 that allows you to gather a wide set of performance data on system components, jobs, and application programs. Once you have collected the data for a period of time, you can use the Performance Tools licensed product to print numerous reports that present the data at different level of details. Refer to *Performance Tools/400 V3R6*, SC41-4340, if you need more information on the Performance Tools.

Performance Tools provide a lot of information that is relevant to our investigation about Symmetric Multiprocessing. Here we intend to highlight what we think is more important:

- The *Component Report* in the section called *"*Component Interval Activity*"* is a good starting point to determine if your system has some spare CPU cycles to devote to parallel processing. You can also subset the analysis to the jobs that are responsible for the query workload and evaluate how large the opportunity for Symmetric Multiprocessing is in terms of available CPU processing power.
- On multiprocessor systems running V3R1 and V3R2, the *WRKSYSACT* command, which is part of Performance Tools, shows the CPU utilization of each individual processor. You can also direct the output of the command to a database file and then run *PRTACTRPT* to print a formatted report. You can sample the system for several minutes periodically during the day or whenever you know that queries are run and determine how well balanced the CPU utilization is across the various processors.
- The *Performance Explorer* can provide you with an important piece of information. When queries require large tables to be sequentially scanned, the OS/400 module to use is **QDBGETM**. The Performance Explorer can be used to determine if a significant part of your query processing is taken by QDBGETM. If this is the case, it is probable that Symmetric Multiprocessing helps thanks to its parallel processing.

  The Performance Explorer is part of Performance Tools in V3R6. On a V3R1 or V3R2 system, use the Timing and Paging Statistics Tool (TPST) to perform the analysis previously described.

2.4.3.2 Database Performance Monitor

This tool has been introduced into OS/400 with Version 3 Release 6 and it is available at no charge (free) on all of the RISC AS/400 systems. The Database Performance Monitor can be used to collect information by the way queries are implemented by the optimizer. You can target an individual job or let the tool collect information on a system-wide basis. Refer to *DB2 for OS/400 Database Programming*, SC41-4701, for a complete description of this tool.

This tool can be extremely useful in our investigation on the applicability of the Symmetric Multiprocessing algorithms because it allows you to analyze the structure of the queries that are run on your system. Here is a sample of the information you can extract:

- You can determine whether the users are running queries that are implemented by means of sequential access. This information is key to establish if there are opportunities for using Parallel Table Scan.
- You can find out which access paths are utilized and the circumstances that determine their use. You can understand if Parallel Index Scan or Parallel Key Positioning is applicable.
- Information on queries that perform grouping or join operations can also be retrieved. If you determine that joining large files together or performing *Group By* selections on large tables happens frequently, then Symmetric Multiprocessing may improve the performance of those queries by combining the hashing algorithms with Parallel Table Scan.
- Information on the ordering criteria used in your queries is also available. Remember that explicit ordering clauses in your queries (such as ORDER BY clauses in SQL queries) may reduce the advantages of using Symmetric Multiprocessing algorithms.

2.4.3.3 Using the Debugger

The debugger provided by OS/400 offers a way to analyze the decisions of the optimizer for an individual job. Just issue:

```
STRDBG UPDPROD(*YES)
```

and the optimizer sends a number of messages to the job log during the execution of queries. You do not need to take any other action nor do you need to change the way you normally operate to run your queries.

The information in the job log allows you to understand what access method was used and why. The same amount of information that you can extract with the Database Performance Monitor can be obtained this way on an individual job basis on any AS/400 system. If you need to submit a batch or a server job to the debugger analysis, you should first service that job by issuing an appropriate STRSRVJOB and then use the STRDBG command.

The debugger is useful if you already have the Symmetric Multiprocessing feature and you want to test some selected queries in order to determine the optimal degree of parallelism. Once the debugger is activated, the optimizer logs messages reporting whether parallelism is used and how many parallel tasks are used. See Section 3.3, “Examples” on page 22 for an example and some more considerations on this subject.

2.4.3.4 Running Print SQL Information (PRTSQLINF)

For SQL queries embedded into AS/400 programs, service programs, or SQL packages, you can gather information on their implementation and on their structure by using the PRTSQLINF command. This command prints the SQL access plan of the object you are analyzing, which, in turn, contains information about the access methods chosen by the optimizer for implementing the embedded SQL statements. The access plan is stored only for static SQL statements; this command is useful if your queries are written in a static SQL statement contained in a program. This happens frequently in batch programs that are meant to produce statistical reports on a large amount of data.

2.4.4 Symmetric Multiprocessing Inhibitors and Limitations

We already mentioned earlier that there are factors that can prevent the optimizer from choosing Symmetric Multiprocessing algorithms. Here is a summary of all the aspects that may limit the applicability of Symmetric Multiprocessing.

- Update Capability: Symmetric Multiprocessing algorithms are available for **read-only** processing. SQL cursors that are capable of updating cannot enjoy any advantage from Symmetric Multiprocessing.
- Commitment control: Most of the new algorithms do not work with commitment control. There is a remarkable exception to this rule: if you use the *CHG isolation level and you specify **ALWBLK(*ALLREAD)**, your queries can still exploit the Symmetric Multiprocessing algorithms because read-only cursors opened with these options are not placed under commitment control.
- Using scrollable cursors: If, in your applications, you reposition the SQL cursors using the scroll functions, parallelism is not used for those cursors. The only cursor positioning that is compatible with parallelism is the **FETCH NEXT** operation.

- Using subselect processing: Symmetric Multiprocessing algorithms cannot be used to resolve queries containing subquery processing. See Section Chapter 8, “Hash Join” on page 43 for more information on optimizing subqueries with Symmetric Multiprocessing.
- Index join implementation: If the optimizer selects to process a join query accessing the secondary files through an index, parallelism is not used. Parallel Table Scan can be used when the join is implemented using Hash Join. Not having a permanent index that matches the join columns increases the chances that Hash Join is used.

It is also useful to remind you that some resource-intensive database activities do not benefit at all from Symmetric Multiprocessing. These activities include:

- Building permanent or temporary indexes: On V3R1 and V3R2 systems, the Symmetric Multiprocessing feature may reduce the necessity for creating some temporary indexes thanks to Hash Group By and Hash Join.
- Sorting database files with *FMTDTA* or sorting large temporary result tables.
- Any type of access that does not use the query engine.

Make sure that your performance concerns are not primarily related to these activities if you want to take advantage of Symmetric Multiprocessing.

Chapter 3. Parallel Table Scan

The Parallel Table Scan method (also referred to as Parallel Data Space Scan method) can be used by the query optimizer when a query needs to process sequentially a large database file. The optimizer may determine that the query run time can be reduced by allowing multiple tasks to process separate portions of the same table in parallel.

In Section 3.1, “How Parallel Table Scan Works,” we discuss how this method works. Then in Section 3.2, “When Parallel Table Scan Is Beneficial,” we shall see what needs to be considered when using it. Section 3.3, “Examples” on page 22 presents some idea about what kind of improvement can be expected.

3.1 How Parallel Table Scan Works

When Parallel Table Scan is used to perform a large table scan, the optimizer divides the table itself into a number of sections that can be processed in parallel. Depending on the parallelism degree option, the optimizer then schedules a number of tasks for the processing of the various sections. Each task produces a segment of the result set and the various segment is coordinated into a coherent result set. In *DB2 for OS/400 SQL Programming*, SC41-4611, you can find a detailed description of the way Parallel Table Scan works.

Parallel Table Scan can also benefit from the AS/400 *Expert Cache* feature that allows the operating system to recognize I/O patterns in any AS/400 job and anticipate I/O operations. This feature is easy to activate; just specify ***CALC** for the paging option of the memory pool where your jobs run.

Refer to *OS/400 Work Management*, SC41-3306, for a complete description of the Expert Cache function.

3.2 When Parallel Table Scan Is Beneficial

Parallel Table Scan is most effective when:

- A complete table scan of a large database file is needed.
- The file is evenly scattered across multiple actuators.

There is a large range of queries that can take advantage of Parallel Table Scan. Queries that use generic SQL *LIKE* predicates are good candidates for Parallel Table Scan. Consider, for example, the query in Figure 3. This query contains a generic text search condition on the *ABSTRACT* field (*"*Retrieve the titles of books that talk about Shakespeare*"*) that cannot be resolved in any other way but by analyzing every single record.

```
SELECT AUTHOR, TITLE, EDITION_DATE
  FROM LIBRARY
    WHERE ABSTRACT LIKE '% Shakespeare %'
```

Figure 3. A Generic LIKE Clause

Another simple example of a query that can benefit from a Parallel Table Scan is found in Section 3.3, “Examples” on page 22. This method is extremely effective

when it is combined with Hash Join and Hash Group By. In those cases, the performance advantage offered by the hashing algorithms, which resort to a table scan in order to build the hash table, are boosted by Parallel Table Scan. See also Chapter 7, “Hash Group By” on page 37 and Chapter 8, “Hash Join” on page 43 for examples of this synergy.

You get the most out of Parallel Table Scan if your queries run on a multiprocessor system. That is where several server tasks can truly work in parallel. Even with uni-processor AS/400 models, Parallel Table Scan can be of benefit especially if your queries are I/O bound. See Chapter 2, “Introduction to Symmetric Multiprocessing” on page 7 for more considerations about parallelism on single processor systems.

Parallel data space scan requires memory to buffer the data that is retrieved in each task. A typical amount of memory needed for each task is about two megabytes. Queries that process tables with variable length fields or that derive longer new fields might need additional memory.

When several tasks are working in parallel and return data to a coordinating task in the application′s job, the resulting rows are presented to the application in an unpredictable sequence. See Section 2.3.1, “Considerations on Ordering Result Sets” on page 14 for a complete discussion about ordering your result sets in a Symmetric Multiprocessing environment.

3.3 Examples

In order to give you an idea of what kind of improvement Parallel Table Scan can produce, we performed several tests running an SQL resource intense query under the following conditions:

- Hardware configuration: AS/400 model 500 with 2144 processor feature (uni-processor), 384MB main storage, 14GB DASD space with 16 disk units.
- Software configuration: OS/400 V3R7 with option 26 DB2 Symmetric Multiprocessing feature installed.
- Database configuration (see Appendix A, “Database Files Used in Examples” on page 135 for a description of the table layout):
  - Table LINEITEM contained 1 200 000 records.
- Job was run in 110MB pool (with 110MB *BASE pool and 70MB *MACHINE pool).

We ran the following query:

```
SELECT SUM(PRICE)
   FROM LINEITEM
      OPTIMIZE FOR 999999999 ROWS
```

This query actually uses two Symmetric Multiprocessing algorithms: Hash Group By (see Chapter 7, “Hash Group By” on page 37) and Parallel Table Scan. The query was run with the different values for the DEGREE parameter of the CHGQRYA command. The following results were observed:

Table 1. Parallel Table Scan - Uni-Processor				
	DEGREE (*NONE)	**DEGREE (*MAX)**	**DEGREE(*NBRTASKS 20)**	**DEGREE(*NBRTASKS 99)**
Response time	90 sec	75 sec	45 sec	45 sec
Average CPU utilization	46%	65%	95%	99%
Number of tasks	-	6	10	35

To obtain the number of tasks that were actually used to process your query, run your query in debug mode. In Figure 4, you can look at the message we obtained after running our test query with *DEGREE(*NBRTASKS 99)*.

```
                        Additional Message Information

Message ID . . . . . :   CPI4330       Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   07/31/96      Time sent  . . . . . :   15:28:10
                 1
Message . . . . :   35 tasks used for parallel table scan of file LINEITEM.
Cause . . . . . :   35 is the average numbers of tasks used for a table scan
  of member LINEITEM of file LINEITEM in library A960202B.
    If file LINEITEM in library A960202B is a logical file, then member
  LINEITEM of physical file LINEITEM in library A960202B is the actual file
  from which records are being selected.
    A file name of *N for the file indicates a temporary result file is being
  used.                                                                    2
    The query optimizer has calculated that the optimal number of tasks is 94
  which was limited for reason code 3. The definition of reason codes are:  3
    1 - The *NBRTASKS parameter value was specified for the DEGREE parameter
  of the CHGQRYA CL command.
                                                                      More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 4. Query Optimizer Message for Parallel Table Scan

2 indicates the maximum number of tasks that are used for your query. This number can be an indirect indicator of how many extents your file currently has. **1** shows you how many tasks were actually scheduled. This number shows the **average** number of tasks that were active while the query was run. If **1** is significantly lower than **2**, it can indicate that some of the extents for the file are significantly smaller than others and tasks that were processing these extents finished processing much faster. This consideration applies to all of the parallel Symmetric Multiprocessing algorithms.

For example, if your file has one large extent and many small ones, the average number of tasks is low. You should also examine the reason code in line **3**. It may give you further insight as to how your query is being processed and what can be done to improve the performance of this query.

If the previous message is not produced during the execution of a query that runs after the debugger has been activated, you can conclude that Parallel Table Scan is not being used.

In order to compare the results, the same query was run on a larger AS/400 model:

- Hardware configuration: AS/400 model 530 with 2162 processor feature (four processors), 4096MB main storage, 100GB DASD space with 28 disk units.

The query was run in a 2900MB storage pool (with 890MB in *BASE pool and 380MB in *MACHINE pool). The following results were observed:

Table 2. Parallel Table Scan - Multiprocessor

	DEGREE(*NONE)	**DEGREE(*NBRTASKS 99)**
Response time	40 sec	10 sec
Average CPU utilization	20%	45%
Number of tasks	-	15

As can be seen from the results, even on a uni-processor system, the query run time can be reduced by half. The best result was achieved with 10 tasks. When going above 10, the system became saturated and although additional tasks were scheduled, the uni-processor could not further benefit from additional parallelism. On the multiprocessor system, further improvement can be achieved. More server tasks can run in parallel on different CPUs that allow query to run four times faster.

The number of tasks scheduled for the query for the multi-processor system happened to be less than for the uni-processor (15 versus 35). This can be explained by various factors:

- Disk units of larger capacity were installed on the multiprocessor system. As a result, the table could have been spread over a smaller number of units.
- When calculating the number of tasks to be used, the optimizer takes into account many factors that may be different in each environment such as:
  - The number and relative performance of CPUs available.
  - The amount of storage available for the job.
  - The number and performance of disk units holding DASD extents for the table.

Some of these factors are volatile so if you run the same query many times even on the same system, you can expect slightly different results.

Chapter 4. Parallel Index Scan

This algorithm can be beneficial whenever a query needs to perform a complete scan of an index that is built over a large database table. The query optimizer may decide that query run time can be reduced by partitioning the index into a number of portions that can be scanned in parallel.

In Section 4.1, “How Parallel Index Scan Works,” we discuss how this method works. Then in Section 4.2, “When to Use Parallel Index Scan,” we shall see what you have to consider when using it and Section 4.3, “Examples” on page 27 gives you some idea of what results you can expect.

4.1 How Parallel Index Scan Works

The book *DB2 for OS/400 SQL Programming*, SC41-4611, includes a detailed description of the way Parallel Index Scan works. Most of the considerations that apply to Parallel Table Scan also apply to Parallel Index Scan (see Section 3.1, “How Parallel Table Scan Works” on page 21).

Remember that this method may also benefit from using the Expert Cache feature, especially if the index itself is extremely large.

4.2 When to Use Parallel Index Scan

The typical queries that can benefit from Parallel Index Scan generally include selection criteria based on ranges of values, such as SQL queries containing BETWEEN or LIKE predicates in their search conditions. Consider, for instance, the following SQL query:

```
CREATE INDEX X1 ON EMPLOYEE (LASTNAME, DEPARTMENT)

SELECT FIRSNAME, LASTNAME, POSITION
  FROM EMPLOYEE
    WHERE LASTNAME BETWEEN 'A' AND 'AZ' OR
       LASTNAME BETWEEN 'L' AND 'LZ' OR
         LASTNAME BETWEEN 'T' AND 'TZ'
           OPTIMIZE FOR 999999999 ROWS
```

In the previous example, the optimizer can choose to use Parallel Index Scan and to schedule three tasks for the parallel processing. Each task might scan a portion of the index X1 that corresponds to the range specified by a different BETWEEN predicate. If the environment allows a higher degree of parallelism, then each partition can be further split.

Similar to a BETWEEN predicate is the **LIKE** clause as it is demonstrated by the following example:

```
CREATE INDEX X1 ON EMPLOYEE (LASTNAME, DEPARTMENT)

SELECT FIRSTNAME, LASTNAME, POSITION
  FROM EMPLOYEE
     WHERE LASTNAME LIKE 'VAN%'
        OPTIMIZE FOR 999999999 ROWS
```

The optimizer first locates the portion of the index correspondent to last names that start with ″VAN″. Depending on the specified degree of parallelism, that section of the index can be further partitioned.

The selection criteria need not necessarily refer to the leftmost columns of an index. Even if secondary key fields are used, sometimes the optimizer can choose to Parallel Index Scan to resolve the query. Consider the following scenario:

```
CREATE INDEX X1 ON EMPLOYEE (LASTNAME, DEPARTMENT)

SELECT FIRSTNAME, LASTNAME, POSITION
  FROM EMPLOYEE
     WHERE DEPARTMENT = 'JLU'
        OPTIMIZE FOR 999999999 ROWS
```

In this case, we assume that there are no other indices built on the *EMPLOYEE* table. Although we only specified one value for the department number, the fastest way to identify the records that fulfill the selection criteria is to scan the entire index.

It should be mentioned here that if the optimizer estimates that the query is going to return a large percentage of the total number of records, possibly Parallel Table Scan is preferred to Parallel Index Scan.

Parallel Index Scan can be used together with Index-Only Access (see Chapter 6, “Index Only Access” on page 33). The combined effect can result into a very substantial improvement since Index Only Access eliminates the need to access the data space and, therefore, a lot of synchronous I/O operations.

Parallel Index Scan works at its best on systems with multiple processors where several server tasks can truly work in parallel.

Unlike Parallel Table Scan, Parallel Index Scan does not look at the physical distribution of index extents across the disk devices. The index is split into logical key ranges. This means that index pages that contain entries related to separate key ranges may be located on the same disk unit. Parallel tasks that work on different key ranges may, therefore, compete for the same actuator.

Another kind of contention may happen when index entries from different index pages point to records located in different file extents on the same disk unit. This situation may cause the DASD actuator to move back and forth between the disk extents and eventually the DASD service time can become very poor.

This contention can defeat the purpose of parallelism and actually can make performance worse. In order to minimize these occurrences, the optimizer is rather conservative in determining the optimal number of tasks to use. You have to experiment with the CHGQRYA command to find the optimal value for the DEGREE parameter for your query. Sometimes you can use a higher *NBRTASKS value to force the optimizer to be more aggressive.

Even though Parallel Index Scan uses an index for implementing the query, there is no guarantee on the order the records are returned. If you need a specific order, you can use the ORDER BY specification but then the Parallel Index Scan is not used.

Besides the common limitations discussed in Section 2.4.4, “Symmetric Multiprocessing Inhibitors and Limitations” on page 19, Parallel Index Scan is also disallowed for resolving nested loop join queries. These queries are join operations that are implemented by identifying a primary and secondary file and by using an index on the secondary file. If the optimizer uses Hash Join, Parallel Index Scan can still be used.

4.3 Examples

In order to point out the performance advantages provided by Parallel Index Scan, we ran the same test query under several different circumstances.

Environment:

- Hardware configuration: AS/400 model 500 with 2144 processor feature (uni-processor), 384MB main storage, 14GB DASD space with 16 disk units.
- Software configuration: OS/400 V3R7 with option 26 DB2 Symmetric Multiprocessing feature installed.
- Database configuration (see Appendix A, “Database Files Used in Examples” on page 135 for a description of the table layout):
  - Table LINEITEM contained 1 200 000 records.
- Job was run in 110MB pool (with 110MB *BASE pool and 70MB *MACHINE pool).

An index was built over the LINEITEM file:

```
CREATE INDEX X1 ON LINEITEM ( ORDERKEY, LINENUMBER )
```

The following query was run:

```
SELECT ORDERKEY, PRICE
  FROM LINEITEM
    WHERE ( ORDERKEY BETWEEN 1300 AND 1500 OR
            ORDERKEY BETWEEN 2300 AND 2500 OR
            ORDERKEY BETWEEN 3300 AND 3500 OR
            ORDERKEY BETWEEN 4300 AND 4500 OR
            ORDERKEY BETWEEN 5300 AND 5500 OR
            ORDERKEY BETWEEN 6300 AND 6500 OR
            ORDERKEY BETWEEN 7300 AND 7500 OR
            ORDERKEY BETWEEN 8300 AND 8500 OR
            ORDERKEY BETWEEN 9300 AND 9500 )
      AND LINENUMBER = 7
        OPTIMIZE FOR 999999999 ROWS
```

This query returns 128 rows.

The query was run with different values for the DEGREE parameter of the CHGQRYA command. The following results were observed:

Table 3. Parallel Index Scan - Uni-Processor

	DEGREE(*NONE)	**DEGREE(*MAX)**	**DEGREE(*NBRTASKS 99)**
Response time	17 sec	15 sec	17 sec
Number of tasks	-	10	18

To see if the Parallel Index Scan was used to implement your query, run query in debug mode and look in the job log. You should see the message that an index was used to implement the query. If you do not see this message, index was not used and, therefore, the Parallel Index Scan is not applicable:

```
                        Additional Message Information

Message ID . . . . . :   CPI4328      Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/01/96      Time sent  . . . . . :   17:39:53

Message . . . . :   Access path of file X1 was used by query.
Cause . . . . . :   Access path for member X1 of file X1 in library A960202B
  was used to access records from member LINEITEM of file LINEITEM in library
  A960202B for reason code 1. The reason codes and their meanings follow:
    1 - Record selection.
    2 - Ordering/grouping criteria.
    3 - Record selection and ordering/grouping criteria.
    If file LINEITEM in library A960202B is a logical file then member
  LINEITEM of physical file LINEITEM in library A960202B is the actual file
  being accessed.
    Index only access was used for this query:  *NO.
    A value of *YES for index only access processing indicates that all of the
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 5. Query Optimizer Message for Parallel Index Scan

The following message shows how many tasks were actually used to process the query.

```
                        Additional Message Information

Message ID . . . . . :   CPI4330      Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/01/96      Time sent  . . . . . :   17:39:56
                    1
Message . . . . :   18 tasks used for parallel index scan of file LINEITEM.
Cause . . . . . :   18 is the average numbers of tasks used for a index scan
  of member LINEITEM of file LINEITEM in library A960202B.
    If file LINEITEM in library A960202B is a logical file, then member
  LINEITEM of physical file LINEITEM in library A960202B is the actual file
  from which records are being selected.
    A file name of *N for the file indicates a temporary result file is being
  used.                                                                  2
    The query optimizer has calculated that the optimal number of tasks is 18
  which was limited for reason code 1. The definition of reason codes are:
    1 - The *NBRTASKS parameter value was specified for the DEGREE parameter
  of the CHGQRYA CL command.
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 6. Number of Tasks Messages for Parallel Index Scan

In Figure 6, you can see that the optimizer indicates the maximum number of tasks that are used for the query (2). The message shows (at 1) how many tasks were actually scheduled. This number is the **average** number of tasks that were active during query run time. Since the various index partitions can contain different number of entries, this number is almost always less than the one reported in 2 .

If this message is not produced, it means that Parallel Index Scan was not used for your query.

As shown in Table 3 on page 27, the best result was achieved with DEGREE(*MAX) when 10 tasks were used. A further increase in the number of tasks did not improve performance. In fact, the run time got longer. This means that the overhead of scheduling and managing many tasks can outweigh the benefit of parallelism. We can expect that some further, albeit not substantial, improvement can be achieved on a multiprocessor system where server tasks run in parallel on different CPUs.

The implementation of the query used for the previous tests can be roughly divided in two phases: record selection using index and retrieving the selected records from the data space.

Parallel Index Scan improves the first phase of the query execution. This part accounts for the smaller portion of the total response time and, therefore, the effect on the total run time is not very spectacular. The reason behind this behavior is that the index scanning process (performed with sequential I/O operations) is faster than the process of retrieving the records from the database file (lots of random I/O operations).

We can draw the following conclusions:

- Substantial benefits can be obtained from Parallel Index Scan when you run queries over large files where the access paths can reach a considerable size.
- The lower that the ratio is between the number of records returned and the total number of records, the more substantial is the benefit.
- If the physical access to the actual record can be eliminated, you notice a much larger benefit (see also Chapter 6, “Index Only Access” on page 33).

Chapter 5. Parallel Key Positioning

This method is similar to Parallel Index Scan. The major difference between the two is that with Parallel Key Positioning, only some sections of an index are scanned.

In Section 5.1, “How Parallel Key Positioning Works,” we discuss how this method works and what kind of queries can benefit from the method.

5.1 How Parallel Key Positioning Works

Some queries may require a partial index scan to be resolved. In particular, if the optimizer identifies several distinct sections of an index that can be independently scanned by parallel tasks, Parallel Key Positioning is utilized. Refer to *DB2 for OS/400 SQL Programming*, SC41-4611, for a more detailed description of the way this access method works.

Parallel Key Positioning can be effectively used when the number of records returned by the query is a small percentage of the total. If the optimizer estimates that a large portion of the total is returned, probably Parallel Table Scan is used instead (see also Chapter 3, “Parallel Table Scan” on page 21).

Parallel Key Positioning can be used together with Index Only Access (see Chapter 6, “Index Only Access” on page 33). Combining these two methods can be extremely beneficial for performance because no I/O operations are issued against the data space where the records are contained and, in addition, the index is processed in parallel.

Although this access method consists of using an index to find the results, the parallel tasks are going to return their partial results to the coordinating process in an unpredictable sequence. If you require a specific order, you must specify an ORDER BY clause. In this case, Parallel Key Positioning is not used.

Parallel Key Positioning is subject to all of the limitations that apply to other Symmetric Multiprocessing algorithms (see Section 2.4.4, “Symmetric Multiprocessing Inhibitors and Limitations” on page 19).

In addition, remember that join queries implemented using the *nested loop join implementation* as described in *DB2 for OS/400 SQL Programming*, SC41-4611, do not benefit from this access method.

Chapter 6. Index Only Access

Index Only Access can be used if all of the columns referenced by a query are contained in a keyed access path. Under these circumstances, no physical access to the data space is needed and performance will greatly benefit.

In Section 6.1, "How Index Only Access Works," we discuss how this method works. Then in Section 6.2, "Examples" on page 34, we shall see what kind of improvement can be expected.

6.1 How Index Only Access Works

Index Only Access is part of the Symmetric Multiprocessing feature for V3R1 and V3R2, and will be delivered at no charge (free) through an OS/400 PTF in V3R7. One of the reasons for delivering Index Only Access to all customers in V3R7 is that this access method does not intrinsically have anything to do with parallelism.

We see that Index Only Access can be combined with Parallel Index Scan, which is a parallel access method, and this synergy provides additional performance benefits, especially on multiprocessor systems.

For a detailed description of Index Only Access, refer to *DB2 for OS/400 SQL Programming*, SC41-4611. The performance advantage of Index Only Access is obtained through a strong reduction in the number of synchronous I/O operations that are needed to perform the query.

If there is an index that contains all the columns referenced by the query, it is possible to produce the result set without retrieving any data from the data space. The overall query processing requires much less effort in terms of system resources (I/O operations, CPU cycles, and main storage).

Any query for which the optimizer chooses to use an index may benefit from this access method:

- Ordinary record selection through an index.
- GROUP BY queries when an index is used for grouping records.
- Join-type queries when a nested loop joining algorithm is used.

Both permanent and temporary indices can be utilized by Index Only Access.

Index key fields do not have to be contiguous to the leftmost key of the index. For example:

```
CREATE INDEX X3
  ON EMPLOYEE ( DEPARTMENT, LASTNAME, FIRSTNAME )

SELECT FIRSTNAME
  FROM EMPLOYEE
    WHERE DEPARTMENT = 'JLU'
```

To enlarge the opportunities for exploiting Index Only Access, sometimes it is necessary that your indices include columns that typically are not part of an index (see Section 6.2, "Examples" on page 34).

Index Only Access cannot be used with indices that:

- Include null-capable key fields.
- Include variable length character fields.

Once you have an appropriate index in place, the optimizer automatically selects Index Only Access whenever this method is economical for executing a query.

6.2 Examples

In order to give you an idea of what kind of advantages are offered by Index Only Access, we ran a test query in two different circumstances:

- We removed the Index Only Access support from our V3R7 system to make sure that this method is never chosen by the optimizer. We ran the first test case in this environment.
- We installed the Index Only Access support and ran the same query again.

Environment:

- Hardware configuration: AS/400 model 500 with 2144 processor feature (uni-processor), 384MB main storage, 14GB DASD space with 16 disk units.
- Software configuration: OS/400 V3R7 with or without the PTF package needed to enable Index Only Access, Hash Join, and Hash Group By (see Chapter 2, “Introduction to Symmetric Multiprocessing” on page 7).
- Database configuration (see Appendix A, “Database Files Used in Examples” on page 135 for a description of the table layout):
  - Table ORDERS contained 150 000 records.
- Job was run in 110MB pool (with 110MB *BASE pool and 70MB *MACHINE pool).

An index was created over the table:

```
CREATE INDEX X3 ON ORDERS ( CUSTOMER, TOTALPRICE )
```

The column TOTALPRICE was added to the index to realize the benefits of the Index Only Access method.

The following query was run:

```
SELECT CUSTOMER, SUM(TOTALPRICE)
  FROM ORDERS
    WHERE CUSTOMER BETWEEN 1000 AND 1100
      ORDER BY CUSTOMER
        OPTIMIZE FOR 999999999 ROWS
```

The query returns 68 rows.

The ORDER BY clause was added to bias the optimizer towards using the index.

Table 4. Index Only Access

	NOT Installed	**Installed**
Response time	25 sec	15 sec

To determine whether Index Only Access was used for your query, run your query in debug mode. You should find a message in the job log resembling the one reported in Figure 7 on page 35.

```
                        Additional Message Information

Message ID . . . . . :   CPI4328       Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/02/96      Time sent  . . . . . :   15:36:55

Message . . . . :   Access path of file X3 was used by query.
Cause . . . . . :   Access path for member X3 of file X3 in library A960202B
  was used to access records from member ORDERS of file ORDERS in library
  A960202B for reason code 3. The reason codes and their meanings follow:
    1 - Record selection.
    2 - Ordering/grouping criteria.
    3 - Record selection and ordering/grouping criteria.
    If file ORDERS in library A960202B is a logical file then member ORDERS of
  physical file ORDERS in library A960202B is the actual file being accessed.
    Index only access was used for this query:  *YES.  1
    A value of *YES for index only access processing indicates that all of the
  fields used for this query can be found within the access path of file X3. A
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 7. Index Only Access with Record Selection

If you find that *YES is specified for Index Only Access (**1**), you can conclude that the optimizer chose this access method.

When your query performs a join operation, Index Only Access can still be used if the optimizer chooses to implement the join using the *nested loop join* algorithms (refer to *DB2 for OS/400 SQL Programming*, SC41-4611, for more information on the various access methods). In this case, Index Only Access is used to access the secondary file. In the job log, after running your join query in debug mode, you should find a message similar to the one reported in Figure 8 on page 36.

```
                        Additional Message Information

Message ID . . . . . :   CPI4326      Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/02/96     Time sent  . . . . . :   13:49:21

Message . . . . :   File ORDERS processed in join position 2.
Cause . . . . . :   Access path for member X3 of file X3 in library A960202B
  was used to access records in member ORDERS of file ORDERS in library
  A960202B for reason code 4. The reason codes and their meanings follow:
    1 - Perform specified record selection.
    2 - Perform specified ordering/grouping criteria.
    3 - Record selection and ordering/grouping criteria.
    4 - Perform specified join criteria.
    If file ORDERS in library A960202B is a logical file then member ORDERS of
  physical file ORDERS in library A960202B is the actual file in join position
  2.
    A file name of *N for the access path indicates it is a temporary access
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 8 (Part 1 of 2). Index Only Access with Join

```
                        Additional Message Information

Message ID . . . . . :   CPI4326      Severity . . . . . :   00
Message type . . . . . :   Information

  path built over file ORDERS.
    A file name of *N for the file indicates it is a temporary file.
    Index only access was used for this file within the query: *YES.  1
    A value of *YES for index only access processing indicates that all of the
  fields used from this file for this query can be found within the access
  path of file X3. A value of *NO indicates that index only access could not
  be performed for this access path.
    Index only access is generally a performance advantage since all of the
  data can be extracted from the access path and the data space does not have
  to be paged into active memory.
Recovery  . . . :   Generally, to force a file to be processed in join
  position 1, specify an order by field from that file only.
    If ordering is desired, specifying ORDER BY fields over more than one file
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 8 (Part 2 of 2). Index Only Access with Join

Again, the optimizer tells you whether Index Only Access was used or not (**1**).

Chapter 7. Hash Group By

Hash Group By introduces a new way to perform grouping on the AS/400 system. The major innovation about Hash Group By is that no indices, either permanent or temporary, are necessary to execute the query.

In Section 7.1, “How Hash Group By Works,” we discuss how this method works. Then in Section 7.2, “When to Use Hash Group By,” we discuss when it is appropriate to use it. In Section 7.3, “Examples” on page 38 you find some examples demonstrating the advantaged of Hash Group By.

7.1 How Hash Group By Works

Hash Group By is provided by the Symmetric Multiprocessing feature for V3R1 and V3R2 and by a special set of OS/400 PTFs for V3R7. Although Hash Group By was originally part of the Symmetric Multiprocessing feature, this method is not inherently related to parallelism. Hash Group By can benefit from parallel processing, but even on uni-processor systems, it can introduce significant advantages.

Hash Group By is a new algorithm that allows the query engine to perform grouping without the need of an index. The table on which the grouping is performed is scanned sequentially. That is why this access method can strongly benefit from Parallel Table Scan, especially on multiprocessor systems. For a detailed description of how this method works, see *DB2 for OS/400 SQL Programming*, SC41-4611.

The traditional method for performing GROUP BY queries involves using an index over the columns referenced by the GROUP BY clause. For every index entry, the system had to retrieve the corresponding record in the database file. Even if an appropriate permanent index is available, this technique generally results in a large number of random I/O operations. Moreover, if no appropriate permanent index existed, the optimizer schedules the creation of a temporary one and the cost in terms of resource utilization grows even further.

With Hash Group By, the table is scanned sequentially and a ″hash table″ is built as a result of the scan. This eliminates most of the random I/O operations. The cost of building a hash table is much lower than the cost of building a temporary index. As we see in Section 7.3, “Examples” on page 38, often Hash Group By offers good performance advantages over the index-based Group By even if a permanent index exists.

When parallelism is enabled, this algorithm can provide even further benefits. The idea behind it is much the same as in the Parallel Table Scan method (see Chapter 3, “Parallel Table Scan” on page 21).

7.2 When to Use Hash Group By

The Hash Group By can be efficiently used by the OS/400 query optimizer if a query:

- Requires GROUP BY processing.

- Processes all of the records or a large percentage of the records of a large database table.
- The ratio between the number of groups and the total number of records is small.

Under different circumstances, such as when the grouping occurs over a limited number of records or when the grouping criteria produce a high number of groups compared to the number of records, the usage of a permanent index might produce better results.

Hash Group By applies to all those queries that perform some summarization over large files. As we demonstrate, this method can prove to be faster than the index-based Group By even if a suitable permanent index exists. The advantages are by far more obvious if you take into account the time it takes to build a temporary index. This is why Hash Group By can be of extreme value for customers that cannot exactly predict the structure of their queries and, therefore, cannot create all of the possible permanent indices to suit their summarizations. In particular, Data Warehousing environments may greatly benefit from Hash Group By and from Hash Join (see also Chapter 8, “Hash Join” on page 43).

One simple example of a query that can benefit from the Hash GROUP BY is found in Section 7.3, “Examples.”

The best results are obtained on systems with multiple processors where several server tasks can truly work in parallel.

The Hash Group By method requires main storage to build the hash table and when parallel support is used, still more memory may be needed. The memory requirement depends on how many tasks are used and on how many groups exist in the database file. If there are many groups, hash tables can become rather large.

When Hash Group By is used, the table is scanned sequentially and, therefore, there is no guarantee about the order in which the groups are returned to the application. This does not mean, however, that Hash GROUP BY cannot be used in queries with the ORDER BY specification. When ORDER BY is used, the optimizer estimates the number of groups that are going to be returned by the query. If that number is large, an index is used (a temporary index is built if a permanent one is not available). If there are not many groups, the optimizer can choose to create a temporary file to hold the results and use a sort routine before returning them to the application.

For more details and examples on ordering result sets produced by queries that use Hash Group By, see Section 7.3.1.1, “Order By and Hash Group By” on page 40.

7.3 Examples

We ran a test query under different circumstances:

Environment:

- Hardware configuration: AS/400 model 500 with 2144 processor feature (uni-processor), 384MB main storage, 14GB DASD space with 16 disk units.

- Software configuration: OS/400 V3R7 with Option 26 DB2 Symmetric Multiprocessing feature installed.
- Database configuration (see Appendix A, “Database Files Used in Examples” on page 135 for a description of the table layout):
  - The table LINEITEM contained 1 200 000 records.
- The job was run in 110MB pool (with 110MB *BASE pool and 70MB *MACHINE pool).

The query of Figure 9 was used for our tests.

```
SELECT SUM(PRICE), SHIPMODE
  FROM LINEITEM
    GROUP BY SHIPMODE
```

Figure 9. A Group By Query

Seven rows were returned.

The query was run in the following scenarios:

- Hash GROUP BY implementation was allowed by specifying ALWCPYDTA(*OPTIMIZE) for the query. Parallel processing was suppressed by CHGQRYA DEGREE(*NONE).
- Same as the preceding but parallel processing was allowed by CHGQRYA DEGREE(*NBRTASKS 20).
- Index implementation was forced by specifying ALWCPYDTA(*YES). There were no permanent indexes over the file that could be used for this query. This was done to show what impact an index build time can introduce to query execution.
- Index implementation was forced by specifying ALWCPYDTA(*YES). This time, the following index was created over the table:

  ```
  CREATE INDEX XS ON LINEITEM ( SHIPMODE )
  ```

- The last test was repeated with Expert Cache activated for the job’s storage pool (PAGING(*CALC) was specified for the pool). This was done to show what impact Expert Cache can have on queries with lots of I/O.

The following results were observed:

Table 5. Hash GROUP BY

	Hash GROUP BY DEGREE (*NONE)	**Hash GROUP BY DEGREE *NBRTASKS 20** [1]	**Index GROUP BY No index**	**Index GROUP BY Index available**	**Index GROUP BY Index available Expert Cache**
Response time	100 sec	85 sec	1870 sec (>31 min)	1300 sec (>21 min)	700 sec (<12 min)
Average CPU utilization	53%	80%	13% (97%) [2]	13%	19%

[1] DEGREE(*NBRTASKS 20) 10 tasks were scheduled to process the query.

[2] 97% while index was built and 13% during grouping processing.

When you run your queries in debug mode, the optimizer does not give you any direct notice that Hash Group By was used. However, if in the job log you find a message telling you that arrival sequence access was used for the query (message CPI4329), it means that the optimizer chose Hash Group By implementation. In Figure 10, you can look at the message that the optimizer produced when we ran the test query in debug mode.

```
                        Additional Message Information

Message ID . . . . . :   CPI4329      Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/05/96     Time sent  . . . . . :   13:54:26

Message . . . . :   Arrival sequence access was used for file LINEITEM.
Cause . . . . . :   Arrival sequence access was used to select records from
  member LINEITEM of file LINEITEM in library A960202B.
    If file LINEITEM in library A960202B is a logical file then member
  LINEITEM of physical file LINEITEM in library A960202B is the actual file
  from which records are being selected.
    A file name of *N for the file indicates it is a temporary file.
Recovery  . . . :   The use of an access path may improve the performance of
  the query if record selection is specified.
    If an access path does not exist, you may want to create one whose
  left-most key fields match fields in the record selection.  Matching more
  key fields in the access path with fields in the record selection will
                                                                      More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 10. Query Optimizer Message for Hash GROUP BY

The results of our test show that Hash Group By is an effective method that can significantly reduce query run time (seven times better than an index implementation even in the best-case scenario). Even on a uni-processor system, the query run time was improved by 15% when the system was allowed to schedule 10 tasks to process the database file in parallel. On a multiprocessor system, still further improvements can be expected.

7.3.1.1 Order By and Hash Group By

If you need to order your result set, you **must** specify an *ORDER BY* clause or an equivalent parameter. We did not specify any ordering criteria for the query of Figure 9 on page 39 and, therefore, there is no guarantee that the result set respects any particular sequence.

When you specify ordering criteria, the optimizer may choose to use Hash Group By and then sort the result set. This access method can still buy you great performance advantages over the index access method, especially when a temporary index is needed.

Let's consider the query in Figure 11 on page 41.

```
SELECT SUM(O_TOTALPRICE), O_ORDERDATE
  FROM A960202C/ORDERS
     WHERE O_ORDERDATE BETWEEN '1994-01-01' AND '1994-12-31'
       ORDER BY 2
         OPTIMIZE FOR 999999 ROWS
```

Figure 11. Combining ORDER BY and Hash Group By

We ran the query in a way that disabled Hash Group By from being used (we used Interactive SQL and set *ALWCPYDTA(*YES)*). This is the job log we obtained running the query in debug mode:

```
All access paths were considered for file ORDERS.
Access path built for file ORDERS.                1
Temporary result file built for query.            2
ODP created.
Blocking used for query.
Connection to relational database SYSNMLKV ended.
```

On an AS/400 model F95 with a storage pool of 128MB, we obtained that the temporary index creation (**1**) lasted 3 minutes and 5 seconds, whereas the creation of the temporary result file (**2**), which was necessary to allow the results to be sorted, required 1 minute and 38 seconds. We enabled Hash Group By by setting *ALWCPYDTA(*OPTIMIZE)* and Symmetric Multiprocessing by running *CHGQRYA DEGREE(*MAX)*. This was the job log:

```
All access paths were considered for file ORDERS.
Arrival sequence access was used for file ORDERS.
Temporary result file built for query.    3
ODP created.
Blocking used for query.
7 tasks used for parallel table scan of ORDERS.
Connection to relational database SYSNMLKV ended.
SQL cursors closed.
```

The temporary index creation is not needed anymore and the temporary result file (**3**) was created in 19 seconds by seven parallel tasks.

The net result was that the query run time was reduced from about 5 minutes to about 20 seconds, more than an order of magnitude.

Chapter 8. Hash Join

This method provides an alternative implementation method for queries involving join criteria that otherwise need an index (permanent or temporary) to complete.

In Section 8.1, “How Hash Join Works,” we discuss how this method works. Then in Section 8.2, “When to Use Hash Join,” we shall see what needs to be considered when using it and Section 8.3, “Examples” on page 44 gives you some idea about what kind of improvement can be expected.

8.1 How Hash Join Works

Hash Join has been packaged with the Symmetric Multiprocessing feature for V3R1 and V3R2 and will be available through an OS/400 PTF for V3R7. Although originally this algorithm was part of Symmetric Multiprocessing, Hash Join is not inherently related to parallel processing and can provide great performance benefits both on uni-processor and on multi-processor systems.

Hash Join allows the OS/400 query engine to perform join operations without the necessity of an index. Prior to the availability of Hash Join, a join query could be performed only by means of the *nested loop join* algorithm, which implied the necessity of an index on the join columns for the designated secondary file. This algorithm sometime produces a large number of synchronous I/O operations and, in addition, it causes a temporary index to be built if no suitable permanent index exists.

When the optimizer chooses to use the Hash Join access method, a hash table is built for each of the files. This process takes a sequential scan of the two files and can take advantage of Parallel Table Scan.

The hash tables are created according to the join criteria. The two hash tables are then merged to produce the final result set with no further need for random I/O access to the data spaces. A more detailed description of the way Hash Join works can be found in *DB2 for OS/400 SQL Programming*, SC41-4611.

8.2 When to Use Hash Join

Queries performing join operations are common and most of these queries can take advantage of Hash Join. As we pointed out for Hash Group By (see Section 7.2, “When to Use Hash Group By” on page 37), this technique expresses its greatest benefits in environments where the structure of the queries cannot be predicted, such as in a Data Warehousing implementation.

You can find an example of a query that can benefit from the Hash Join method in Section 8.3, “Examples” on page 44.

Hash Join, as with any other method discussed in this part of the book, has to comply with the limitations we discussed in Section 2.4.4, “Symmetric Multiprocessing Inhibitors and Limitations” on page 19.

In addition, you must remember that Hash Join support cannot be used for the following types of queries:

- Queries that specify join operators other than *equal to*, also known as *non-equijoin* queries.
- Left, outer, or exception join queries.
- Queries that are run against a join logical file created with DDS.

8.3 Examples

Here are the results of a test case we ran to demonstrate the advantages of Hash Join.

Environment:

- Hardware configuration: AS/400 model 500 with 2144 processor feature (uni-processor), 384MB main storage, 14GB DASD space with 16 disk units.
- Software configuration: OS/400 V3R7 with option 26 DB2 Symmetric Multiprocessing feature installed.
- Database configuration (see Appendix A, “Database Files Used in Examples” on page 135 for a description of the table layout):
  - Table LINEITEM contained 600 000 records.
  - Table ORDERS contained 150 000 records.
- Job was run in 110MB pool (with 110MB *BASE pool and 70MB *MACHINE pool). Expert Cache was active for the pool.

The following query was run:

```
SELECT O.ORDERSTATUS, SUM ( L.PRICE )
  FROM ORDERS O, LINEITEM L
    WHERE O.ORDERKEY = L.ORDERKEY
      GROUP BY O.ORDERSTATUS
        OPTIMIZE FOR 999999999 ROWS
```

Three rows were returned.

The query was run in the following scenarios:

- Hash Join implementation was allowed by specifying ALWCPYDTA(*OPTIMIZE) for the query. Parallel processing was suppressed by CHGQRYA DEGREE(*NONE).
- Same as the preceding but parallel processing was allowed by CHGQRYA DEGREE(*NBRTASKS 20).
- Index implementation was forced by specifying ALWCPYDTA(*YES). For this test, the following index was created to perform join:

  ```
  CREATE INDEX L_ORDER ON LINEITEM (ORDERKEY)
  ```

  and another index was created to perform grouping:

  ```
  CREATE INDEX O_ORDERSTATUS ON ORDERS (ORDERSTATUS)
  ```

The following results were observed:

Table 6. Hash Join			
	Hash Join DEGREE(*NONE) 1	**Hash Join DEGREE(*NBRTASKS 20) 2**	**Index Join**
Response time	120 sec	100 sec	165 sec

1 Hash Join was used to build a temporary file. Then a Hash GROUP BY was used to process specified grouping.

2 Hash Join was used to build a temporary file. Eighteen tasks were scheduled to build a hash table for the LINEITEM file and seven tasks were used to build a hash table for the ORDERS file. Then a Hash GROUP BY was used to process specified grouping. Six tasks were scheduled to build a hash table.

To discover what kind of join implementation was used for your query, run your query in a debug mode. If Hash Join was used, the following message is sent to the job log:

```
                        Additional Message Information

Message ID . . . . . :   CPI4333       Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/05/96      Time sent  . . . . . :   17:59:02

Message . . . . :   Hashing algorithm used to process join.
Cause . . . . . :   The hash join method is typically used for longer running
  join queries. The original query will be subdivided into hash join steps.
    Each hash join step will be optimized and processed separately. Debug
  messages which explain the implementation of each hash join step follow this
  message in the joblog.
    The list below shows the file names of the files used in this query. The
  format of each entry in this list is the number of the hash join step, the
  filename as specified in the query, the member name as specified in the
  query, the filename actually used in the hash join step, and the member name
  actually used in the hash join step.
      1  A960202B/ORDERS       ORDERS      A960202B/ORDERS       ORDERS,
                                                                      More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 12. Query Optimizer Message for Hash Join

As the results point out, the Hash Join method complemented by Hash Group By was 25% faster than the traditional implementation, even when parallel processing was not used.

When parallel support was used, the total improvement amounted to about 40% of the run time. Keep in mind that these results were obtained on a uni-processor system where parallelism cannot be exploited to its full extent. Further improvements can be expected on a multiprocessor system (see also Section 3.3, “Examples” on page 22).

8.4 Hash Join and Subquery Processing

Even though many subqueries can be resolved in a join statement, Symmetric Multiprocessing algorithms (Hash Join and Parallel Table Scan) are not applicable to subqueries at the time this publication was written.

In many cases, though, the performance advantages of hash join versus building a temporary index are so compelling that you may want to turn your subqueries into join statements manually and then execute them.

Consider, for example, the following query:

```
SELECT O_ORDERKEY FROM A960202C/ORDERS
 WHERE O_ORDERKEY IN (SELECT L_ORDERKEY FROM A960202C/LINEITEM
  WHERE L_EXTENDEDPRICE > 50000)
   OPTIMIZE FOR 99999999 ROWS
```

Figure 13. Subquery Implemented as a Join

The previous SQL statement is logically equivalent to the following request: ″Show me all of the order numbers for orders having at least a single line item which is worth more than 50 000 dollars″.

We executed this query on our test database and always obtained an index join implementation. A temporary index had to be built because a permanent one with the appropriate characteristics was not available.

Notice that the query of Figure 13 can be manually turned into a join as you can see in Figure 14.

```
SELECT DISTINCT O_ORDERKEY
 FROM A960202C/ORDERS, L_ORDERKEY
  WHERE L_EXTENDEDPRICE > 50000 AND
   O_ORDERKEY=L_ORDERKEY
    OPTIMIZE FOR 99999999 ROWS
```

Figure 14. A Join Query Equivalent to the Subquery

We ran the query of Figure 14 and obtained a much faster implementation through Hash Join and Parallel Table Scan. Due to the *DISTINCT* clause, the optimizer chooses to create, process, and, finally, sort a temporary result file (**1**). The processing of the temporary result file also takes place through multiple parallel tasks as you can see in Figure 15 (**2**).

```
     Hash join method being used
     Arrival sequence access was used for file LINEITEM.
     Arrival sequence access was used for file ORDERS.
     Arrival sequence access was used for file *QUERY0001.  1
     Temporary result file built for query.
     ODP created.
     Blocking used for query.
     6 tasks used for parallel table scan of *QUERY0001.  2
     8 tasks used for parallel table scan of LINEITEM.
     8 tasks used for parallel table scan of ORDERS.
```

Figure 15. Messages Issued During the Execution of the Modified Subquery

It is important to point out that the benefit of turning a subquery into a join decreases with the size of the temporary result file because the sorting process does not take advantage of parallelism. In other words, if the subquery selects a lot of records, the improvement may not be dramatic.

Chapter 9. Setting Up and Implementing DB2 Multisystem

This chapter covers the fundamental aspects of planning for and implementing solutions based on DB2 Multisystem. Topics dealt with in this chapter include:

- Communications set up
- Relational Database Directory configuration
- Customizing distribution criteria

9.1 Prerequisites to Activating DB2 Multisystem

This section of the book identifies some of the basic considerations when planning to install the DB2 Multisystem for OS/400 product.

Aside from identifying the hardware and the implementation of the DB2 Multisystem requirements, there are other issues that may require attention:

- Files to be distributed across multiple systems must be externally described.
- The Data File Utility (DFU) is not supported on distributed files. This is because the utility uses relative record number processing. Distributed database files are physically distributed across multiple systems. Each system has its own relative record location for its share of the file. Thus, DFU′s file access method is defeated here.
- Variable length records are not supported as distributed files. However, you can continue to use them in the multiple system environment as long as no programs request to open one of these files with the variable record length processing option.
- Files having multiple members cannot be supported as distributed files.

9.1.1 Communications Considerations

When you plan to install DB2 Multisystem, you need to consider what communications is best for your installation. This decision is important to the overall success of the project because it is relevant to the users′ response time.

You want to consider the following issues:

- Are you going to be sending or receiving large amounts of data across nodes with your applications or queries?
- Are there many users accessing the distributed data on a node other than their own?
- How many jobs can be simultaneously active?
- Will there be any changes in the company′s future that may require a different approach to distributed data?

There are many different types of communications media to choose from. For instance, you may choose to use OptiConnect if your nodes are contained in a campus type of environment. OptiConnect uses optical fibers, which support extremely high data transfer rate. However, OptiConnect technology allows a maximum length for communications links of 400 meters (approximately 1300 feet). If you are going to use DB2 Multisystem because you are expanding your system horizontally, OptiConnect is definitely the best option.

Local Area Network is also a valid option if OptiConnect is not feasible or economical. Wide Area Network communications are also a possibility for customers who want to integrate multiple locations into one single database. If you are going to pursue this option, you should carefully plan your distribution criteria in order to minimize the amount of data flowing across the network. However, your performance expectations should take into account that some common database operations such as opening a file or reading a record by key need to be broadcast to all the systems in the network when they are directed to a distributed file.

DB2 Multisystem uses DDM communications support. When configuring communications, specify *Secure Location *YES* in your controller descriptions or in your configuration lists. In this way, you enable the target DDM jobs to run with the same user profile as the source jobs. See also Section 11.5, “Security Considerations” on page 85 for more information on this subject.

9.1.1.1 Relational Database Directory

Prior to using DB2 Multisystem, you also need to configure the Relational Database Directory on the various participating systems.

See *Distributed Database Programming*, SC41-3702, for information on configuring the Relational Database Directory.

To check how the Relational Database Directory has been set up, use the Work with Relational Database Directory Entries (WRKRDBDIRE) command. In Figure 16, you can look at the display that is shown by issuing this command.

```
                Work with Relational Database Directory Entries

Position to  . . . . . .

Type options, press Enter.
  1=Add   2=Change   4=Remove   5=Display details   6=Print details

        Relational          Remote
Option  Database            Location  Text

  __    PRODUCTION          F95       Production AS/400
  __    CORPORATE           CC3       Remote AS/400 CC
  __    WAREHOUSE           *LOCAL    Warehouse AS/400









                                                                  Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 16. Example of Relational Database Directory Entries

Make sure that there is at least one system defined as the local (*LOCAL) system.

If there is no system defined as the local system, you need to designate the local system by specifying *LOCAL in the *Remote Location* parameter of the relational database directory entry.

Note: Each system in your node group must have the node names consistent across all nodes.

Add the local system reference to the remote location if necessary on the Work with Relational Database Directory Entries display or use the Add Relational Database Directory Entry (ADDRDBDIRE) command.

9.1.2 Security

Authorizations to distributed files are managed the same way as for any other AS/400 object. File authorities can be changed on any of the participating nodes and the change is automatically propagated to all of the nodes. This mechanism implies that you should create the same user profiles on every node and assign them the same authorities. The same consideration applies to authorization lists.

Also, verify that the necessary libraries are in place at every node. You need to set the ownership of physical files and public authorities consistently on all systems within your node group.

9.1.3 Partition Map

The partition map is the virtual director that determines the way your records are distributed to the nodes in the node group.

The partition map contains multiple partition numbers from 0 to 1023. Each partition is associated to a node number that relates directly to a system in your node group.

In Figure 17, you can look at an example that shows the *default* partition map.

Sample Node Group	**Sample Partitioning File**	
Node Group A	*Partition Number*	*Node*
SystemA	0	1(SystemA)
SystemB	1	2(SystemB)
SystemC	2	3(SystemC)
	3	1(SystemA)
	4	2(SystemB)
	5	3(SystemC)
	6	1(SystemA)
	.	
	.	
	1023	

Figure 17. Basis For Node Group and Elements of Default Partitioning Map

Figure 17 refers to a scenario where the node group (GROUPA) contains three systems. By default, the partitioning attributes assigned an equal share of partitions to each node.

Important!

Even though each node is assigned the same number of partitions, this does not necessarily imply that each node receives the same number of records of a distributed file. The number of records assigned to each node depends on the way the values of the partitioning key are hashed to partition numbers. See Section 9.2.1, “Creating A Node Group” on page 55 and also see Chapter 10, “How To Distribute” on page 61.

9.1.4 Partitioning Key

When you create a distributed file, you need to indicate a field or group of fields that act as the partitioning key of the file. When the system needs to determine on which node a record belongs, it processes the contents of the partitioning key by means of a *hash function*. The result returned by the hash function is an integer number in the range that goes from 0 to 1023 and that number is the partition number where the record belongs.

For example, you decide to distribute your data using the zip code field as the partitioning key for your file.

Let′s take the case of default partitioning described in the previous section. If it happens that 50% of your business comes from a specific area, one of the three nodes contains at least fifty percent of the data. The other two nodes contain the remaining twenty-five percent in proportions that are not predictable unless you know the contents of the file. This is why it is important to analyze how your data is distributed.

Remember, the hash algorithm decides what partition your data resides in. It can happen that over ninety percent of your data is located on one particular system. You need to analyze how the data is hashed and also what partitions have to be assigned to each system.

The distribution strategy largely depends on the reason why you are moving to DB2 Multisystem.

If your business requires horizontal growth and you are connecting your systems locally, you probably want your data to be distributed evenly across the nodes in the node group. You probably choose a partitioning key with a wide range of values, such as a customer or an order number, a social security number, and so on.

On the other hand, if you are using DB2 Multisystem to establish a corporate-wide view over your company′s data that happens to be geographically distributed, then transactions on the individual nodes should avoid referencing remote records as much as possible. This way, you minimize the impact of remote access on response time. The partitioning key, in this case, should identify the location where each record belongs and should not have a wide range of possible values.

If you are distributing existing data, you can verify how your data is distributed *before* you actually distribute your file by running an SQL query that uses the new **Hash** scalar function. This function is also supported by the *Open Query File* command. Using the example of the zip code field, we can determine what the partitioning is by running the following query:

```
SELECT ZIPCD, COUNT(*), HASH(ZIPCD)
 FROM CUSTOMER
  GROUP BY ZIPCD
   ORDER BY ZIPCD
```

In Figure 18, you can look at the results of the query.

```
                Display Data

Position to line  . . . . .
....+....1....+....2....+....3....+....4
 ZIPCD     COUNT ( * )   HASH ( ZIPCD )
 10027           1,499              162
 10582             999              573
 31105               1              766
 34827             999            1,014
 50002               1              620
 50342             499              700
 55901             699              410
 58234           3,500              644
 80551           1,299              570
 90113             999              492
 90292              69              522
 91001             300              747
 91002               1              734
 91003           2,000              731
 91005               1              727
 91032           1,012              264
 91035               1              297
 91036               1              282
********  End of data  ********

F3=Exit      F12=Cancel      F19=Left
```

Figure 18. Example of SQL Display Using Hash Function

The system uses the partitioning map to direct the records to the system designated by the partition number. The results of the SQL query using the HASH scalar function allows you to determine how you can customize your partitioning map.

9.1.4.1 Selecting a Good Partitioning Key

This is probably the single most important step of designing your DB2 Multisystem solution.

- Select partitioning key fields that do not get updated often. Updating the partitioning key should be avoided. DB2 Multisystem allows you to update the partitioning key provided that the update operation does not cause the record to belong to a different node. If the update operation requires that the record moves to another node, the operation will fail.
- Date, time, timestamp, or floating-point numeric fields cannot be used in the partitioning key.

- On files that are often joined, consider using the join field as the partitioning key to increase performance.
- If you want to achieve even distribution across the systems in your node group, select a partitioning key that has many different values. This option is definitely the best for large data warehouses if they are distributed across multiple nodes of equal size.
- If your systems are geographically distributed, you probably want to avoid transferring data across the network as much as you can. For this reason, on each system you should concentrate the records that belong to that particular location, such as information about local customers or information related to the local inventory. In this way, you minimize the impact of communications media on response time.

  You may not always find it simple to select a group of fields that identifies the geographical location where records should belong. In a file that contains your customers′ data, maybe the zip code is a suitable field. In other cases, it might be close to impossible to find good candidates and you may even want to create new fields to identify where the records belong.

We talk more extensively about this subject in Chapter 10, “How To Distribute” on page 61.

9.1.5 Determine What Files To Distribute

There are some considerations that you may want to keep in mind when it is time to make a decision about which files should become distributed files. The following list includes some information that may help you in making decisions:

- Distributed files can only have one member. Multi-member physical files cannot be distributed with DB2 Multisystem.
- Distributed files must be externally described. Program described files are not supported.
- Data File Utility (DFU) does not work well against distributed files. See Section 9.1, “Prerequisites to Activating DB2 Multisystem” on page 49.
- If you want to create a unique key access path for a distributed file, the partitioning key must be a subset of the unique key. You can always change the structure of your unique keys by *adding* the partitioning key to them. Remember that if you modify existing access paths, you may jeopardize existing applications. If your files already have a unique key that does not contain the partitioning key, you should operate as follows before distributing:

  1. Modify the existing unique key to become a non-unique key, including the same fields as before. This ensures that all applications keep working.
  2. Create a new unique-keyed logical file or index containing the fields of the previous access path plus the partitioning key.
- Only one *based on* file is allowed for distributed files. Distributed join logical files are not allowed, although join SQL views and joining distributed files with SQL commands is supported. Joining local files with distributed files is also supported. However, join queries that involve distributed files may sometimes perform differently from join queries between local files. See Section 12.5, “Distributed Join” on page 110 for more information on join operations that involve distribute files.

- Indexes and logical files created on distributed files are automatically created on all nodes. Each node contains the portion of the access path that is relevant to the node itself. This design choice guarantees good performance of access path maintenance because the system has to update only the local portion of an access path when data is modified in a distributed file. On the other hand, when you request to read a record by key, each node receives the request unless the partitioning key is part of the key you are using.

9.2 Activating DB2 Multisystem

If you need to distribute some existing database files, you activate distribution using the CHGPF command, as we discuss in Section 11.8, “Redistribution Considerations” on page 99. Make sure you allocate an appropriate window of time to allow the systems to distribute your records. Distributing a file potentially means moving every record. All logical files based on distributed files also become distributed. In other words, the process of activating DB2 Multisystem for an existing large database may be lengthy.

Of course, the time it takes to scatter the records around strongly depends on the number of records and on the speed of your communications media. Some informal tests made by the Rochester laboratory show that in a network of three large AS/400 systems (Model 530, 4-ways, running Version 3 Release 7 of OS/400) connected in a token-ring LAN (16MB), files of about one million records spread in about 10 minutes.

If you need to consolidate multiple separate local files into one larger distributed file, this can be done by copying the records to each local file into the distributed file. For example, you may need to consolidate three separate *STOCK* files that keep the inventory at three different locations into a single distributed file. You need to create a separate distributed file with a different name, let′s say *STOCKAUX*. You then copy the contents of each STOCK file into STOCKAUX. At the end, you delete the separate STOCK file and rename STOCKAUX into STOCK. Applications using STOCK keep working as they used to.

9.2.1 Creating A Node Group

Prior to creating the distributed files, you must create a Node Group, which is an AS/400 object of type *NODGRP that contains the reference to the systems across which the records must be distributed and the partition map. Node groups can be saved and restored with the SAVOBJ and RSTOBJ commands as we see in Section 11.4.1, “How to Save Distributed Files” on page 80 and Section 11.4.2, “How to Restore Distributed Files” on page 81.

The following example shows how to create a node group using the default partitioning:

```
                        Create Node Group (CRTNODGRP)

Type choices, press Enter.

Node group . . . . . .   MYNODGRP      Name
  Library  . . . . . .     MYLIB       Name, *CURLIB
Relational database  . . . . . . > WAREHOUSE1  1
                                  > WAREHOUSE2
              + for more values > WAREHOUSE3
Partitioning file  . . . . . .   *NONE        Name, *NONE
  Library  . . . . . .     *LIBL       Name, *LIBL, *CURLIB
Partitioning member  . . . . . .   *FIRST       Name, *FIRST
Text 'description' . . . . . .   *BLANK


                                                                  Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 19. Example of Creation of a Node Group

Remember that the node group contains a reference to the relational database names with which the systems are known in the Relational Database Directory of your network. In Figure 19 (**1**), you can see that our three systems have been assigned the relational database names *WAREHOUSE1*, *WAREHOUSE2*, and *WAREHOUSE3*. See also Figure 16 on page 50.

9.2.2 Displaying A Node Group

If you want to look at the contents of a node group use the Display Node Group (DSPNODGRP) command. Issuing this command brings up the display in Figure 20 on page 57 that shows the association between the systems in the group and their node number.

```
                          Display Node Group

Node Group:   GROUPA          Library:   LIBA
Relational                  Node
Database                   Number
WAREHOUSE1                      1
WAREHOUSE2                      2
WAREHOUSE3                      3

                                                                       Bottom
F3=Exit   F11=Partitioning Data   F12=Cancel   F17=Top   F18=Bottom
(C) COPYRIGHT IBM CORP. 1980, 1996.
```

Figure 20. Example of Display Node Group

Press F11 to display the partition number assigned to each node.

```
                          Display Node Group

Node Group:   GROUPA          Library:   LIBA
Partition      Node
  Number      Number
       0           1
       1           2
       2           3
       3           1
       4           2
       5           3
       6           1
       7           2
       8           3
       9           1
      10           2
      11           3
      12           1
      13           2
      14           3

                                                                       Bottom
F3=Exit   F11=Partitioning Data   F12=Cancel   F17=Top   F18=Bottom
(C) COPYRIGHT IBM CORP. 1980, 1996.
```

Figure 21. Example of Display Node Group Partitioning Data

Notice that the node numbers are assigned to the partitions in the order that you specified them on the CRTNODGRP command.

Although the node group object contains the partition number and node number, this information is also contained in the actual distributed physical file.

9.2.3 Changing A Node Group

If you want to change the partition map of a node group, you can do it by using the Change Node Group Attribute (CHGNODGRPA) command. You can either assign a specific partition number to a selected partition or force the value of the partitioning key to a specific partition. For example, if we want to change assign partition number 10 to the node number 1, we use the following command:

```
CHGNODGRPA NODGRP(LIBA/GROUPA) PTNNBR(10)
   NODNBR(1)
```

```
                          Display Node Group

Node Group:   GROUPA          Library:   LIBA
Partition      Node
  Number      Number
       0           1
       1           2
       2           3
       3           1
       4           2
       5           3
       6           1
       7           2
       8           3
       9           1
      10           1          Notice the Change of Node Number
      11           3
      12           1
      13           2
      14           3

                                                                    Bottom
F3=Exit   F11=Partitioning Data   F12=Cancel   F17=Top   F18=Bottom
(C) COPYRIGHT IBM CORP. 1980, 1996.
```

Figure 22. Example of Change Node Group Attributes Partition Number

If you want all of the zip codes in the range of 91001 through 91035 to be distributed to node 1, we use the following commands:

```
CHGNODGRPA NODGRP(LIBA/GROUPA) CMPDTA(91001) NODNBR(1)
CHGNODGRPA NODGRP(LIBA/GROUPA) CMPDTA(91002) NODNBR(1)
CHGNODGRPA NODGRP(LIBA/GROUPA) CMPDTA(91004) NODNBR(1)
CHGNODGRPA NODGRP(LIBA/GROUPA) CMPDTA(91005) NODNBR(1)
CHGNODGRPA NODGRP(LIBA/GROUPA) CMPDTA(91006) NODNBR(1)
                      ......
CHGNODGRPA NODGRP(LIBA/GROUPA) CMPDTA(91035) NODNBR(1)
```

All of the files created or distributed using the GROUPA node group and have a partitioning key that stores records with the 91001 through 91035 values are directed to the specified node number 1.

It is important you understand that using the *CMPDTA* parameter to direct a value to a given node equates to assigning the partition number that results from hashing the value to the node. The system takes the value that is specified in the *CMPDTA* parameter and, using the hash function, determines the corresponding partition: the partition map is modified accordingly. This also implies that any other values of partitioning keys that hash to this partition number are directed to the selected node. Consider, for instance, the case of a

distributed file whose partitioning key is the *STATE* field. In the United States, the various states are designated by two characters. The hashing function happens to give the same number for ″MT″ (Montana) and ″TX″ (Texas). Both of these values hash to the partition number 53.

Based on the previous discussion, you can conclude that if you issue the following command:

```
CHGNODGRPA NODGRP(LIBA/MYNODGRP) CMPDTA(MT) NODNBR(1)
```

you ultimately update the node number for the partition number 53. Once you apply this modification to the file, both the records from Montana and those from Texas end up onto the same system.

In fact, records with ″MT″ and with ″TX″ cannot be separated if you choose the STATE field as partitioning key. You would need to choose a different partitioning key if your requirement is to separate these records.

9.3 Monitoring Your Distributed Files

You want to periodically monitor the way your distributed files are actually distributed across the nodes. This helps you to maintain the correct distribution of your files and to avoid performance exposures due to poor distribution.

Monitor the distribution records by creating a simple SQL queries using the scalar functions NODENUMBER or NODENAME that are also available for the OPNQRYF command.

9.4 Control Language Commands and Distributed Files

The distributed file has the same object type of *FILE as a local file. For this reason, almost every CL command that applies to normal files also applies to distributed files. There are some limitations and restrictions that you must be aware of. For more information on CL commands for distributed files, see *DB2 Multisystem for OS/400*, SC41-3705.

9.5 CCSIDs and SRTSEQ Tables

Coded character set identifiers (CCSIDs) and sort sequence tables are resolved from the originating system.

Chapter 10. How To Distribute

Before distributing database files across two or more systems in a network, you need to determine how you want your files to be distributed. Ultimately, you have to determine which field (or fields) of your files represent a good partitioning key. This choice has a long-lasting influence on all aspects of application structure and operations.

In some cases, you may want to change the way your files are distributed. DB2 Multisystem for OS/400 provides a simple interface to perform the redistribution task. Although this process is simple to use, for a large file, it can involve moving around a large number of records, can take a long time, and can require a lot of system resources. A well-chosen distribution strategy helps avoid this painful restructuring.

In this chapter, we intend to give you the necessary information for choosing the right partitioning keys for your files.

10.1 How To Choose A Partitioning Key

There are several different strategies to distribute data with DB2 Multisystem. Each strategy has a distinct purpose and, therefore, can lead to different decisions concerning the choice of a partitioning key and the allocation of partitions to nodes in a node group.

The three most important strategic lines along which you may want to move are:

- Optimizing the workload distribution among multiple systems. This is probably the most common strategy of data warehousing customers.
- Logical separation of data between systems. This is the line followed by customers that need to integrate separate locations into a single database.
- Visibility node: this technique allows you to concentrate all the data on some nodes called the data nodes and leave the remaining nodes, called visibility nodes, with just the description of the distributed files.

Any practical approach requires some combination of the preceding strategies, but to see the basic differences between them, it is more convenient to discuss them separately.

10.1.1 Distribution of Workload Among Systems

This strategy comes into play if you have several AS/400 systems (of similar or different capacity and performance) and want to have them working in concert with the same database and same applications.

Workload distribution can have many practical uses:

- You may want to evenly distribute the workload between similar systems.
- You may want to make the distribution proportional to individual system performance.
- If you reached the capacity limits for a single AS/400 system, you may try to step over these limits by implementing horizontal growth.

What is common for all these cases is that you want to obtain some sort of statistical distribution of your data among the nodes so that each system stores a certain percentage of the total number of records.

To obtain this type of distribution, select a partitioning key that does not change often or, even better, that is never updated. Secondly, the candidate partitioning key should have good statistical characteristics:

- It should have many different values. For example, a name or address normally have many different values while gender or marital status do not.
- There should be approximately the same number of records for each value of the field. If, for example, you use a city name as a partitioning key and it happens that most of your customers are located in one or two cities, this is a poor choice.

Another important aspect of choosing a partitioning key is which kinds of queries are most often performed over the file or which kinds of queries are most critical from the point of view of the time to get a result. The implementation of queries over distributed files has some specific features that cannot be ignored. This issue is discussed in more detail in Chapter 12, “Distributed Queries” on page 101.

Distributing a large file can be a long-running process that requires a lot of system resources. Therefore, it probably is not a good idea to experiment with an actual file.

Instead, we recommend that you perform some kind of investigation to predict the possible results of using different partitioning keys and different mappings between partitions and the nodes where they should reside.

When working with distributed files, the operating system performs two different mapping procedures:

- Mapping from the database file record to the partition number by applying a hashing function to the value of the partitioning key.
- Mapping from the partition number to an actual node where the record should be placed using partitioning information held inside of the distributed file object.

This gives us two variables to play with. By choosing a different partitioning key, we can change mapping from a record to a partition; and by changing the partitioning information in a node group object, we can change mapping from a partition to a node.

The following steps are an example of the kinds of experiments that can be done without actually distributing data:

1. A partitioning file can be created as described in Section 10.2.2, “Working With Partitioning Files” on page 68. This file contains some initial mappings between partitions and nodes. In this phase, we do not need to be concerned with actual node names. We can refer to nodes by their numbers.
2. Now we can run a query over a candidate file and a partitioning file to determine how many records go to each node.

   In our example, PARTFILE is a partitioning file, LINEITEM is the file to be distributed, and ORDERKEY is a potential partitioning key.

   The query:

```
SELECT COUNT(*) AS NUMBER_FOR_NODE, P.NGNNUM AS NODE_NUMBER
FROM LINEITEM T, PARTFILE P
WHERE HASH(T.ORDERKEY) = P.NGPTNN
GROUP BY P.NGNNUM
```

produced the following result for our example:

```
                                Display Data
                                            Data width . . . . . . :     28
Position to line  . . . . .   _____     Shift to column  . . . . . .   _____
....+....1....+....2....+...
NUMBER_FOR_NODE  NODE_NUMBER
     1,198,744           1
         1,056           2
         1,344           3
********  End of data  ********
```

Figure 23. Distribution of Records Between Nodes for Distributed File

This query shows that the vast majority of the records are allocated to node number 1 if we distributed the file LINEITEM using the partitioning key ORDERKEY and the partitioning file PARTFILE.

3. Now we can modify the partitioning file by:

   Adding more nodes and mapping some partitions to new nodes.
   Changing the mapping between partitions and nodes.

   You would now run the query again to look at the new distribution.

   You can also run the previous query for some other field in the file to verify if it is a better partitioning key.

4. This iterative process (step 2 and step 3) can be repeated as many times as needed before the optimal variant is found. It should be noted, however, that such a query can be a long-running one. In our case, file LINEITEM contained 1 200 000 records and the query ran for 3.5 minutes on a dedicated F95 system.

5. When the decision about partitioning is made, the partitioning file used in experiments can be used to create a node group that finally gives names to nodes. This node group and a chosen partitioning key can now be used to actually distribute the file.

10.1.2 Logical Separation of Data

You may want to distribute your data in a more logical, meaningful, and more controllable way than just statistically spreading data across systems. This approach is probably mandatory for customers that plan to use DB2 Multisystem in a geographically distributed environment.

There are different reasons to follow this strategy:

- You may want to allow applications to work on data that reside on the local node as much as possible. For example, suppose that most of the operations of a branch office relate to some clearly defined group of customers. The AS/400 system installed in this branch can have a portion of the entire customer file containing information about the customers serviced by this branch office.

- You want to offload some part of the file that is no longer considered current (archive type of data) from a central AS/400 system to some auxiliary (or remote) system.

All of these examples assume that there is no significant number of record accesses crossing the communications link between nodes. Again, an important factor is which kinds of queries are run over the file. See Chapter 12, “Distributed Queries” on page 101 for the discussion of query-related considerations.

Evidently, for this strategy, a potential partitioning key should not contain too many different values.

Under the best circumstances, you are able to use existing fields for your partitioning key. As we already explained in Section 9.2.3, “Changing A Node Group” on page 58, the system cannot map a partitioning key value directly to a node number and if two different values happen to hash to the same partition number, records that contain them cannot be separated.

Also, you cannot use a range of key values. In fact, key values that are close from a logical point of view can produce very different hash values. For example, ″New York″ maps to partition 730, but ″NewYork″ maps to partition 575.

However, it is important to point out that, for example, ″2036″ (four characters), 2036 (integer data type) and 2036.0 (decimal data type) all map to the same partition, in this case, 694. This is not a coincidence, of course, but a design choice. In spite of the difference in data types, records from different files but with the same value for the partitioning key can be located on the same system.

Before making a decision about a partitioning key, it is a good idea to analyze how its values are mapped to a partition number by a hash function. You can use an SQL statement similar to the following example:

```
SELECT DISTINCT ADDRESS, HASH(ADDRESS)
FROM CUSTOMER_FILE
```

If you cannot identify a suitable partitioning key, you probably need to add a special field to your file that identifies the location of each record. When a new record is added to the file, applications should also maintain this field, determining where it belongs.

Unfortunately, in that case, some modifications to your existing applications are required.

One example of a special partitioning key is found in Appendix B, “Hash Function Table” on page 137.

10.1.3 Visibility Node

Suppose you have a corporate AS/400 system running your applications and containing all of the database files needed for these applications. If you want to move some applications to a branch AS/400 system, you face a complex decision: either you move the application programs together with some data, or you only move the programs and keep all the data at the central system.

Relocating a part of a database to another system is not always easy. You may need to keep copies of the same file in sync, which sometimes requires efforts and special products.

Moving only programs is an easier option: the only challenge to face is remote database access. The AS/400 system already allows remote access to database files with DRDA or DDM support. DRDA requires SQL and, in general, you need to make changes to your applications when you move from a standalone system to a distributed environment.

On the other hand, DDM files do not support SQL or Query/400 and, again, you may need to make some application changes.

The visibility node concept implemented by DB2 Multisystem can provide a simple and elegant solution. Your data is concentrated on one or more systems, called the *data nodes*. Other systems, called *visibility nodes* only contain the description of the distributed files, but no records. In other words, the partition map is such that no partitions are assigned to the visibility nodes.

The visibility node can still access all of the current data in the distributed files. All of the applications can continue to work with the files as if they were local files. Interfaces such as Query/400 or SQL are supported and can be used.

Since no data reside on a visibility node, distributed file objects serve only as an interface to data nodes where all of the actual database activity is performed (processing, journaling activity, and so on).

Going back to our initial example, the system at the remote branch office does not need a lot of disk space since all of the database activities are concentrated at the data nodes.

From a CPU utilization point of view, the visibility node does not gain much from offloading the database workload to the data nodes. In fact, you may notice that even though all of the database operations are actually performed at the data nodes, the visibility node still has to spend some CPU cycles to perform the underlying DDM activities.

In other words, in a **transaction-oriented** environment, a visibility node has to sustain a CPU workload that is probably **higher** than the CPU workload it would carry if all the database access was local. In a **query intensive** environment, a visibility node actually **offloads** all of the CPU cycles involved with query processing to the data nodes. In this case, visibility nodes can greatly benefit from DB2 Multisystem. Even if your visibility node does not have a fast processor, query performance can be good provided that the data nodes are powerful enough. Another limiting factor is the communications media. If your query moves a lot of records across the line, the advantage of exploiting the computing power at the data nodes is defeated by the overhead introduced by the communications media.

Since a visibility node contains no records, no real data management activity is needed on this type of nodes. For example, although journaling must be started at the visibility node also, you never find any entries in the receivers and you do not need to manage the journals. Actually, you can even avoid starting the journaling function for the visibility node (see Section 11.2, “Journaling” on page 74) although it is not recommended.

To set up a visibility node, a node group object has to be created that allocates all of the partitions to the data nodes. You should use a partitioning file where you assign the 1024 available partitions to the data nodes and no partitions to the visibility nodes.

More information about working with node groups and partitioning files is found in Section 10.2, “Working With Node Groups and Partitioning Files.”

10.2 Working With Node Groups and Partitioning Files

In this section, we shall discuss some practical aspects of working with node groups and partitioning files.

10.2.1 Working With Node Groups

When a node group is created, two types of information are encapsulated inside of a node group object:

- Information about partition-to-node number mapping.
- Communications information for participating nodes.

Mapping information comes from a partitioning file if one was used during the node group object creation. If you want to change this mapping afterwards, you can either change mapping for a specific partition:

```
CHGNODGRPA NODGRP(GROUP2) PTNNBR(123) NODNBR(5)
```

or you can change mapping for a specific value of a partitioning key:

```
CHGNODGRPA NODGRP(GROUP2) CMPDTA(79) NODNBR(5)
```

Note: Both commands produce exactly the same results and have exactly the same meaning.

When you change the partition map in a node group object, this operation does not influence the record distribution of any existing file that refers to the node group. The partition map is extracted from a node group object at the time the file is created and stored within the distributed file object itself. To activate the changes made to the node group object, you need to use the CHGPF command. This command has a new parameter (*NODGRP*) that allows you to specify either the same node group as the one that was used at creation time or a different node group. In any case, the new partitioning information is extracted from the node group object and records are migrated as appropriate.

When you look at the contents of a node group object using the DSPNODGRP command, you see a list of nodes and a mapping table that maps partitions to different nodes in a node group. A node group object also contains complete communications information for every node, including:

Remote location name
Local location name
Remote network identifier
APPN node name

All of this communications information is taken from relational database directory entries and put inside of the node group object at node group creation time. This information cannot be displayed by the users and cannot be modified either.

The only way to change the communications information contained in a node group object is to restore the object from backup media. At restore time, the system reads a list of node names saved together with a node group object, and for each node in the list, the communications information of the corresponding relational database directory entries is extracted.

You can only restore a node group object on a system that is part of a node group and that has a Relational Database Directory that references all of the nodes specified in the node group.

If the node group object references some nodes that do not exist in the relational database directory, the node group object is restored but the system marks it as unusable. In this case, the system produces the message *CPF32C2* after the restore operation is completed as shown in Figure 24.

```
                        Additional Message Information

Message ID . . . . . :   CPF32C2       Severity . . . . . :   40
Message type . . . . . :   Diagnostic
Date sent  . . . . . :   08/21/96      Time sent  . . . . . :   11:12:59

Message . . . . :   Node group TEST in library A960202B cannot be used.
Cause . . . . . :   Node group TEST in library A960202B cannot be used on this
  system. A node group can only be used on the system on which it was created,
  or any other system that was originally specified when the node group was
  created. This system was not specified as being in the node group.  This
  error can also occur if the Relational Database (RDB) entries for all the
  systems in the node group did not exist in the RDB directory at the time of
  the restore.
Recovery  . . . :   Either create a new node group, restore this node group to
  a system that was originally specified at node group creation time, or use
  the WRKRDBDIRE command to add all the needed entries to the RDB directory
  and restore the node group again.
                                                                       Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 24. RSTOBJ Message for Node Group Object

Look for this message in your job log after you issue the restore command. This condition is not considered an error and the restore command reports a successful completion.

If a node group object is marked as unusable, you cannot use it to create or change a distributed file. When you try to refer to such a node group, the create or change command fails and the same message (*CPF32C2*) is sent to the job log.

You can still keep unusable node group objects on the system (for distribution purposes, for example). When an unusable node group object is restored, the system verifies the communications information again and the node group can become usable if all of the conditions are met.

When the DB2 Multisystem feature was designed, IBM interviewed several customers and consultants to find out how often they estimated the

communications configuration changes in real life. According to this survey, the communications configuration is not updated frequently. Therefore, for performance reasons, it was decided to keep a copy of the communications information within the node groups and the distributed files instead of retrieving it at run time. This design choice has the performance advantage of not requiring access to any object other than the file itself to determine the communications information.

Changing the communications configuration is potentially disruptive if you implement a DB2 Multisystem solution. If you need to change your configuration parameters (such as some remote location names), you still need the old configuration to be in place when you change your distributed files; otherwise it is impossible for the system where the change is applied to reach the remote portions of the files and propagate the changes.

Since this requirement is not always easy to comply with, the safest way to proceed is as follows:

1. First, modify the distributed file so that it becomes local. All of the records are sent to the local node:

   ```
   CHGPF FILE(YOURLIB/YOURFILE) NODGRP(*NONE)
   ```

2. Modify your communications configuration.
3. Update the relational database directories.
4. Create a new node group with the same partition map as the one you used previously or restore the old one.
5. Finally, use the CHGPF command to propagate the changes from the node group object to the distributed file objects.

This process, in circumstances where large files are involved, may take a long time. Also consider that the system where you are making the change needs to contain the entire distributed file at some point in time.

To restate what was already mentioned, all of the necessary information is copied from node groups to the distributed files at file creation or file change time. At run time, the system never needs to access any node group objects. This means that node groups do not really need to exist all the time. Moreover, when a change or create command references a node group, the object is only needed on the node where the command is issued.

However, we recommend you do not remove your node group objects from the systems where they were created as they may be used to distribute new files according to the same criteria used for some other files you have previously distributed.

10.2.2 Working With Partitioning Files

When a node group is created without specifying a partitioning file, the default partitioning is used. The default partitioning uniformly distributes partitions across nodes in a node group. For example, if you have four nodes, the partition map assigns 256 partitions to each node in the order specified in Figure 25 on page 69.

```
                              Display Node Group

 Node Group:   GROUP1         Library:   QTEMP

 Partition      Node
   Number      Number
         0           1
         1           2
         2           3
         3           4
         4           1
         5           2
         6           3
         7           4
         8           1
         9           2
        10           3
        11           4
        12           1
        13           2
        14           3
                                                                      More...
 F3=Exit   F11=Node Data   F12=Cancel   F17=Top   F18=Bottom
```

Figure 25. Displaying Partitioning Map for a Node Group Object

In most practical cases, a customized partitioning is needed and a partitioning file must be used.

Formally, the partitioning file is made of 1024 records of one field (a two-byte binary field specifying the node number). However, you can create partitioning files with extra fields, provided that the first field conforms to the previous requirement. This flexibility allows you to better document your partitioning file.

In our tests, we used one field for the node number and a second field for the partition number. The second field makes it easier to modify the partition map. In addition, this second field allows us to run some useful queries over the partitioning file itself (see Section 10.1.1, “Distribution of Workload Among Systems” on page 61 and Section 10.3, “Monitoring for Distribution” on page 70).

The following display shows DDS for the partitioning file:

```
Columns . . . :    1  71           Edit                        A960202B/SRCE
 SEU==> _____________________________________________________________ PARTFILE
       ********** Beginning of data **************************************
001.00      A                                     UNIQUE
002.00      A          R PARTREC
003.00      A            NGNNUM         4B        RANGE(1 32)
004.00      A            NGPTNN         4B        RANGE(0 1023)
005.00      A          K NGPTNN
       ********** End of data ********************************************
```

Figure 26. Sample DDS for a Partitioning File

A partitioning file can also be created using SQL:

```
CREATE TABLE PARTFILE(NGNNUM SMALLINT NOT NULL WITH DEFAULT,
                      NGPTNN SMALLINT NOT NULL WITH DEFAULT,
                           UNIQUE (NGPTNN))
```

You can use DFU to populate the file with records or use interactive SQL to insert records in the file.

If you already have an existing distributed file, you can extract partitioning information from it and populate the partitioning file (you can create a simple distributed file for this purpose with default partitioning).

Use the following steps:

1. Extract partitioning information from an existing distributed file:

   ```
   DSPFD FILE(LINEITEMD) TYPE(*NODGRP)
     OUTPUT(*OUTFILE) OUTFILE(QTEMP/DSPFD)
   ```

2. Then copy partitioning information from the resulting OUTFILE to your partitioning file:

   ```
   CPYF FROMFILE(DSPFD) TOFILE(PARTFILE) MBROPT(*REPLACE)
       INCCHAR(NGRDBN 1 *EQ ' ') FMTOPT(*MAP *DROP)
   ```

   The same result can be achieved using SQL:

   ```
   INSERT INTO PARTFILE (NGNNUM, NGPTNN)
     SELECT NGNNUM, NGPTNN
     FROM QTEMP/DSPFD WHERE NGRDBN = ' '
   ```

Now you can change the mapping of an individual partition with DFU or interactive SQL, referencing the partition number specified in the second field.

It is important to remember that when a partitioning file is used to create a node group object, this file is processed in an arrival sequence (that is, record 0 corresponds to partition number 0 and so on up to record 1023, which corresponds to partition number 1023). Therefore, when using the partitioning file of a proposed structure, make sure that the arrival sequence corresponds to partition number sequence of the second file. To guarantee this, the RGZPFM command can be used:

```
RGZPFM FILE(PARTFILE) KEYFILE(*FILE)
```

10.3 Monitoring for Distribution

When working with a distributed file in a real environment, it is important to monitor regularly for the changes in distribution.

It is possible, for example, that the distribution of values of a chosen partitioning key can change with time. As a result, some partitions can grow faster than expected. The most obvious indicator you can regularly monitor is the number of records stored on every node. The DSPFD command can be used for this purpose:

```
                              Display Spooled File
File  . . . . . :   QPDSPFD                          Page/Line   6/25
Control . . . . .   B_______                        Columns     1 - 78
Find  . . . . . .   ____________________________
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
 Format       Fields  Length  Identifier
  LINEITEM         16      161  400FF819FB64B
    Text . . . . . . . . . . . . . . . . . . . . :
  Total number of formats  . . . . . . . . . . . :           1
  Total number of fields . . . . . . . . . . . . :          16
  Total record length  . . . . . . . . . . . . . :         161
                                        Member List
                          Source Creation    Last Change                  Dele
  Member           Size    Type  Date       Date     Time      Records    Reco
  LINEITEM      128983040        07/25/96 08/14/96 12:10:32     857210
    Text:
  Total number of members  . . . . . . . . . . . :               1
  Total number of members not available  . . . . :               0
  Total records  . . . . . . . . . . . . . . . . :          857210  1
  Total deleted records  . . . . . . . . . . . . :               0
  Total of member sizes  . . . . . . . . . . . . :       128983040
                                                                          Bottom
F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys
```

Figure 27. Displaying Description of a Distributed File

In Figure 27, you can see that in **1**, the system shows how many records are located on the local node. The SQL functions **NODENAME** and **NODENUMBER** can also be used to determine with a single query how many records are distributed on every system in the node group. These functions can also be used with the OPNQRYF command.

Another important consideration is how record accesses are distributed. Usually, the file is distributed with the assumption that most record accesses are local with only an occasional reference to records in remote parts of the file.

To detect changes in the pattern of record accesses, performance reports produced by the Performance Tools/400 licensed program product can be used.

When a program or query opens a distributed file, one or more DDM jobs are started on each of the remote nodes where parts of the file reside. Performance Tools can be used to spot excessive communications traffic due to these DDM jobs. This indicate that the partitioning criteria are not optimal.

If you experience some undesirable skew in records distribution, or more accesses to remote records than planned, you may want to reconsider the distribution strategy for the file and either choose a different partitioning key or change the partition map.

DB2 Multisystem allows you to make all of these changes, including adding and removing nodes, in a simple way without having to restart from a local file and redistribute every record.

Before you change the way the file is distributed, it is wise to estimate how many records are migrated as a result of the change. The following procedure can be helpful in performing this estimate:

1. Create a table to contain an intermediate result:

```
CREATE TABLE MIGRATION
          (NODEFROM SMALLINT NOT NULL WITH DEFAULT,
           NODETO SMALLINT NOT NULL WITH DEFAULT)
```

2. Use the partitioning file that was used to initially distribute the file (see Section 10.2.2, “Working With Partitioning Files” on page 68) and make any changes to the file to reflect the new partitioning you are going to implement.

3. Select information about all of the records that are moved when you change a distributed file:

```
INSERT INTO MIGRATION (NODEFROM, NODETO)
 SELECT NODENUMBER(T), P.NGNNUM
 FROM LINEITEM T, PARTFILE P
 WHERE HASH(T.ORDERKEY) = P.NGPTNN
  AND P.NGNNUM <> NODENUMBER(T)
```

   In this example, LINEITEM is the distributed file that partitioning is going to change, PARTFILE is the modified partitioning file, and ORDERKEY is the partitioning key being considered.

4. Produce the report on the records that are moved:

```
SELECT NODEFROM, NODETO, COUNT (*)
FROM MIGRATION
GROUP BY NODEFROM, NODETO
ORDER BY 1
```

   You receive a report similar to the one shown in Figure 28.

```
                              Display Data
                                          Data width . . . . . :      33
Position to line  . . . . .  _____     Shift to column  . . . . . .  _____
....+....1....+....2....+....3...
NODEFROM  NODETO     COUNT ( * )
      1        2             736   1
      1        3             968
      2        1         171,866
      2        3             190
      3        1         171,372
      3        2             172
********  End of data  ********
```

Figure 28. Estimating Record Migration When Redistributing File

1 In our example, this line shows that 736 records must be moved from node 1 to node 2 to accomplish the change.

5. Now you can repeat the preceding procedure, testing different ways to redistribute the file. (Be careful to remove all of the records from the MIGRATION table, or remove and re-create it to get rid of obsolete data.)

6. When you are satisfied with the results, a partitioning file used in the preceding procedure (PARTFILE) can directly serve as input to the CRTNODGRP command to create a new node group with new control information for distribution.

7. Finally, the distributed file can be redistributed using the new node group object as a source of information for the change.

Chapter 11. Managing Distributed Files

Once you have designed your DB2 Multisystem environment, you can start planning for any change to database management. Distribution, although facilitated by DB2 Multisystem, introduces some additional complexity even though the new environment is based upon the solid foundation of familiar AS/400 functions.

In this chapter, we discuss how to manage a distributed database in a DB2 Multisystem environment. Here you can find information on disaster recovery procedures, security, National Language Support, and work management. We also deal with some aspects of the redistribution of distributed files.

A key concept for understanding how distributed files work is symmetry. Every node where a part of a single distributed file is located should have an identical configuration in all possible respects. Even when absolute symmetry is not technically required (see Section 11.5, “Security Considerations” on page 85, for example), managing a distributed environment can become much easier if every node is equivalent.

Another aspect of symmetry is that all nodes are equal in terms of performing any kind of processing on a distributed file. As soon as a distributed file is created, or a local file is changed to become distributed, there is no way to tell where the file was originally created.

Most file operations (normal file I/O, running queries, starting and stopping journaling, changing file attributes, controlling security, and so on) that relate to the distributed file as a whole can be initiated from any node where any part of the file resides.

11.1 Availability

One important aspect of distributed computing is availability. So far in this book, the basic assumption has been that all of the participating nodes were always available and all of the communications links active.

When this is not the case, many operations with distributed files are not allowed. In particular, opening a distributed file with the native database interface will fail if one of the nodes become out of reach. This limitation implies that applications that use distributed files cannot operate if a node or a link becomes unavailable.

An important exception is represented by some types of queries that can run on the local part of distributed files without accessing the remote nodes. This is discussed in Chapter 12, “Distributed Queries” on page 101. However, availability issues should be taken into great considerations once you decide to run transactional activities in a DB2 Multisystem environment.

Some functions can tolerate a communications loss and perform only on the local part of the file (or on currently available nodes only). This is mentioned later when individual issues are discussed. But such situations do lead to the loss of symmetry and normally require some corrective actions.

11.2 Journaling

Journaling is a function that is essential to any disaster recovery strategy. For details about journaling support on the AS/400 system, see the *OS/400 Backup and Recovery - Advanced*, SC41-4305.

For distributed files, journaling is performed independently on each node. Journal entries are placed in a journal receiver on each node independently. They only relate to data changes that take place in the local portion of the file.

Journal management and save/restore procedures are performed independently on every node. We discuss it in more detail later.

As usual, you can start journaling by using the STRJRNPF command for a distributed physical file and by using the STRJRNAP command for the distributed logical files and for the access path associated with the distributed physical file.

DB2 Multisystem requires that on the every node you have create a journal by the same name and locate it in the same library as you specify in the STRJRNPF command. If you specify *LIBL for the library name the STRJRNPF command, the library name is first resolved on the local node and the resolved name is propagated.

When you issue the STRJRNPF command to start journaling for a distributed file, journaling is first started on the local node where the STRJRNPF command is run and then on the remaining participating nodes. The list of nodes is processed sequentially; if one of the nodes cannot be reached or the journal cannot be found, the system does not even try to start journaling on the remaining nodes and you receive a message similar to the one shown in Figure 29.

```
                        Additional Message Information

Message ID . . . . . :   CPF704B       Severity . . . . . :   40
Message type . . . . . :   Diagnostic
Date sent  . . . . . :   08/20/96      Time sent  . . . . . :   16:24:39

Message . . . . :   Journaling started locally but distributed requests
  failed.
Cause . . . . . :   1 files started journaling locally.  However, 1 of these
  files are distributed and the start journal request could not be distributed
  to one or more systems in the files' node group.
Recovery  . . . :   Use the Display Job Log (DSPJOBLOG) command to see the
  previous error messages to determine which start journal requests were not
  distributed.  If you want to end journaling for these files because the
  start journal command did not distribute, use the End Journaling Physical
  File (ENDJRNPF) command.
    Save the files that started journaling to allow for later data recovery
  using the journal.
                                                                        Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 29. Starting Journaling for Distributed File

If this happens, you should correct the problem and start journaling at the various nodes where it could not be started with the first attempt.

Technically, it is allowed to have some parts of the distributed file being journaled and other parts being not journaled. You can also force the usage of different journals on the various nodes. We recommend you avoid both of these conditions for consistency reasons and ease of management.

Journaling physical files is the basis for many recovery procedures. It is also a prerequisite for commitment control (see also Section 11.3, “Commitment Control” on page 76). Therefore, STRJRNPF has a global effect on the entire distributed file.

The situation is different for access path journaling, which is a facility intended to reduce the recovery time after abnormal IPLs. Because of this, the STRJRNAP command only starts journaling on the node where the command is issued. Distributed files are no different from ordinary files as far as access path journaling is concerned.

As usual, you should save all of the related objects to backup media after you start journaling. This provides a starting point for future recovery procedures.

Stopping journaling works the same as starting journaling. The operational behavior of the ENDJRNPF command is the same as the STRJRNPF command. If there are problems related to stopping journaling on remote nodes, journaling is ended on the local node and a message is sent to the job log:

```
                        Additional Message Information

Message ID . . . . . :   CPF704C       Severity . . . . . :   40
Message type . . . . . :   Diagnostic
Date sent  . . . . . :   08/20/96      Time sent  . . . . . :   16:22:18

Message . . . . :   Journaling ended locally but distributed requests failed.
Cause . . . . . :   1 files ended journaling locally.  However, 1 of these
  files are distributed and the end journal request could not be distributed
  to one or more systems in the files' node group.
Recovery  . . . :   Use the Display Job Log (DSPJOBLOG) command to see the
  previous error messages to determine which end journal requests were not
  distributed.  If you want to start journaling for these files because the
  end journal command did not distribute, use the Start Journaling Physical
  File (STRJRNPF) command.



                                                                       Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 30. Ending Journaling for a Distributed File

Once started, journaling is performed independently on every node in a network. Also, journal receiver management should be planned for each node separately. As usual, journal receivers should be detached from a journal and saved to backup media and a new journal receiver should be attached.

Journal attributes can be set independently on each node to tell the operating system that the journal receiver should be detached from a journal automatically and a new journal receiver automatically created. All of the preceding processing has only local significance and has no relation to remote nodes whatsoever. All of the necessary coordination of these tasks should be planned and is the responsibility of a system administrator.

In particular, journals and journal receivers residing on different nodes are unrelated objects. Therefore, the system does not let you use journal receivers from one node in a node group to be used for applying changes to the same distributed file on another node.

Since journaling is a local activity, applying journal changes to a journaled file with the APYJRNCHG command also has only local significance. The same is true for the RMVJRNCHG command as well.

When applying or removing journaled changes to a journaled distributed file, you certainly want to coordinate these activities on all nodes in a network. Making sure that this activity takes place in a consistent way across the nodes is a responsibility of the database administrator.

For related information, see Section 11.4, “Disaster Recovery Planning” on page 79.

11.3 Commitment Control

When updating distributed files, an application can request that a consistent transaction is performed across all parts of the file on all nodes in a node group. To request this support, a distributed file should be opened under commitment control. After commitment control is started for the application′s job, COMMIT and ROLLBACK operations are performed consistently over all parts of a distributed file.

A standard requirement for commitment control on the AS/400 system is that your physical files should be journaled. If you plan to use commitment control, you should make sure that all parts of a distributed file are journaled on every node in a node group.

As for usual file processing, files opened under commitment control can be journaled to the same or different journals. It is recommended that you journal all of the files to the same journal. This simplifies the backup/recovery procedures and introduces less overhead for commit and rollback processing. More about journaling distributed files is available in Section 11.2, “Journaling” on page 74.

When a distributed file is opened under commitment control, the system starts a distributed unit of work over all nodes that contain parts of the distributed file. Every update to a distributed file generates an entry in a journal on a node where an updated record resides.

To coordinate COMMIT and ROLLBACK operations, the system uses concepts of the distributed unit of work and two-phase commit of the Distributed Relational Database Architecture (DRDA) support that is part of the base OS/400. When a COMMIT or ROLLBACK operation is requested, a two-phase protocol is performed that coordinates the execution of these operations throughout all of

the participating nodes. On every AS/400 system, there is a system job (*QLUR*) that processes all two-phase COMMIT and ROLLBACK operations that this system is participating in.

When a distributed file is opened by an application, a DDM job is started on every participating node that performs all of the accesses to respective parts of the distributed file (including updates) on behalf of the application′s job. (More information about jobs when you are working with distributed files is in Section 11.7, “Managing Jobs” on page 91).

If a communications link becomes unavailable during the processing of distributed files under commitment control, this situation has different effects on the various jobs in the node group. Factors involved are the type of failure and the status of the job at the time of the failure:

- Some jobs are abnormally terminated as a result of a communications failure. Database changes are automatically rolled back.
- The application′s job receives an error notification on the next attempt to access the file.
- Some jobs related to the originating application′s job on some nodes may not be affected by the failure.

The safe action for an application is to perform a ROLLBACK operation. As a result, changes made by the application′s job itself are rolled back. Changes on nodes that are still connected are also rolled back. The database is brought back into a consistent state, corresponding to the completion of the last committed transaction.

At this point, you can investigate the failure and restore communications. Processing restarts from a consistent state.

There is a possibility that a communications failure happens just in the middle of the COMMIT processing. Due to the nature of two-phase protocols, there is a small time frame where the systems cannot establish if all of the nodes have committed the transaction. As a result of this rare occurrence, the originating job hangs until either normal operations are restarted or an operator forces a commit or rollback. See *DB2/400 Advanced Database Functions*, GG24-4249, for a more comprehensive discussion about this situation.

Once the undecided state is resolved, your environment is in a situation similar to what is described on page 77. The applications must be restarted, but at least you can restart from a safe consistent state.

There can be extreme cases when a communications break is a result of some disaster that leads to a data loss and some major changes in an environment, for example:

- The disk unit crashes on some nodes and the system needs a full reload.
- A disaster results in the permanent loss of one or some of the systems and all of the data must be restored from a backup.

In this case, the data loss has happened already and can no longer be avoided. Then you have to proceed to recovery procedures to restore the database to some reliable state. But we mentioned previously, the commitment definition in an undecided state is a persistent entity and you need to force it away from the system.

To perform this task, use the WRKCMTDFN command:

```
WRKCMTDFN JOB(*ALL) STATUS(*UNDECIDED)
```

```
                        Work with Commitment Definitions
                                                                System:   SYSTEM03
Type options, press Enter.                3
  5=Display status   12=Work with job   14=Forced commit
  16=Forced rollback ...
  2
     Commitment                                        Resync In
Opt  Definition  Job         User        Number  Progress
16   QDBAGILE    P23YDTYKE   A960202B    002613  YES
                                                                          Bottom
F3=Exit   F5=Refresh   F9=Command line   F11=Display logical unit of work ID
F12=Cancel   F16=Sort by logical unit of work ID   F24=More keys
```

Figure 31. Forcing ROLLBACK for Commitment Definition

On the WRKCMTDFN display, use Option 16 (Forced rollback 2) for the commitment definition you want to terminate and to force a ROLLBACK operation for this unsuccessful transaction. Once you confirm that you really want to force a rollback, press F23 to see additional options. The following display is shown:

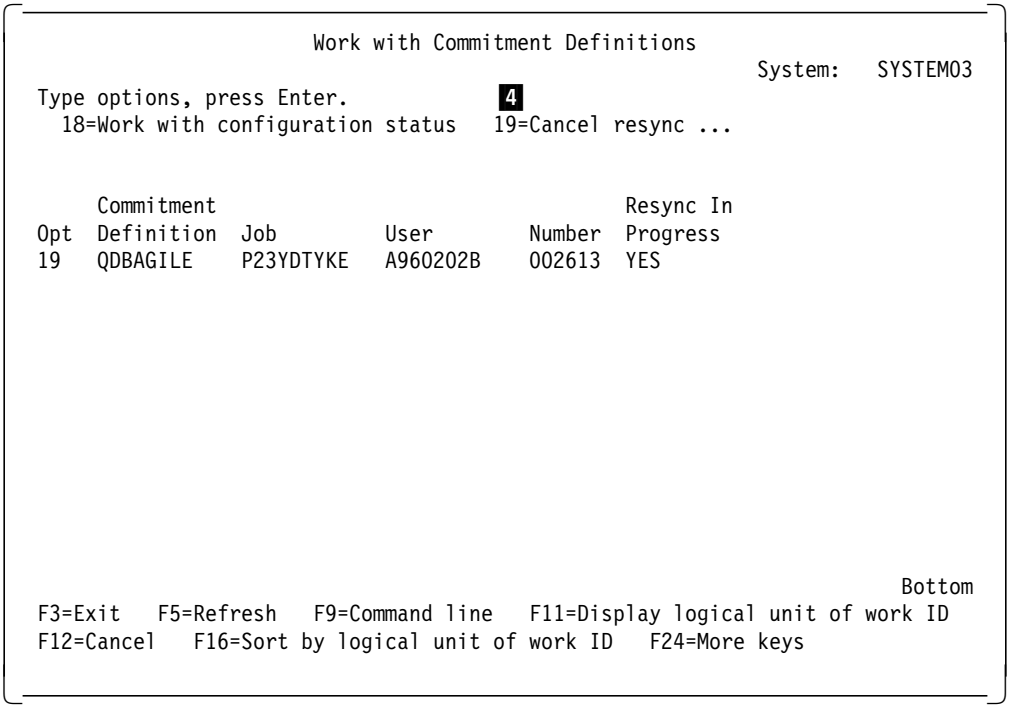


Figure 32. Cancelling Resynchronization for Commitment Definition

Enter option 19 (Cancel resync **4**) beside the commitment definition to cancel the resynchronization process with remote nodes. After the request for resynchronization cancellation is accepted, you must wait for several minutes before it is processed.

If the application's job that was the initiator of the transaction is still active (no IPLs were performed), the application receives a failure notification from the COMMIT operation. The job is no longer suspended. Now you can remove the job from the system and start restoring your database.

Actually, if you think you know what the state of your file is at the moment of the failure, you can decide to avoid the lengthy restore procedure. You still must resolve the undecided state of the commitment definition. You can use the forced rollback procedure (option 16 **2** in Figure 31 on page 78) as previously described. Otherwise, if you have reasons to believe that changes on remote nodes were committed, you can use the forced commit procedure (option 14 **3** in Figure 31 on page 78).

In this case, the system cannot guarantee that your distributed file ends up in a consistent state and the responsibility for using this option and possibly force the database in an inconsistent state is yours. You should verify later on that your database still is in a consistent state manually or with an ad-hoc application.

11.4 Disaster Recovery Planning

As we mentioned at the beginning of this chapter, every portion of a distributed file is an individual object. In particular, save/restore operations on distributed files have local scope. In other words, in order to save or restore a distributed file, you need to process every portion separately.

11.4.1 How to Save Distributed Files

To save a distributed file, you can use the SAVLIB or SAVOBJ commands. As we explained earlier, you only save the portion of the entire distributed file that is located on the node where the save command is issued.

As a result of a save, you probably want to end up with a set of tapes containing a consistent copy of all the portions of a distributed file. It is also convenient to be able to execute the save procedure from a single workplace rather than having to sign on to every system in the node group.

The DB2 Multisystem feature provides a solution for both of these requirements:

- When you want to save an object, you normally use the ALCOBJ command to place an exclusive lock and prevent anyone from modifying the object before the save operation is complete. The function of the ALCOBJ command has been extended so that it has global effect on a distributed file. You can establish a lock on an entire distributed file by issuing the ALCOBJ only once and on any of the systems in the node group.

  The jobs holding the lock on the remote systems are DDM jobs that represents your job′s activities in relation to the specific distributed file. More information about these jobs is found in Section 11.7, “Managing Jobs” on page 91.

  To release the lock, use the DLCOBJ command.

- A distributed file object can be a target of the SBMRMTCMD command as well as a DDM file.

  When you issue the SBMRMTCMD command and refer to a distributed file in the DDMFILE parameter, the requested command is executed on every remote node of the distributed file.

If you use journaling for your distributed files, you need to save all of the journal receivers on all participating nodes simultaneously in order to have a backup set reflecting a single consistent point of recovery across all nodes. As we discussed in Section 11.2, “Journaling” on page 74, journals and journal receivers are independent objects on different nodes of a node group. You need to save the journal receivers in a consistent way as well since the complete update history of a single distributed file is contained in the combination of the journal receivers of each node.

In summary, if you want to make a consistent save of a distributed file across all the participating nodes, you can use the following procedures:

1. Place an exclusive lock on all parts of the distributed file:

   ```
   ALCOBJ OBJ((A960202B/LINEITEM *FILE *EXCL))
   ```

2. Save the local part of the file to a backup tape:

   ```
   SAVOBJ OBJ(LINEITEM) LIB(A960202B) DEV(TAP01) OBJTYPE(*FILE)
   ```

3. Save all of the remote parts of the file to respective backup tapes (it is assumed that operators have loaded the correct tapes on the various systems):

   ```
   SBMRMTCMD
      CMD(SAVOBJ OBJ(LINEITEM) LIB(A960202B)
                 DEV(TAP01) OBJTYPE(*FILE))
      DDMFILE(A960202B/LINEITEM)
   ```

4. Create a new journal receiver on the local system:

```
CRTJRNRCV JRNRCV(A960202B/JRNRCV2)
```

5. Create new journal receivers on the remaining systems for the file:

```
SBMRMTCMD
   CMD( CRTJRNRCV JRNRCV(A960202B/JRNRCV2))
   DDMFILE(LINEITEM)
```

6. Detach the current journal receiver from a journal and attach a new journal receiver on the local node:

```
CHGJRN JRN(A960202B/JOURNAL) JRNRCV(A960202B/JRNRCV2)
```

7. Perform the same procedure for the remote nodes:

```
SBMRMTCMD
   CMD(CHGJRN JRN(A960202B/JOURNAL) JRNRCV(A960202B/JRNRCV2))
   DDMFILE (A960202B/LINEITEM)
```

8. Save the detached journal receiver on the local system:

```
SAVOBJ OBJ(A960202B) LIB(A960202B)
       DEV(TAP01) OBJTYPE(*JRNRCV)
```

9. Save the journal receivers you just detached on the remote nodes:

```
SBMRMTCMD
  CMD(SAVOBJ OBJ(JRNRCV1) LIB(A960202B)
             DEV(TAP01) OBJTYPE(*JRNRCV))
  DDMFILE (A960202B/LINEITEM)
```

10. Once saved, delete this journal receiver on the local node: journal receiver on a local node:

```
DLTJRNRCV JRNRCV(A960202B/JRNRCV1)
```

11. Do the same operation for the remote nodes:

```
SBMRMTCMD
   CMD(DLTJRNRCV JRNRCV(A960202B/JRNRCV1))
   DDMFILE(A960202B/LINEITEM)
```

12. Release the lock from the file on all nodes:

```
DLCOBJ OBJ(A960202B/DISTFILE *FILE *EXCL)
```

13. You now have a set of tapes (one for each node) that together hold a complete backup image of a distributed file.

Please note that since update activity on different nodes for the distributed file may vary, so the number and names of journal receivers for all nodes can be different. You must track saved journal receivers separately for each node or develop a single consistent journal receiver management policy for all nodes. In the previous example, journal receivers were changed simultaneously on all nodes and were named the same (although their size is different). You can use a similar convention or some other one that is convenient for you.

11.4.2 How to Restore Distributed Files

The same as for save operations, restore operations have only local effects on the system where they are performed. It is the customer's responsibility to keep parts of the distributed file in synchronization on different nodes in a network.

The same as with save commands, you can use the ALCOBJ command to prevent access to the file being restored and the SBMRMTCMD command to

simplify sending commands to AS/400 systems where the file is distributed. This is explained in Section 11.4.1, “How to Save Distributed Files” on page 80.

A portion of the distributed file can be restored only to the same system and to the same library it was saved from. If you try to restore a portion of distributed file to an incorrect node, the restore operation fails and produces the message reported in Figure 33.

```
                        Additional Message Information

Message ID . . . . . :   CPF32C1       Severity . . . . . :   50
Message type . . . . . :   Diagnostic
Date sent  . . . . . :   08/26/96      Time sent  . . . . . :   14:23:28

Message . . . . :   Distributed file NATION2 and saved file not the same.
Cause . . . . . :   The restore of file NATION2 in library A960202B was not
  done, or the partition key attribute of the file has been removed. The
  reason code is 1. The reason codes and their meanings are as follows:
    01 - The saved file's node ûûû·QQQNE     *LIB, is not the same as the
  file's node PRODUCTION.
    02 - The saved file's library is different than the library that the file
  is being restored to. The partition key attribute of the file has been
  removed.
    03 - The saved file's node ûûû·QQQNE     *LIB is different than the node
  of the system. The partition key attribute of the file has been removed.
Recovery  . . . :   Do one of the following based on the reason code shown,
                                                                      More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 33 (Part 1 of 2). Restoring Distributed File to Incorrect Node

```
                        Additional Message Information

Message ID . . . . . :   CPF32C1       Severity . . . . . :   50
Message type . . . . . :   Diagnostic

  and then try the request again.
    01 - Restore the file to the correct node PRODUCTION, or use the
  ALWOBJDIF(*YES) parameter to restore the file. The file can also be restored
  by restoring the file to a different library than the saved library.
    02 - The partition key attribute of the file has been removed from the
  file because the FROM and TO libraries of the save and restore are
  different. The file has been restored as a local file.
    03 - The partition key attribute of the file has been removed from the
  file because the node of the file is not the same as the node of the system.
  The file has been restored as a local file.




                                                                      Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 33 (Part 2 of 2). Restoring Distributed File to Incorrect Node

If you intentionally want to restore a distributed file to a different node, you can do it if you restore the file to a different library or if you specify the ALWOBJDIF(*ALL) parameter on the RSTOBJ or RSTLIB commands. In this case, the file is restored just as if it was an ordinary file and loses all of the attributes that are specific for a distributed file. In other words, the restored file is not a distributed file. This operation can be useful if you want to restore the contents of an entire distributed file onto a single system.

If you lose the availability of one or more nodes in your network, you cannot use distributed files (some exceptions are mentioned in Chapter 12, “Distributed Queries” on page 101). If one of the systems in the node group becomes permanently unavailable, the portions of distributed files that resided there cannot be restored on the remaining systems because each portion becomes a separate local file. You can redistribute your records removing the missing system from the node groups, but to perform this redistribution, the missing system should be available because DB2 Multisystem tries to migrate the records out of that system onto the remaining ones.

If a system is lost, you are left with two alternatives:

- Replace the missing system with a new one and configure it the same way in terms of communications and relational database directory. You can then restore the portions of the distributed files onto this system and redistribute them if you want.
- Restore all portions of each distributed file on a single system. For each distributed file, you should create a local copy containing all of the records, and once you have rebuilt the entire file, you can redistribute the records among the remaining systems.

Let us now have a look at a more complex case: you have lost one of the nodes due to some type of disaster and need to recover all of the data stored in a distributed file to a non-distributed file on a single AS/400 system. It is assumed that you have all of the necessary backup tapes available: all of the portions of the distributed file, the journals, and the journal receivers for every system in the network.

The procedure for recovery owes its complexity to the restrictions imposed by the journaling environment: the journal, the journal receivers, and the journaled file should be restored to the same library where they were saved from, but in a distributed environment, all the related objects have the same name (library, journals, and the file itself). When you try to concentrate the various portions on a single system, you need to move the objects around as we show in the following procedure. In our example, we used a set of save files. In a real environment, you probably use tapes as a backup media.

1. Perform step 2 to step 10 for every part of the distributed file.
2. Clear the original library of all objects. Primarily, you need to remove the distributed file, its journal and the associated receivers.

   ```
   CLRLIB LIB(A960202B)
   ```

3. Restore the file to a library that is **different** from the library where it was saved from. This makes this part of a distributed file just an ordinary local file.

   ```
   RSTOBJ OBJ(LINEITEM) SAVLIB(A960202B)
          DEV(*SAVF) SAVF(SAVLIB/FILE1)
          RSTLIB(QTEMP)
   ```

4. Move the file back into the original library. If you do not do this, you will have problems later when you apply the journal changes to the file.

```
MOVOBJ OBJ(QTEMP/LINEITEM) OBJTYPE(*FILE) TOLIB(A960202B)
```

5. Restore the corresponding journal to the original library.

```
RSTOBJ OBJ(QSQJRN) SAVLIB(A960202B)
       DEV(*SAVF) SAVF(SAVLIB/JRN1)
```

6. Start journaling for the file created in step 4 using the journal restored in step 5. This re-creates a journaling environment.

```
STRJRNPF FILE(A960202B/LINEITEM)
         JRN(A960202B/QSQJRN) IMAGES(*BOTH)
```

7. Restore any journal receivers saved for this part of the distributed file since the last save of the file object itself.

```
RSTOBJ OBJ(*ALL) SAVLIB(A960202B)
       DEV(*SAVF) OBJTYPE(*JRNRCV) SAVF(SAVLIB/JRNRCV1)
```

8. Now you can apply the journal changes in the usual way. On the APYJRNCHG command, specify the journal receiver range restored in step 7. Once this step is complete, the restored file contains all of the data up to the time when the last journal receiver had been saved.

```
APYJRNCHG JRN(A960202B/QSQJRN) FILE(A960202B/LINEITEM)
          RCVRNG(A960202B/QSQJRN0001  A960202B/QSQJRN0003)
          FROMENT(*FIRST) TOENT(*LAST)
```

9. End journaling for the restored file.

```
ENDJRNPF FILE(A960202B/LINEITEM)
```

10. Move the file object away from the library and rename it to some temporary name. You will use it later on.

```
MOVOBJ OBJ(A960202B/LINEITEM) OBJTYPE(*FILE) TOLIB(TEMPLIB)
RNMOBJ OBJ(TEMPLIB/LINEITEM) OBJTYPE(*FILE) NEWOBJ(LINEITEM1)
```

11. You need to repeat the previous steps for every portion of the distributed file. Once finished, all of the necessary pieces of the original file are available and you can collect all of the data in one single local file using the CPYF command.

During a complex recovery procedure, you might need to delete some of the portions of a distributed file while some of the links among the systems in the node group are still not available. You notice that the Delete File (DLTF) command tries to delete the entire distributed file and, therefore, tries to access all of the nodes in the node group. If this is not possible, after a timeout, a message is sent to the system operator message queue (QSYSOPR message queue):

```
                        Additional Message Information

Message ID . . . . . :   CPA32B4       Severity . . . . . :   99
Message type . . . . . :   Inquiry
Date sent  . . . . . :   08/06/96      Time sent  . . . . . :   15:44:33

Message . . . . :   File LINEITEM on node SYSTEM03 can not be deleted. (C I)
Cause . . . . . :   Distributed file LINEITEM in library A960202B on node
  SYSTEM03 can not be deleted because communications to the node can not be
  connected.
Recovery  . . . :   Enter one of the following:
    C -- The file will not be deleted.
    I -- Attempt to delete the file.
Possible choices for replying to message . . . . . :
  C -- The file will not be deleted.
  I -- Attempt to delete the file.

                                                                      Bottom
Type reply below, then press Enter.
Reply  . . . .   _____________________________________________________________

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 34. Deleting Distributed File Object When Node Not Accessible

If you want to delete the file anyway, reply I (Ignore) to this message. As a result, the system deletes only the local part of the distributed file. Later, you probably need to reach the remote systems and delete any other parts of the distributed file still in existence.

11.5 Security Considerations

Access to distributed files is governed by the same security mechanisms that regulate any other OS/400 object.

When the GRTOBJAUT command is issued against a distributed file, the system first grants the requested level of authority to the local portion of the distributed file object. Then the system goes through the list of nodes stored within the file description, accesses each node in turn and applies the same command to each node. This implies that you create the same user profiles on every node in the node group.

If some node in a network is not accessible (due to a communications problem) or there are some other problems on the remote node (for example, the user profile does not exist), the GRTOBJAUT command reports the failure for this specific node but the command is still executed on the remaining nodes. In this case, you can reach a non-symmetrical configuration where the user profile has authority on some nodes but not on some others. To correct the security configuration, you must access the nodes where the command did not take place and manually issue the GRTOBJAUT command.

The same considerations equally apply to the RVKOBJAUT command.

Suppose you have a distributed *CUSTOMER* file partitioned according to geographical criteria. Every system has a portion of the entire customer file with data related to local customers only. You may want to restrict access to data in

such a way that users can update records for the local customers, but can only read customer records located on remote nodes.

You may give users a different level of authority to different parts of the distributed files by way of an authorization list. When the GRTOBJAUT command is used to grant authority to the distributed file through an authorization list, the process is essentially the same as for individual user profiles. You still have to create authorization lists with the same name on every node of the node group. But there is no requirement that the authorization list contents should be the same.

Controlling user profile authority to the authorization list, you can legitimately have different access rights to the distributed file for the same user profile on different nodes in the node group. For example, if you have user profile USER1 and authorization list AUTL1 on systems SYSTEM01 and SYSTEM02, you can grant authorities in the following way:

- Grant authority to the distributed file to the authorization list:

  ```
  GRTOBJAUT OBJ(A960202B/LINEITEM) OBJTYPE(*FILE) AUTL(AUTL1)
  ```

  This command associates the distributed file LINEITEM to authorization list AUTL1 on both systems.

- On system SYSTEM01, grant change authority to USER1 to the authorization list AUTL1:

  ```
  ADDAUTLE AUTL(AUTL1) USER(USER1) AUT(*CHANGE)
  ```

- On system SYSTEM03, grant read-only authority to USER1 to the authorization list AUTL1:

  ```
  ADDAUTLE AUTL(AUTL1) USER(USER1) AUT(*USE)
  ```

If the home system for user USER1 is SYSTEM01, then the previous sequence of commands granted USER1 read and update rights on the local data but only read access to remote data.

In a similar way, using an authorization list, you can create a configuration where different nodes in a node group have a different population of users (user profiles).

In order for the security mechanism to work, you have to make sure that DDM jobs at the remote systems run with the same user profile as the originating job at the local system. All accesses to remote parts of the distributed file on behalf of the user are performed through DDM jobs. The user profile that a DDM job uses is determined by the way communications entries are configured in the corresponding subsystem. By default, all communication requests are routed to the QCMN subsystem (when QCTL is chosen as the controlling subsystem) or QBASE subsystem (if QBASE is the controlling subsystem).

Look at the communication entries of the QCMN subsystem:

```
                        Display Communications Entries
                                                              System:   PID400B
Subsystem description:   QCMN           Status:   ACTIVE

                              Job                          Default         Max
Device        Mode        Description    Library        User          Active
*ALL          *ANY        *USRPRF                       *SYS          *NOMAX  1
*ALL          DMIRROR     DMCJOBD        D_MIRROR       D_MIRROR      *NOMAX
*ALL          QCASERVR    *USRPRF                       *NONE         *NOMAX
*ALL          QIJS        *USRPRF                       QIJS          *NOMAX
*ALL          QPCSUPP     *USRPRF                       *SYS          *NOMAX
*ALL          QSOCCT      *USRPRF                       QUSER         *NOMAX
*APPC         QADSM       *USRPRF                       QADSM         *NOMAX









                                                                        Bottom
Press Enter to continue.

F3=Exit   F12=Cancel
```

Figure 35. QCMN Subsystem Communications Entries

You can see at **1** that by default, ***SYS** user is used for DDM jobs. *SYS user for an AS/400 system is QUSER.

Normally, QUSER does not have enough authority to access any important files. Of course, you can authorize QUSER to all necessary objects, or change the communication entry for the subsystem to use a more powerful user profile. But then all remote requests are performed with the same level of authority and you lose the ability to control object access for a user by user basis.

You can use configuration lists to provide a better solution:

1. On every node in the network, create a special local location name to be used for the relational database and distributed file access. In order to do this, add an entry to the APPN local location list:

```
                          Change Configuration List                  SYSTEM01
                                                                  08/07/96  10:24:45
Configuration list  . . :   QAPPNLCL
Configuration list type :   *APPNLCL
Text  . . . . . . . . . :   Local Location List

Type changes, press Enter.

-----APPN Local Locations-----          -----APPN Local Locations-----
Local                                   Local
Location  Text                          Location  Text
M01       Secure Location name          ________  ____________________
 2        ____________________          ________  ____________________
________  ____________________          ________  ____________________
________  ____________________          ________  ____________________
________  ____________________          ________  ____________________
________  ____________________          ________  ____________________
________  ____________________          ________  ____________________
________  ____________________          ________  ____________________
________  ____________________          ________  ____________________
________  ____________________          ________  ____________________
                                                                  More...
F3=Exit   F12=Cancel   F17=Top   F18=Bottom
```

Figure 36. APPN Local Location List

You can choose any location name unique in your network. Of course, try to follow the naming conventions established for your network.

2. Now on every node in the network, add entries to the APPN remote location lists. These entries serve two purposes:
   - To relate local and remote location names.
   - To declare the remote location name as a secure location.

```
                          Change Configuration List                  SYSTEM01
                                                                  08/07/96  10:28:19
Configuration list  . . :   QAPPNRMT
Configuration list type :   *APPNRMT
Text  . . . . . . . . . :   Remote Location List

Type changes, press Enter.

----------------------------APPN Remote Locations-------------------------------
 5        Remote       6     Remote    Control    4                  3
Remote    Network    Local    Control   Point     Location           Secure
Location  ID         Location Point     Net ID    Password           Loc
M02       NET0002    M01      SYSTEM02  NET0002   1212121212121212   *YES
M03       NET0003    M01      SYSTEM03  NET0003   1313131313131313   *YES
________  *NETATR    *NETATR  ________  *NETATR   ________________   *NO
________  *NETATR    *NETATR  ________  *NETATR   ________________   *NO
________  *NETATR    *NETATR  ________  *NETATR   ________________   *NO
________  *NETATR    *NETATR  ________  *NETATR   ________________   *NO
________  *NETATR    *NETATR  ________  *NETATR   ________________   *NO
________  *NETATR    *NETATR  ________  *NETATR   ________________   *NO
                                                                  More...
F3=Exit   F11=Display session information   F12=Cancel   F17=Top   F18=Bottom
```

Figure 37. APPN Remote Location List

When you specify *YES in the ″Secure Loc″ column (3), the system now performs all of the communication requests coming from this remote location using the user profile of the requester.

You can configure for additional security if you specify a password in 4 . Now before a session is established using this pair of local and remote location names, the two systems exchange this password to authenticate its partner. Passwords are exchanged in encrypted form so your systems are better protected from unauthorized access.

Of course, this configuration should be performed symmetrically on all participating systems.

3. Create or modify your relational database entries to refer to the location entries previously configured:

```
                    Change RDB Directory Entry (CHGRDBDIRE)

Type choices, press Enter.

Relational database  . . . . . > SYSTEMO2      Character value
Remote location  . . . . . .   MO2      5     Name, *LOCAL, *ARDPGM, *SAME
Text . . . . . .   Relational database on SYSTEMO2
     ,

                           Additional Parameters

Device:
  APPC device description  . . .   *LOC          Name, *LOC, *SAME
Local location . . . . . .   MO3      6    Name, *LOC, *NETATR, *SAME
Remote network identifier  . . .   NETOOO2  5     Name, *SAME, *LOC, *NETATR...
Mode . . . . . .   *NETATR        Name, *NETATR, *SAME
Transaction program  . . . . . .   *DRDA         Character value, *DRDA, *SAME



                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 38. Changing Relational Database Directory Entry

The local location specified here (6 in Figure 38) should be the same as the local location in the remote location list (6 in Figure 37 on page 88) and remote location name and network identifier parameters here (5 in Figure 38) should be the same as in Figure 37 on page 88 (see 5).

Again, symmetrical configuration should be created on every node in a node group.

After this configuration is complete, all requests for distributed files coming from a remote node are executed under the user profile of the user who initiated them. Now you are able to control access to the distributed file on a user-by-user basis.

11.6 National Language Support

The same NLS considerations apply to a distributed file as for ordinary local files. All parts of the file have the same NLS-related attributes:

- Coded Character Set Identifier (CCSID)
- Sort sequence
- Language identifier

When one of these attributes is changed by the CHGPF command, new values are propagated to all of the nodes in a node group of the distributed file. CHGPF is an atomic operation, it is either performed on all of the nodes for the file or all of the parts of the file remain unchanged. In this way, the system guarantees that at any moment in time, all of the parts of the file have the same set of basic attributes.

When an application is working with a distributed file, all NLS characteristics of the file behave the same as for a local file. All necessary conversions are performed by the system depending on the CCSID attribute of the file object and CCSID value of the job accessing the file, regardless of which node the requested record was stored on. If either of the CCSIDs is *HEX (or 65535), the system performs no conversion and delivers any character data exactly as it is stored in the file. This creates no problems when all of the systems in the network use the same language (and CCSID) because no conversion is needed anyway. But when you are working in a multi-lingual environment, you should always explicitly specify the CCSID value both for your jobs and for the files. This gives the system the necessary information to be able to perform appropriate conversions automatically.

Let us consider the configuration in Figure 39 on page 91. A distributed file F was created with CCSID 285 (U.K. English) and was distributed over two systems. System A uses CCSID 37 (U.S. English) and system B is configured with CCSID 285. By default, all jobs in the system inherit the CCSID attribute from the system configuration (system value QCCSID).

If a user on system B inputs a dollar sign ($) in a character field of file F, the hex value for the dollar sign in code page 285 is 4A and this value is inserted in the file, since the CCSID of the job and the CCSID of the file are the same.

Now, a user on system A reads this value from the file F. On system A, jobs normally run with CCSID 37 which is different from 285. The hex value, 4A, is converted to the hex value 5B, which is the code point for the dollar sign in code page 37. As a result, system A correctly displays a dollar sign to the user.

If either the CCSID for the file was set to *HEX (65535) or the QCCSID system value of system B was *HEX, no conversion occurs and the user on system A sees the ″wrong″ character, in this case a cent sign. This situation is shown in Figure 39 on page 91.

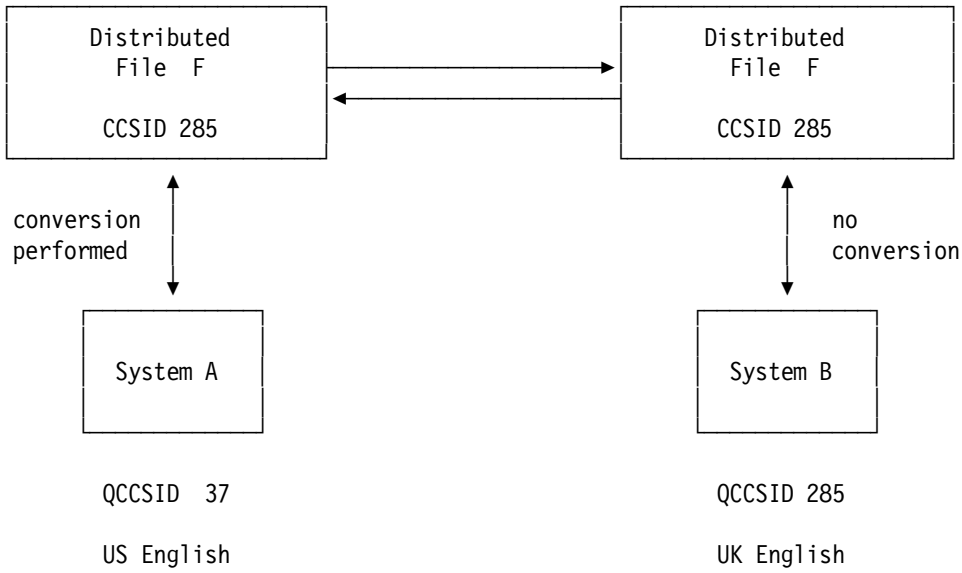


Figure 39. Distributed File on Systems with Different CCSIDs

When you distribute the file over nodes with different CCSIDs configured (QCCSID system value), the implications are the same as when you move an ordinary file from a system with one CCSID to a system with a different CCSID.

11.7 Managing Jobs

When you operate with distributed files, DB2 Multisystem may activate several jobs on the various systems to service your requests.

When a distributed file is opened, a DDM job is started on each node of the node group defined in the distributed file object. If the file is opened under commitment control, a second DDM job is started on each system. Updating distributed files under commitment control requires a distributed unit of work. This kind of processing requires a special type of APPC conversation with a remote node called protected conversation. These conversations require additional jobs to be started.

Once you open a second distributed file within the same job, the system can share the existing conversations with other nodes and, therefore, no new DDM jobs are started.

A DDM job started with a protected conversation can equally well service requests that are not issued under commitment control. It means that if you open the first distributed file in the job under commitment control, no other jobs are needed for the respective node. In this case, no matter how many distributed files are used, you have one single DDM job on every node where any part of any open distributed file resides.

When you are performing some other operation with a distributed file other than file I/O (for example, running the GRTOBJAUT command), still another DDM job is started on every participating node. System commands that are relevant to a

distributed file as a whole (with the exception of CHGPF command), require a separate conversation and, therefore, a separate DDM job on the remote systems.

The CHGPF command is always an atomic operation (it either changes all parts of the file or does not change anything). To ensure this consistency, a distributed unit of work and a protected conversation are required. The CHGPF command can reuse a protected DDM job if it was started by opening some distributed file under commitment control.

Depending on the sequence of actions in your job, you can have up to three DDM jobs started for your job on every node in a node group:

- One unprotected DDM job for file I/O not under commitment control
- One protected DDM job for file I/O and CHGPF processing
- One protected DDM job for other file management operations

You can see these jobs if you issue the WRKACTJOB command on the remote node:

```
                            Work with Active Jobs                       SYSTEM02
                                                              08/28/96  13:09:31
CPU %:   29.9     Elapsed time:   00:00:02     Active jobs:   83

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QBATCH         QSYS        SBS      .0                 DEQW
__   QCMN           QSYS        SBS      .0                 DEQW
1     MO1           A960202B    EVK      .0                 ICFW
2     MO1           A960202B    EVK      .0                 ICFW
3     MO1           A960202B    EVK     6.3                 ICFW
__     P23YDTYK     A960202B    EVK      .0  *  -PASSTHRU    EVTW
__   QCTL           QSYS        SBS      .0                 DEQW
__     QSYSSCD      QPGMR       BCH      .0  PGM-QEZSCNEP    EVTW
__   QINTER         QSYS        SBS      .0                 DEQW
                                                                      More...
Parameters or command
===>
F3=Exit      F5=Refresh   F10=Restart statistics   F11=Display elapsed data
F12=Cancel   F23=More options   F24=More keys
```

Figure 40. DDM Jobs Supporting Distributed Environment

Jobs, marked by **1**, **2**, and **3** in Figure 40 are all started to service one job on the originating system.

If you run a query over the distributed file, the same DDM job that was used for file I/O operations can be used to process the query on the corresponding node. However, it may happen that the various branches of a query need to access distributed files and then they start DDM jobs of their own. This can happen if the query references more that one file, for instance, or if the query optimizer needs a distributed temporary file. See Chapter 12, “Distributed Queries” on page 101 for more information on query implementation with distributed files.

Even when a query is run over one single distributed file, additional DDM jobs are started. Each remote DDM job needs to refer back to the originating system and to interact with the other nodes. For example, a file is distributed across systems SYSTEM01, SYSTEM02, and SYSTEM03. A query is initiated on SYSTEM01. Figure 41 shows the jobs active on SYSTEM01. In Figure 42 on page 94, you can see what happened during the execution on SYSTEM02. The situation SYSTEM03 is symmetrical to the one presented for SYSTEM02.

```
                            Work with Active Jobs                       SYSTEM01
                                                                 08/28/96  13:31:10
CPU %:     .0     Elapsed time:   00:00:00     Active jobs:   94

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QBATCH         QSYS        SBS      .0                 DEQW
__   QCMN           QSYS        SBS      .0                 DEQW
4     MO2           A960202B    EVK      .0                 ICFW
5     MO3           A960202B    EVK      .0                 ICFW
__     P23YDTYK     A960202B    EVK      .0  *  -PASSTHRU     EVTW
__     SYS19        A960218A    EVK      .0                 ICFW
__   QCTL           QSYS        SBS      .0                 DEQW
__     DSP01        QSECOFR     INT      .0  *  -CMDENT       DSPW
6     P23YDTYKL     A960202B    INT      .0  CMD-STRSQL       RUN
                                                                More...
Parameters or command
===>
F3=Exit      F5=Refresh   F10=Restart statistics   F11=Display elapsed data
F12=Cancel   F23=More options   F24=More keys
```

Figure 41. WRKACTJOB Display for Query Originating Node

In this figure, 6 is the job that initiates the query. 4 is a DDM job started from SYSTEM02 by the job marked with 7, as you can see in Figure 42 on page 94. Similarly, 5 is a DDM job initiated by SYSTEM03.

```
                              Work with Active Jobs                     SYSTEM03
                                                                 08/28/96  13:20:50
CPU %:    3.4     Elapsed time:   00:11:21     Active jobs:   84

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QBATCH         QSYS        SBS      .0                 DEQW
__   QCMN           QSYS        SBS      .0                 DEQW
__     M01          A960202B    EVK      .0                 ICFW
7    M01          A960202B    EVK      .8                 ICFW
__     M01          A960202B    EVK      .0                 ICFW
8    M03          A960202B    EVK      .0                 ICFW
__     P23YDTYK     A960202B    EVK      .0  *  -PASSTHRU     EVTW
__   QCTL           QSYS        SBS      .0                 DEQW
__     QSYSSCD      QPGMR       BCH      .0  PGM-QEZSCNEP     EVTW
                                                                    More...
Parameters or command
===>
F3=Exit      F5=Refresh   F10=Restart statistics   F11=Display elapsed data
F12=Cancel   F23=More options   F24=More keys
```

Figure 42. WRKACTJOB Display for Remote Node for Query

7 is the job that performs processing on SYSTEM02 for the query that runs on SYSTEM01. This job starts secondary DDM jobs on SYSTEM01 (originating node) and SYSTEM03. **8** is a secondary DDM job started on SYSTEM02 from SYSTEM03 by the SYSTEM03 counterpart of the job **7**.

If you run the CHGPF command for a distributed file, still more jobs are started. CHGPF processing requires exchanging information between all participating nodes. CHGPF is an atomic operation and is required to run under commitment control. Commitment control in a distributed environment means a distributed unit of work. To support a distributed unit of work, APPC protected conversations (and corresponding DDM jobs) are needed. If there are no active protected conversations to required nodes in a DDM job that processes the CHGPF command on some remote node (this job was mentioned on page 92), new conversations and, therefore, DDM jobs are started.

We now refer again to Figure 42. In the same environment, the CHGPF command is run in the same job on SYSTEM01. Let us see what has changed on SYSTEM02:

```
                              Work with Active Jobs                    SYSTEM02
                                                                 08/28/96  14:01:02
CPU %:    2.7     Elapsed time:   00:51:33     Active jobs:   85

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QBATCH         QSYS        SBS      .0                 DEQW
__   QCMN           QSYS        SBS      .0                 DEQW
__     MO1          A960202B    EVK      .0                 ICFW
9     MO1          A960202B    EVK      .9                 ICFW
__     MO1          A960202B    EVK      .1                 ICFW
__     MO3          A960202B    EVK      .0                 ICFW
10    MO3          A960202B    EVK      .0                 ICFW
__     P23YDTYK     A960202B    EVK      .0  *  -PASSTHRU     EVTW
__   QCTL           QSYS        SBS      .0                 DEQW
                                                                    More...
Parameters or command
===>
F3=Exit      F5=Refresh   F10=Restart statistics   F11=Display elapsed data
F12=Cancel   F23=More options   F24=More keys
```

Figure 43. DDM Jobs Started for CHGPF Command

In the previous figure, the job marked with **9** is responsible for running the CHGPF command at SYSTEM02. This job starts two additional DDM jobs with protected conversations (to SYSTEM01 and SYSTEM03 respectively). **10** is the same kind of job started from SYSTEM03 to SYSTEM02.

In our scenario, we activated all of the possible types of DDM jobs related to operating with distributed files. All future requests, no matter what distributed files are referenced, use one or more of these jobs (provided that the node groups involved contain the same nodes).

It is important you remember that these jobs all relate to the single originating job on SYSTEM01. Any request coming from a different job on SYSTEM01 start a new, totally independent network of DDM jobs.

Although there are so many DDM jobs around, you do not need to be concerned too much about managing all of these jobs. The system automatically takes care of starting and ending these jobs. You may need to know what jobs are currently active in the system and what purpose they serve.

More information about DDM jobs is found in *Distributed Database Programming*, SC41-3702.

11.7.1 Managing Temporary Result Writer Jobs

When query is run that uses the temporary result writer support (see Chapter 12, “Distributed Queries” on page 101), you may have to deal with a different type of job in your system. Temporary writer jobs are used to offload some parts of query processing from the application′s job. When query is processed using this support, the query processor sends requests to an internal temporary writer′s queue. Temporary writers pick up this request, perform the necessary processing, and report results back through another queue.

Temporary writers are paired jobs called QQQTEMP1 and QQQTEMP2. Being separate jobs, they work in parallel with the application's job where query is run.

There are two groups of asynchronous writer jobs in the system. Look at the WRKACTJOB display:

```
                          Work with Active Jobs                    SYSTEM01
                                                          08/28/96  17:25:32
CPU %:     .0     Elapsed time:   00:00:00     Active jobs:   147

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QDCPOBJ4       QSYS        SYS      .0                  EVTW
__   QDCPOBJ5       QSYS        SYS      .0                  EVTW
__   QFILESYS1      QSYS        SYS      .0                  TIMW
__   QJOBSCD        QSYS        SYS      .0                  EVTW
__   QLUR           QSYS        SYS      .0                  EVTW
__   QLUS           QSYS        SYS      .0                  EVTW
__   QPFRADJ        QSYS        SYS      .0                  EVTW
13   QQQTEMP1       QSYS        SYS      .0                  DEQW
__   QQQTEMP2       QSYS        SYS      .0                  DEQW
                                                                    More...
Parameters or command
===>
F3=Exit      F5=Refresh   F10=Restart statistics   F11=Display elapsed data
F12=Cancel   F23=More options   F24=More keys
```

Figure 44 (Part 1 of 2). Temporary Result Writer Jobs on WRKACTJOB Display

```
                          Work with Active Jobs                    SYSTEM01
                                                          08/28/96  17:25:32
CPU %:     .0     Elapsed time:   00:00:00     Active jobs:   147

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QSYSWRK        QSYS        SBS      .0                  DEQW
__     QNSCRMON     QSVSM       BCH      .0  PGM-QNSCRMON    DEQW
14     QQQTEMP1     QPGMR       BCH      .0  PGM-QQQTEMP1    DEQW
__     QQQTEMP1     QPGMR       BCH      .0  PGM-QQQTEMP1    DEQW
__     QQQTEMP2     QPGMR       BCH      .0  PGM-QQQTEMP2    DEQW
__     QQQTEMP2     QPGMR       BCH      .0  PGM-QQQTEMP2    DEQW
__     QTCPIP       QTCP        BCH      .0                  DEQW
__     QTFTP00379   QTCP        BCH      .0                  DEQW
__     QTFTP00427   QTCP        BCH      .0                  DEQW
                                                                    More...
Parameters or command
===>
F3=Exit      F5=Refresh   F10=Restart statistics   F11=Display elapsed data
F12=Cancel   F23=More options   F24=More keys
```

Figure 44 (Part 2 of 2). Temporary Result Writer Jobs on WRKACTJOB Display

At 13, there are two system jobs QQQTEMP1 and QQQTEMP2; and under QSYSWRK subsystem at 14, there are two more similar looking pairs of jobs.

Functionally, there is no difference between these groups of jobs. They serve the same purpose. The difference is in the way these jobs are started. System jobs are always present in the system. They are started at IPL time and stay in the system all of the time. When queries want to use temporary writer's support, requests are put in the queue. The presence of QQQTEMP1 and QQQTEMP2 system jobs is a guarantee that there are always some jobs in the system to process these requests.

When these system jobs are busy processing the request, new arriving requests have to wait in the queue until the processing of the current request is over. In such cases, additional writer jobs running in the QSYSWRK subsystem come into play. They can pick up requests from the queue and process them. As a result, more queries can exploit temporary writer's support simultaneously.

A user cannot control system temporary writer jobs, but writer jobs in QSYSWRK subsystem can be controlled. These jobs are started from a temporary writer's startup job that is run from the autostart job entry whenever the QSYSWRK system is started:

```
                        Display Autostart Job Entries
                                                              System:   SYSTEM01
Subsystem description:   QSYSWRK        Status:   ACTIVE

Job            Job Description     Library
QCQAJE           QCQAJE            QSVMSS
QDB2MULTI        QQQTEMPS          QSYS   15
QFSIOPJOB        QFSIOPWK          QSYS
QIJSSCD          QIJSSTRJD         QIJS
QNSAJE           QNSAJE            QSMU
QPM400           Q1PJOBD           QSYS
QSYSWRKJOB       QSYSWRK           QSYS
QZMFECOX         QZMFEJBD          QSYS









                                                                        Bottom
Press Enter to continue.

F3=Exit   F12=Cancel
```

Figure 45. Autostart Job Entries for QSYSWRK Subsystem

Look at the QQQTEMPS job description 15 (only part of it is shown):

```
                          Display Job Description
                                                              System:   SYSTEM01
Job description:   QQQTEMPS       Library:   QSYS

Message logging:
  Level  . . . . . . . . . . . . . . . . . . . . :   4
  Severity . . . . . . . . . . . . . . . . . . . :   30
  Text . . . . . . . . . . . . . . . . . . . . . :   *SECLVL
Log CL program commands  . . . . . . . . . . . . :   *NO
Accounting code  . . . . . . . . . . . . . . . . :   *USRPRF
Print text . . . . . . . . . . . . . . . . . . . :   *SYSVAL

Routing data . . . . . . . . . . . . . . . . . . :   RUNPTY20

Request data . . . . . . . . . . . . . . . . . . :   CALL QSYS/QQQTEMPS
                                                       16


Device recovery action . . . . . . . . . . . . . :   *SYSVAL
                                                                       More...
Press Enter to continue.

F3=Exit   F12=Cancel
```

Figure 46. QQQTEMPS Job Description

This job description executes the QQQTEMPS program in library QSYS **16**. This program, by default, starts two pairs of temporary writer jobs. You can pass a parameter to the QQQTEMPS program that causes it to start as many pairs of writers as specified by the parameter. For example, if you specify:

```
CALL QSYS/QQQTEMPS 5
```

in a QQQTEMPS job description, every time the QSYSWRK subsystem is started, five pairs of temporary writer jobs are submitted to the QSYSWRK subsystem.

You can also run the preceding command at any time you feel you need to increase the number of writer jobs and more writers are submitted to the QSYSWRK subsystem.

In this way, you can control how many temporary writer′s jobs are in your system at any given time. The more writers in the system, the more distributed queries can be run in parallel.

When temporary writer jobs are processing a distributed query, they start DDM jobs of their own to remote systems to access remote parts of distributed files residing on respective nodes. These DDM jobs are much the same as the ones described in a previous section.

The temporary result writer jobs are also documented in informational APAR II09526, which can be downloaded to your system using the SNDPTFORD command.

These DDM jobs are started under the user profile of their initiating writer job (for system writer jobs, it is QSYS, and for writer jobs in the QSYSWRK subsystem, it is QPGMR by default). However, the access to data is performed with the authority of the user who runs the query. This authority is, therefore, changed on a query-by-query basis.

11.8 Redistribution Considerations

When you want to change distribution attributes for a distributed file, use the CHGPF command. You can change all of the attributes by a single command:

- Add one or more new nodes to the node group of the file.
- Remove one or more nodes from the node group of the file.
- Change the partitioning key for the file.
- Change the partitioning map that performs mapping between partitions and nodes in the node group.

The CHGPF command can also make a local file become a distributed file or, vice versa, convert a distributed file to a local one. It is interesting to note that this is an unusual way to move a file from one AS/400 system to another without going through the save/restore procedure:

1. Create a node group object that refers to two AS/400 systems with default partitioning mappings.
2. Distribute a file over two nodes using the CHGPF command that uses any field of the file as a partitioning key and refer to the preceding node group object.
3. Go to the second AS/400 system (target system) and run the CHGPF command, which turns the distributed file just created into a normal local file.

This method does not involve using any external media (tapes) and may require considerably less intermediate DASD storage than saving to a save file or using ObjectConnect.

If a distributed file has logical files, SQL views, or SQL indexes built over it, the entire file network is automatically redistributed as a whole.

If a file was journaled when the redistribution process is run, it is journaled to the same journal after redistribution.

If any referential constraints are defined for a distributed file, constraints should be manually removed before redistribution is run and redefined after the process is complete. Files associated by a constraint should have identical distribution attributes. This is a restriction of the DB2 Multisystem for OS/400 feature. This restriction makes it possible to enforce any constraint locally on each node without going out to other nodes in a network. Since it is not possible to redistribute more than one file by a single command, you must take care of constraints manually.

Redistribution of the file (as well as changing a local file to distributed and a distributed file to local) is an atomic operation. It always performs all of the changes for all nodes for the file. If something goes wrong, all of the changes made are rolled back and the original environment is restored.

Redistribution can be started from any node where any part of the distributed file resides. If node group attributes are being changed, the modified node group object should be accessible on the node where the CHGPF command is run. All of the necessary information is extracted on the originating node and sent to all of the participating nodes for processing. A node group object is not needed on remote nodes and if present, is not used.

Redistribution is a symmetrical process and runs simultaneously on every node. The originating node performs its own processing and also coordinates the entire process. The originating node establishes a protected APPC conversation with each remote node configured in original file attributes and starts a DDM job. If a conversation is already available, it is re-used and no new jobs are started. (For more information about jobs involved in processing distributed files, see Section 11.7, “Managing Jobs” on page 91). Then the redistribution request is sent to every node for the file for processing.

On every node, the redistribution process goes through the following major steps:

- Renames all participating files (the distributed file itself and any logical files dependent on it) with some temporary names. All constraints are also redefined in such a way that this temporary network is the exact replica of the original database. This allows the system to roll back the operation, should a failure occur.
- New files, with the original names, are created.
- All the necessary communications links are established. Protected conversations are used. If no active conversation is found, a new one are started together with a DDM job. (For more information about jobs involved in processing distributed files, see Section 11.7, “Managing Jobs” on page 91).
- Every record in the temporary file is read and its partitioning key hashed. The system determines where the record belongs, according to the new distribution criteria.

  If the record should remain on the local node, it is written to the local portion of the file. Otherwise, it is sent to the appropriate node. All I/O operations at this stage are buffered, so records are read and written with the highest possible efficiency to reduce DASD accesses and communications traffic.
- Finally, when all of the records are transmitted and written and all of the nodes reported successful completion to the coordinating node, the changes are committed and any temporary objects are finally deleted.
- If any problem occurs, the situation is brought back to its original status.

Free DASD space is needed on every node for the redistribution process to run. Be sure you take space into account if you are redistributing large files. Estimate the amount of data that will be migrated to each system with the technique exposed in Section 10.1, “How To Choose A Partitioning Key” on page 61.

As records migrate from one node to another one, this process generates communications traffic. If many records are moved as a result of the redistribution process, communications can become a bottleneck. Before actually proceeding with redistribution, you can estimate how many records migrate. See Section 10.3, “Monitoring for Distribution” on page 70 for some tips on this topic.

Chapter 12. Distributed Queries

This chapter discusses how the AS/400 query optimizer handles queries that run on distributed files. We also present some suggestions for you to improve performance of your distributed queries.

Any query interface available on the AS/400 system (interactive or imbedded SQL, Open Query File, Query/400) can be used to query distributed files. If you want more information on these topics, you can find an excellent chapter on distributed queries in *DB2 Multisystem for OS/400*, SC41-3705.

12.1 Distributed Optimization

When distributed queries are processed, there are two levels at which decisions about the implementation are made, the distributed level and the local level.

1. At the distributed level, the query optimizer has to determine which files should be processed on which nodes and establish what the major steps are for the implementation. The distributed optimizer is a new part of the query optimizer that comes with the DB2 Multisystem feature. The distributed optimizer broadcasts the distributed query to all the participating nodes. Each node executes the query on its own local data and returns the results back to the initiating node. The results are then coordinated into a single result set.
2. When a query is received by each participating node, the optimizer creates an access plan based on the local configuration and data statistics.

The distributed optimizer controls the flow of execution of a query, synchronizes processing on different nodes, processes the intermediate results received from remote nodes, and returns the final result to the requesting application.

The distributed optimizer makes general decisions, based on the data characteristics at the local node. These decisions include:

- How many steps are required to execute the query?
- Which nodes will participate in query?
- Which intermediate temporary files need to be created?
- Which piece of query should run on each node?

The local optimizer on each node goes into further detail to determine for instance:

- Which access paths or indexes should be used?
- Can you use an access path or a temporary file and sort for ORDER BY processing?
- Which implementation method should be used (for example, whether to use an access path or a hash algorithm to implement GROUP BY processing)?
- Which local temporary files are needed?
- Can Symmetric Multiprocessing be used to improve performance?

As a result of the processing made by the local optimizer, an access plan is built on each node. The various access plans can vary on the different nodes for the same query.

Although the detailed access plan for distributed queries is always created at run time, even for static embedded SQL, the system can store a permanent access plan for these queries also. The distributed access plan only contains information about the first phase of query optimization. Details of the local implementation of query parts are not saved and this part of query optimization is always completely rebuilt at run time.

Both distributed and local files can be referenced in one query. Distributed files referenced in the same query can be distributed according to different partitioning criteria, such as using different node groups or with different partitioning keys.

If you want to verify what decisions were made by the query optimizer to implement your query, place your job in debug mode. The query optimizer places performance-related messages into the job log. You see messages produced both by the distributed optimizer and by the local optimizer on every participating node.

For example, we ran the following query on SYSTEM01:

```
SELECT PRICE
   FROM ORDERS
     WHERE ORDERSTATUS = 'F'
```

The ORDERS file was distributed across SYSTEM01, SYSTEM02, and SYSTEM03. We received the messages reported in Figure 47.

```
                          Display All Messages
                                                          System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008547

    Performing distributed query.                              1
    All access paths were considered for file ORDERS.          2
    Arrival sequence access was used for file ORDERS.          2
    Access path suggestion for file ORDERS.                    2
    Following messages created on target system SYSTEM02.
    All access paths were considered for file ORDERS.          3
    Arrival sequence access was used for file ORDERS.          3
    Following messages created on target system SYSTEM03.
    All access paths were considered for file ORDERS.          4
    Arrival sequence access was used for file ORDERS.          4
    ODP created.
    Blocking used for query.


                                                                        Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 47. Query Optimizer Messages for Distributed Query

Message **1** indicates that a distributed query is being run. Messages marked by **2** are local optimizer messages for the part of the query that is run on

SYSTEM01. 3 and 4 are messages produced by the local optimizer on SYSTEM02 and SYSTEM03 respectively.

You can use these messages to tune the performance of your queries in the same way as for purely local queries. See *DB2 for OS/400 SQL Programming*, SC41-4611, for a discussion on optimizing SQL queries.

12.1.1 Communications Considerations

When a distributed query is being run, the various communications links that are necessary for the query to run should be available. Every participating node must be reachable from the originating node.

When a query takes several steps to execute (see, for example, Section 12.5.3, “Re-Partitioned Join” on page 116), temporary distributed files can be created to hold intermediate results that are processed by later steps of the query. These temporary files are distributed files themselves. To create and access these files, additional communications links may be needed.

Let us consider the simple configuration in Figure 48 on page 104. This example shows the communications implications of running distributed queries. More details on how distributed queries are implemented are discussed later in this chapter.

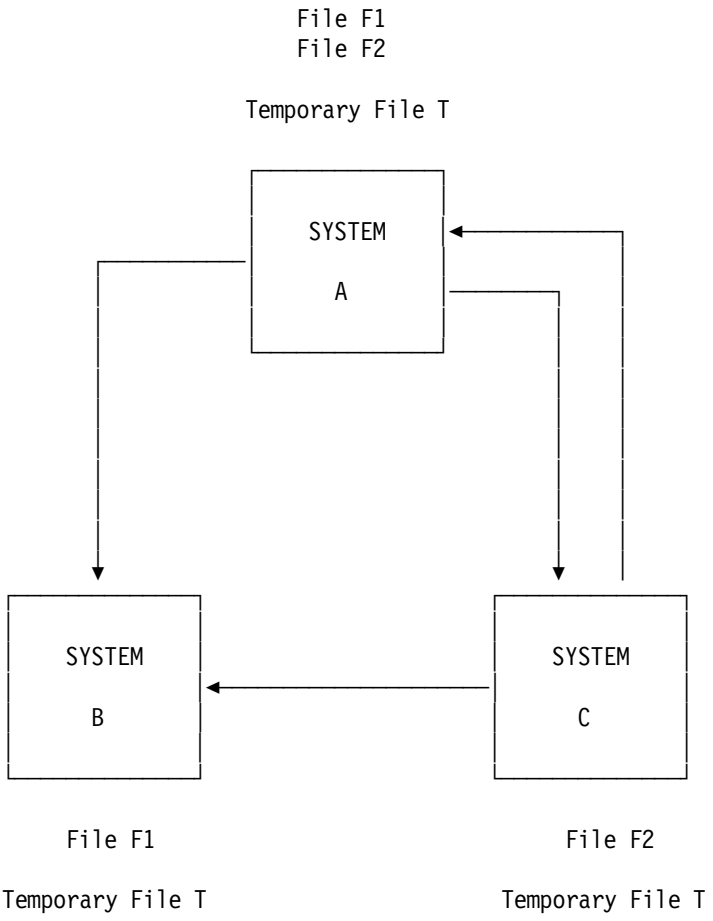


Figure 48. Distributed Query over Three-Node Network

In this figure:

- File F1 is distributed over System A and System B.
- File F2 is distributed over System A and System C.
- On System A, we run a query that joins F1 and F2.
- The optimizer splits the query processing into two steps:
  1. The system creates a temporary distributed file (file T) to hold the record selected from file F2, ignoring for the time being, the join criteria. File T spans the three systems.
  2. The distributed file T is now joined with file F1 on System A and System B.

Now we can count how many DDM jobs were activated to process this query (you may also want to look at Section 11.7, “Managing Jobs” on page 91):

- DDM jobs are started from System A to System B and System C to access File F1 and File F2 and coordinate query execution.
- DDM jobs are started from System C to System A and System B to distribute the temporary File T.

You can see that even for a simple query, you potentially need many communications links to be available. In a network of more than three nodes

where files are distributed according to different criteria, it may be difficult to predict which communications links should be available to implement the query. You also need to take into account that the query optimizer can vary its implementation decisions, depending on data statistics. Over time, the same query can be implemented in different ways.

Based on these considerations, the safest strategy for successfully running any type of distributed query is to ensure that all of the possible links among the various systems can be started (or a ″mesh network″). This also explains why it is best to configure communications in a symmetrical way on the various systems.

When you configure the Relational Database Directory, you need to choose an APPC mode for remote database access. By default, the mode configured in the relational database directory is taken from the system network attributes (**1** in Figure 49).

```
                    Change RDB Directory Entry (CHGRDBDIRE)

Type choices, press Enter.

Relational database  . . . . . > SYSTEMO2      Character value
Remote location  . . . . . .   MO2           Name, *LOCAL, *ARDPGM, *SAME
Text . . . . . .   Remote database on SYSTEMO2
     ’

                           Additional Parameters

Device:
  APPC device description  . . .   *LOC          Name, *LOC, *SAME
Local location . . . . . .   MO1           Name, *LOC, *NETATR, *SAME
Remote network identifier  . . .   NETOOO2       Name, *SAME, *LOC, *NETATR...
Mode . . . . . .   *NETATR  1   Name, *NETATR, *SAME
Transaction program  . . . . . .   *DRDA         Character value, *DRDA, *SAME



                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 49. Configuring APPC Mode for Distributed File Processing

In general, the *BLANK* mode is the default mode. Mode BLANK has an upper limit of eight conversations simultaneously active between any two remote locations in a network.

This limit might be easily overcome by any DB2 Multisystem environment. Consider using an APPC mode with a larger number of conversations (IBM-defined or created by you).

In some particular cases, the optimizer can understand by analyzing the query that only one specific node contains the data to be retrieved. In this case, only the communications link to the specified node needs to be active, or even no communications at all if the selected node is the local one. This category of queries is discussed in Section 12.2, “Single File Queries” on page 106.

12.2 Single File Queries

When a single file query is run, the query optimizer at the initiating node determines where the query has to be sent. The appropriate parts of the query are sent to each node for execution. For example, consider the following query:

```
SELECT PRICE, ORDERKEY
  FROM LINEITEM
    WHERE PRICE > 100000
```

File LINEITEM is distributed over SYSTEM01, SYSTEM02, and SYSTEM03 using partitioning key SHIPMODE. See Appendix A, “Database Files Used in Examples” on page 135 for more details on the structure of this file.

The previous query is sent to all of the nodes for execution. Each node returns records that satisfy the search criteria. A permanent index on the PRICE field can be used by each node, but the decision of using it or not is up to each individual node and is based on the local data statistics.

Running this query in debug mode, we obtained the following message in the job log:

```
                          Additional Message Information

Message ID . . . . . :   CPI4341       Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   09/05/96      Time sent  . . . . . :   14:47:14

Message . . . . :   Performing distributed query.
Cause . . . . . :   Query contains a distributed file. The query was processed
  in parallel on the following nodes: SYSTEM01, SYSTEM02, SYSTEM03.
Recovery  . . . :   For more information on processing of distributed files,
  refer to the DB2 Multisystem for OS/400 book, SC41-3705.









                                                                        Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 50. Determining Nodes for Distributed Query

If no special ordering is requested in a query, the order with which records are returned is not predictable.

Sometimes the query optimizer can determine that the data requested by the query resides on a subset of the nodes in the node group. In this case, the query is sent only to nodes that contain the requested records.

The optimizer can infer this information only if the query contains selection criteria that conform to the following set of rules:

- The selection criteria reference all the fields in the partitioning key.

- Only the equality operator (=) can be used for the fields in the partitioning key.
- The fields in the partitioning key must be compared to constants.

For example, if file LINEITEM has a partitioning key consisting of fields ORDERKEY, LINENUMBER, and PRICE, the following query is an example that fulfills the criteria above mentioned:

```
SELECT .....
  FROM LINEITEM
   WHERE ORDERKEY = 123456
     AND LINENUMBER = 2
       AND PRICE = 150000
```

When this specification is found, the system can calculate the hash value for the partitioning key and find the node where records should be searched. These selection criteria are called an **isolatable record selection**.

If you explicitly indicate the partitions or the nodes where you want the system to search the data (you can do it by using the functions NODENUMBER, NODENAME, or PARTITION) the system only directs the query to the specified nodes.

If your query does not contain an isolatable records selection, consider including additional (possibly redundant) predicates in your query. This does not change the meaning of the query but may improve its performance characteristics.

For example, your CUSTOMER file includes the STATE and the CITY fields and it is geographically distributed on the STATE field. Consider the following query:

```
SELECT ....
  FROM CUSTOMER
    WHERE CITY = 'CHICAGO'
```

The optimizer cannot determine where the requested records are located and the query is sent to all the nodes.

You can add an obvious and redundant search condition such as:

```
SELECT ...
   FROM CUSTOMER
     WHERE CITY = 'CHICAGO'
       AND STATE = 'IL'
```

The additional expression does not change the logical meaning of the query but allows the optimizer to locate the records. Only one node receives the query.

Besides the obvious performance advantages, this type of query can produce results even though part of the network or some of the nodes are not available. In particular, if the query is directed to the local node only, no communications links are needed for the execution.

Another case where the optimizer immediately determines that not all nodes need to be sent a distributed query is when you configured one on more visibility nodes (see Section 10.1.3, “Visibility Node” on page 64). In this scenario, the data reside on some of the nodes, called the data nodes. The remaining nodes, called visibility nodes, contain the file description and no records.

If you have a single data node, any query that is initiated on that system is not sent to any other node. No communication links are needed in this case. If you

initiate a query on a visibility node, the only necessary link goes from there to the data node.

Remember that these considerations **only apply to query processing and do not hold true for native I/O processing**. If you use native database I/O, the *open file* operation is sent to every node in the network and even if a single node is not available, operating with the distributed file is not allowed.

12.3 Record Ordering

When ordering is specified in a distributed query, the ordering criteria are sent to every node along with the query. Every node can perform ordering of its result set independently. Based on the type of the query, the optimizer can choose one of two possible implementations:

- Collect all of the returned records from all nodes in a temporary file and then sort this temporary file on the coordinator node. With this method, no records can be returned to the application before all of the records are received from all nodes.
- Allow every node to order its records and then perform a merge of the ordered records while they are being received from remote nodes. From a performance standpoint, this is the fastest implementation. Records can be returned to the application before all of the records are received from other nodes. Another advantage is that sorting the result subsets is performed in parallel on all of the participating nodes.

A merge implementation can occur if ordering is requested and no UNION or final grouping is required. Otherwise, records are accumulated in a temporary file and a final sort is performed on the coordinator node.

12.4 UNION and DISTINCT Clauses

If the distributed query performs a UNION of several SELECT statements, these SELECT statements are processed serially, one after another. Each SELECT statement is a distributed query in itself and can be processed in parallel on all of the participating nodes. Records resulting from each SELECT statement are brought back to the coordinator node to perform the union operation.

If ORDER BY is specified for a union query, the result set is always collected in a temporary file on the coordinator node, sorted, and then returned to the application.

If DISTINCT is specified for a distributed query, the result set is brought back to the coordinator node and placed in a temporary file. Once all of the records are received, the temporary file is sorted and records are returned to the application. During the sort, the duplicate records are eliminated.

For example, the following query is run on SYSTEM01:

```
SELECT DISTINCT SHIPMODE
FROM LINEITEM
```

The following job log fragment shows details of the query implementation:

```
                              Display All Messages
                                                                System:   SYSTEM01
 Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008642
3 > strsql
    CONNECT to relational database SYSTEM01 completed.
    Current connection is to relational database SYSTEM01.
    Performing distributed query.
    Arrival sequence access was used for file LINEITEM.
    Temporary result file built for query.                    1
    Following messages created on target system SYSTEM02.
    Arrival sequence access was used for file LINEITEM.
    Temporary result file built for query.
    Following messages created on target system SYSTEM03.
    Arrival sequence access was used for file LINEITEM.
    Temporary result file built for query.
    Temporary result file built for query.                    2
    ODP created.
    Blocking used for query.
                                                                          Bottom
 Press Enter to continue.

 F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 51. Query Optimizer Messages for Distributed Query with DISTINCT

At **1** in Figure 51, a temporary file is built to hold the results of local processing on node SYSTEM01. Similar messages were produced for two other nodes of this distributed file, SYSTEM02 and SYSTEM03. At **2**, a second temporary result file was built on SYSTEM01 (coordinator node) to implement the DISTINCT specification.

If an ORDER BY clause is specified together with DISTINCT, a distributed query can perform faster because the records returned by the individual nodes are already sorted and the process of eliminating the duplicates becomes easier. In addition, no temporary result file is needed for the DISTINCT clause and sorting is performed in parallel on all nodes. No final sorting is needed. If the requesting application runs interactively, the user does not have to wait until all the records have been processed to look at the first pages of results.

We added an ORDER BY clause to the previous query. The resulting query is as follows:

```
SELECT DISTINCT SHIPMODE
   FROM LINEITEM
     ORDER BY 1
```

In Figure 52 on page 110, you can look at the changes in the implementation.

```
                          Display All Messages
                                                              System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008642
    Performing distributed query.
    All access paths were considered for file LINEITEM.
    Arrival sequence access was used for file LINEITEM.
    Temporary result file built for query.                    3
    Following messages created on target system SYSTEM02.
    All access paths were considered for file LINEITEM.
    Arrival sequence access was used for file LINEITEM.
    Temporary result file built for query.                    3
    Following messages created on target system SYSTEM03.
    All access paths were considered for file LINEITEM.
    Arrival sequence access was used for file LINEITEM.
    Temporary result file built for query.                    3
    ODP created.
    Blocking used for query.

                                                                          Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 52. Query Optimizer Messages for Distributed Query with DISTINCT and ORDER BY

In our example, the local optimizer on all the nodes decided to order the results sorting a temporary file (**3**). In other cases, local optimizers can choose to use an access path to implement local ordering. Note that no final temporary result file was built on the coordinator node SYSTEM01 and that each node performed ordering independently.

It must be observed that adding the ORDER BY clause can improve performance if the various nodes run at a comparable speed. If the requester is much faster than other nodes, it can easily happen that collecting the records and sorting them centrally is much quicker than letting the slower systems sort their portion in parallel.

12.5 Distributed Join

DB2 Multisystem leaves users the most complete freedom about joining distributed files. Any type of combination is allowed (joining distributed files between them and joining distributed and local files, irrespectively of the way the files are partitioned). Nevertheless, there are some important potential implications on performance that depend on the structure of the join query.

In the best scenario, the various nodes need to compare only the local portions of the files referenced in the join query. No cross-system comparison is needed. This scenario corresponds to joining two files that are **partition-compatible** in relation to the join query. Two files are partition-compatible if all of the following statements are true:

- Both files are distributed over the same nodes in the same order. For example, if File F1 is distributed over Node A, Node B, and Node C, and File F2 is distributed over Node C, Node B, and Node A, they are not partition-compatible.

- Both files have the same partitioning map.
- Only the equality (=) operator is used for the join criteria.
- The two partitioning keys are part of the join criteria. In other words, all the fields of the partitioning key of the first file are joined to the corresponding partitioning key fields of the second file.
- The corresponding fields in the partitioning key must have a compatible data format (see *DB2 Multisystem for OS/400*, SC41-3705, for a complete description of this type of compatibility).

When files in a join query are not partition-compatible, the query optimizer has to implement a query in several steps.

Depending on the partitioning characteristics of files in a join query, we can identify four types of join queries:

- Co-located join
- Directed join
- Re-partitioned join
- Broadcast join

These are discussed in the following sections.

12.5.1 Co-located join

A co-located join is the simplest and the optimal way to implement a join query. All of the files in a join are partition-compatible and corresponding records are located on the same node. A co-located join is performed in one step and does not require the creation of temporary files. Only the result of the join flows across communication lines from remote nodes to the coordinator node.

The addition of more predicates to the query that do not contain fields from partitioning keys of either file does not affect the ability to perform a co-located join.

Let us look at an example. File LINEITEM and File ORDERS are distributed over System SYSTEM01, System SYSTEM02, and System SYSTEM03. The partitioning key is ORDERKEY for both files.

The following query is run on SYSTEM01:

```
SELECT L.ORDERKEY, O.CLERK
  FROM  LINEITEM L, ORDERS O
    WHERE L.ORDERKEY = O.ORDERKEY
      AND L.SHIPMODE = 'SHIP'
        AND O.ORDERSTATUS = 'F'
```

The addition of the two last predicates does not prevent the system from performing a co-located join.

Now let us look into the job log to find out how the query was implemented:

```
                          Display All Messages
                                                              System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008818

    Performing distributed join for query.                 1
    Access path built for file ORDERS.
    File LINEITEM processed in join position 1.
    File ORDERS processed in join position 2.
    Following messages created on target system SYSTEM02.
    Access path built for file ORDERS.
    File LINEITEM processed in join position 1.
    File ORDERS processed in join position 2.
    Following messages created on target system SYSTEM03.
    Access path built for file ORDERS.
    File LINEITEM processed in join position 1.
    File ORDERS processed in join position 2.
    ODP created.
    Blocking used for query.
                                                                          Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 53. Query Optimizer Messages for Query with a Co-Located Join

The following display shows the details for message **1**:

```
                        Additional Message Information

Message ID . . . . . :   CPI4342       Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/30/96      Time sent  . . . . . :   17:09:45

Message . . . . :   Performing distributed join for query.
Cause . . . . . :   Query contains join criteria over a distributed file and a
  distributed join was performed, in parallel, on the following nodes:
  SYSTEM01, SYSTEM02, SYSTEM03.
    The library, file and member names of each file involved in the join
  follow: A960202B/LINEITEM/LINEITEM, A960202B/ORDERS/ORDERS.
    A file name beginning with *QQTDF indicates it is a temporary distributed
  result file created by the query optimizer and it will not contain an
  associated library or member name.
Recovery  . . . :   For more information on processing of distributed files,
  refer to the DB2 Multisystem for OS/400 book, SC41-3705.

                                                                          Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 54. Query Optimizer Message with Details about Co-Located Join

12.5.2 Directed Join

In the directed join, files being joined are not partition-compatible but the fields of the partitioning key of one file are used in the join clause. This implementation, for example, takes place if the join fields from the second file do not match the partitioning key for this file or if the second file is not a distributed file. Since a co-located implementation is not possible, this query cannot be performed in one step and a temporary result file is needed. This file contains the data extracted from the second file, according to the predicates specified in the query and ignoring the join conditions.

This temporary file is a distributed file. It is distributed according to the node group and the partitioning map of the first file. Join fields from the file for which this temporary file is being built are used as the partitioning key for distribution. These temporary files are created in the QRECOVERY library and are completely managed by the system.

Once the temporary file has been created, it can be joined with the first file and this operation is a co-located join.

A directed join can only be used if join fields in a query have partition-compatible types and only the equal operator is used in a join (equijoin query).

In Figure 55, the query joins the file LINEITEM and the file ORDERS. File LINEITEM is distributed over SYSTEM01, SYSTEM02, and SYSTEM03 with ORDERKEY as a partitioning key. File ORDERS is distributed over the same nodes with CUSTOMER as a partitioning key. This is the same query as in Section 12.5.1, “Co-located join” on page 111, but the ORDERS file is distributed differently.

```
SELECT L.ORDERKEY, O.CLERK
   FROM  LINEITEM L, ORDERS O
     WHERE L.ORDERKEY = O.ORDERKEY
        AND L.SHIPMODE = 'SHIP'
          AND O.ORDERSTATUS = 'F'
```

Figure 55. A Directed Join Example

ORDERKEY is not the partitioning key for the ORDERS file, whereas it is the partitioning key for LINEITEM. A temporary file is created to distribute the ORDERS records using partitioning information extracted from LINEITEM and ORDERKEY as a partitioning key.

The following job log fragment shows performance messages issued by the query optimizer for this query.

```
                          Display All Messages
                                                              System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008863

    Optimizer debug messages for distributed query step 1 of 2 follow:  1
    Temporary distributed result file *QQTDF0001 built for query.       2
    Following messages created on target system SYSTEM03.
    Arrival sequence access was used for file ORDERS.
    Following messages created on target system SYSTEM13.
    Arrival sequence access was used for file ORDERS.
    Arrival sequence access was used for file ORDERS.
    Optimizer debug messages for distributed query step 2 of 2 follow:  1
    Performing distributed join for query.                              3
    Access path built for file LINEITEM.
    File 0108863001 processed in join position 1.                       4
    File ORDERS processed in join position 2.
    ODP created.
    Blocking used for query.
                                                                    Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 56. Query Optimizer Messages for Query with Directed Join

Messages at **1** show that the distributed query was processed in multiple steps (two steps in this case). The message at **3** indicates that a co-located join is being performed on the temporary file created in step **2** and file LINEITEM.

The message at **2** reports details for the temporary file being produced. The following display shows the detailed contents of this message:

```
                        Additional Message Information

Message ID . . . . . :   CPI4345       Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/30/96      Time sent  . . . . . :   17:19:28

Message . . . . :   Temporary distributed result file *QQTDF0004 built for
  query.
Cause . . . . . :   Temporary distributed result file *QQTDF0004 was created
  to contain the intermediate results of the query for reason code 1. The
  reason codes and their meanings follow:
    1 - Data from member ORDERS of file ORDERS in library A960202B was
  directed to other nodes.
    2 - Data from member ORDERS of file ORDERS in library A960202B was
  broadcast to all nodes.
    3 - Either the query contains grouping fields (GROUP BY) that do not match
  the partitioning keys of the distributed file or the query contains grouping
  criteria but no grouping fields were specified.
                                                                    More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 57 (Part 1 of 4). Temporary Distributed Result File for Directed Join

```
                        Additional Message Information

Message ID . . . . . :   CPI4345       Severity . . . . . :   00
Message type . . . . . :   Information

    4 - Query contains join criteria over a distributed file and the query was
  processed in multiple steps.
    A library and member name of *N indicates the data comes from a query
  temporary distributed file.
    File *QQTDF0004 was built on nodes: SYSTEM01, SYSTEM02, SYSTEM03.    5
    It was built using partitioning keys: ORDERKEY.                      6
    A partitioning key of *N indicates no partitioning keys were used when
  building the temporary distributed result file.
Recovery  . . . :   If the reason code is:
    1 - Generally, a file is directed when the join fields do not match the
  partitioning keys of the distributed file. When a file is directed, the
  query is processed in multiple steps and processed in parallel. A temporary
  distributed result file is required to contain the intermediate results for
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 57 (Part 2 of 4). Temporary Distributed Result File for Directed Join

```
                        Additional Message Information

Message ID . . . . . :   CPI4345       Severity . . . . . :   00
Message type . . . . . :   Information

  each step.
    2 - Generally, a file is broadcast when join fields do not match the
  partitioning keys of either file being joined or the join operator is not an
  equal operator. When a file is broadcast the query is processed in multiple
  steps and processed in parallel. A temporary distributed result file is
  required to contain the intermediate results for each step.
    3 - Better performance may be achieved if grouping fields are specified
  that match the partitioning keys.
    4 - Because the query is processed in multiple steps, a temporary
  distributed result file is required to contain the intermediate results for
  each step. See preceding message CPI4342 to determine which files were
  joined together.
    For more information on processing of distributed files, refer to the DB2
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 57 (Part 3 of 4). Temporary Distributed Result File for Directed Join

```
                        Additional Message Information

Message ID . . . . . :   CPI4345      Severity . . . . . :   00
Message type . . . . :   Information

  Multisystem for OS/400 book, SC41-3705.

                                                                      Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 57 (Part 4 of 4). Temporary Distributed Result File for Directed Join

Lines **5** and **6** in Figure 57 on page 114 indicate to which systems the temporary file was distributed and which partitioning key was used.

12.5.3 Re-Partitioned Join

In a re-partitioned join, files being joined in a query are not partition-compatible but this time, the files are joined over fields that do not match the partitioning key for either file. This query requires two temporary files for its execution:

- The first temporary file is built for the first file in the join query. This file contains all of the data selected from the first file, as specified in the selection criteria, ignoring the join predicates.
- A second temporary file is built for the second file in a join. Again, it contains data extracted from the second file, ignoring the join predicates.

These temporary files are distributed files. They are distributed using the node group and partitioning map of the second of the two files. The join fields from the second file are used as the partitioning key for distribution.

As for a directed join, temporary files are created in the QRECOVERY library and are completely managed by the system.

The two temporary files are partition-compatible and can be joined using a co-located join.

As in the case of the directed join, the re-partitioned join implementation can only be used if the join fields are of partition-compatible types and only the equal operator is used for the join predicates.

In the following query, file LINEITEM is distributed over SYSTEM01, SYSTEM02, and SYSTEM03 with SUPPLIER as a partitioning key. File ORDERS is distributed over the same nodes with CUSTOMER as a partitioning key. This is the same

query as in Section 12.5.1, “Co-located join” on page 111, but the files are distributed differently.

```
SELECT L.ORDERKEY, O.CLERK
   FROM  LINEITEM L, ORDERS O
      WHERE L.ORDERKEY = O.ORDERKEY
         AND L.SHIPMODE = 'SHIP'
           AND O.ORDERSTATUS = 'F'
```

The join field, ORDERKEY, does not belong to either of the partitioning keys. The join comparison is an equality and is performed over compatible fields. The re-partitioned algorithm can be utilized. These are the three steps the system takes to execute this query:

- The first temporary file is created, containing the records from LINEITEM that satisfy the condition SHIPMODE = 'SHIP'. These records are distributed according to the partitioning criteria used for ORDERS.
- The second temporary file is created. It contains records from ORDERS that satisfy the condition ORDERSTATUS = 'F'. These records are distributed according to the same criteria as for the ORDERS file itself.
- A co-located join is performed for the two temporary files.

The following job log fragment shows performance messages issued by the query optimizer for this query.

```
                          Display All Messages
                                                              System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008301

    Optimizer debug messages for distributed query step 1 of 3 follow:  1
    Temporary distributed result file *QQTDF0001 built for query.       2
    Following messages created on target system SYSTEM02.
    Arrival sequence access was used for file LINEITEM.
    Following messages created on target system SYSTEM03.
    Arrival sequence access was used for file LINEITEM.
    Arrival sequence access was used for file LINEITEM.
    Optimizer debug messages for distributed query step 2 of 3 follow:  1
    Temporary distributed result file *QQTDF0002 built for query.       3
    Optimizer debug messages for distributed query step 3 of 3 follow:  1
    Performing distributed join for query.                              4
    Hashing algorithm used to process join.
    Arrival sequence access was used for file 0108301001.
    Arrival sequence access was used for file 0108301002.
                                                                    More...
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 58. Query Optimizer Messages for Re-Partitioned Join

In Figure 58, messages at **1** show that the distributed query was processed in multiple steps (three steps in this case). Messages at **4** and below indicate that a co-located join is being performed of the temporary file created in step **2** and another temporary file created in step **3**

The message at **2** reports details for the creation of the first temporary file. The following display shows the detailed contents of this message:

```
                        Additional Message Information

Message ID . . . . . :   CPI4345
Date sent  . . . . . :   09/02/96      Time sent  . . . . . :   12:10:15

Message . . . . :   Temporary distributed result file *QQTDF0001 built for
  query.

Cause . . . . . :   Temporary distributed result file *QQTDF0001 was created
  to contain the intermediate results of the query for reason code 1. The
  reason codes and their meanings follow:
    1 - Data from member LINEITEM of file LINEITEM in library A960202B was
  directed to other nodes.
    2 - Data from member LINEITEM of file LINEITEM in library A960202B was
  broadcast to all nodes.
    3 - Either the query contains grouping fields (GROUP BY) that do not match
  the partitioning keys of the distributed file or the query contains grouping
  criteria but no grouping fields were specified.
                                                                        More...
Press Enter to continue.

F1=Help   F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 59 (Part 1 of 2). Temporary Distributed Result File for First File in a Re-Partitioned Join

```
                        Additional Message Information

Message ID . . . . . :   CPI4345
Date sent  . . . . . :   09/02/96      Time sent  . . . . . :   12:10:15

    4 - Query contains join criteria over a distributed file and the query was
  processed in multiple steps.
    A library and member name of *N indicates the data comes from a query
  temporary distributed file.
    File *QQTDF0001 was built on nodes: SYSTEM01, SYSTEM02, SYSTEM03.   5
    It was built using partitioning keys: ORDERKEY.                     6
    A partitioning key of *N indicates no partitioning keys were used when
  building the temporary distributed result file.
Recovery  . . . :   If the reason code is:
    1 - Generally, a file is directed when the join fields do not match the
  partitioning keys of the distributed file. When a file is directed, the
  query is processed in multiple steps and processed in parallel. A temporary
  distributed result file is required to contain the intermediate results for
                                                                        More...
Press Enter to continue.

F1=Help   F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 59 (Part 2 of 2). Temporary Distributed Result File for First File in a Re-Partitioned Join

Lines **5** and **6** in Figure 59 indicate to which systems the temporary file was distributed and which partitioning key was used.

Message **3** in Figure 58 on page 117 shows information about the second temporary file being created. The following display shows the details for this message:

```
                        Additional Message Information

Message ID . . . . . :   CPI4345
Date sent  . . . . . :   09/02/96      Time sent  . . . . . :   12:24:18

Message . . . . :   Temporary distributed result file *QQTDF0002 built for
  query.

Cause . . . . . :   Temporary distributed result file *QQTDF0002 was created
  to contain the intermediate results of the query for reason code 1. The
  reason codes and their meanings follow:
    1 - Data from member ORDERS of file ORDERS in library A960202B was
  directed to other nodes.
    2 - Data from member ORDERS of file ORDERS in library A960202B was
  broadcast to all nodes.
    3 - Either the query contains grouping fields (GROUP BY) that do not match
  the partitioning keys of the distributed file or the query contains grouping
  criteria but no grouping fields were specified.
                                                                        More...
Press Enter to continue.

F1=Help   F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 60 (Part 1 of 2). Temporary Distributed Result File for Second File in a Re-Partitioned Join

```
                        Additional Message Information

Message ID . . . . . :   CPI4345
Date sent  . . . . . :   09/02/96      Time sent  . . . . . :   12:24:18

    4 - Query contains join criteria over a distributed file and the query was
  processed in multiple steps.
    A library and member name of *N indicates the data comes from a query
  temporary distributed file.
    File *QQTDF0002 was built on nodes: SYSTEM01, SYSTEM02, SYSTEM03.    5
    It was built using partitioning keys: ORDERKEY.                      6
    A partitioning key of *N indicates no partitioning keys were used when
  building the temporary distributed result file.
Recovery  . . . :   If the reason code is:
    1 - Generally, a file is directed when the join fields do not match the
  partitioning keys of the distributed file. When a file is directed, the
  query is processed in multiple steps and processed in parallel. A temporary
  distributed result file is required to contain the intermediate results for
                                                                        More...
Press Enter to continue.

F1=Help   F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 60 (Part 2 of 2). Temporary Distributed Result File for Second File in a Re-Partitioned Join

Again, messages 5 and 6 show that this temporary file was distributed on SYSTEM01, SYSTEM02, and SYSTEM03 and ORDERKEY was used as a partitioning key.

If you compare Figure 57 on page 114 with Figure 59 on page 118 and Figure 60 on page 119, you can see that the re-partitioned join is similar to the directed join. The difference is that when the directed join is implemented, only one file is re-partitioned, whereas for the re-partitioned join, both files need to be re-partitioned.

12.5.4 Broadcast Join

When none of the join implementations previously discussed can be used, the system performs a broadcast join. In a broadcast join, all the necessary records from one of the two files are sent to all the nodes across which the second file is distributed. This type of processing can be lengthy and may take a lot of network, CPU, and disk space resources. A broadcast join has to be performed, for instance, if you do not specify an equal condition for the join criteria.

A temporary file is created for the broadcast file. This temporary file contains all of the data selected from the broadcast file, according to the selection criteria of the join query and ignoring the join predicates. This temporary file is a distributed file. It is distributed to all nodes for the remaining file in a join. But this is a special kind of a distributed file; it has no partitioning key. Every node contains all of the necessary data. These files can be created only by the system when a broadcast join is performed.

Broadcast join is used when no transformations discussed in the two previous sections can produce partition-compatible files:

- When join fields have a data type of date, timestamp, or floating-point numeric that cannot be used in a partitioning key.
- When non-equal comparison is used for join fields (non-equijoin queries).

In some cases, the optimizer can choose to perform a broadcast join instead of a re-partitioned join. Performing a re-partitioned join requires establishing additional connections between systems, starting more DDM jobs, and moving many records between the nodes in a network. On the other hand, a temporary file creation can be done in parallel on all of the nodes for the file. Broadcast join requires sending all of the records for the file to all of the participating nodes, but this must be done only for one file in a join and broadcast join may require fewer active conversations among the nodes.

The distributed query optimizer weighs the relative cost of performing a re-partitioned or a broadcast join, and can choose to use one or the other, depending on the current configuration.

In the following example, a broadcast join was used. For this query, LINEITEM is distributed over SYSTEM01, SYSTEM02, and SYSTEM03 with ORDERKEY as a partitioning key. ORDERS is distributed over SYSTEM01, SYSTEM02, and SYSTEM03 with ORDERKEY as a partitioning key. Non-equal comparison is used for join fields, so a broadcast join has to be performed (this query makes no logical sense and is used only to illustrate a broadcast join):

```
SELECT L.ORDERKEY, O.CLERK
   FROM  LINEITEM L, ORDERS O
      WHERE L.ORDERKEY <> O.ORDERKEY
         AND L.SHIPMODE = 'SHIP'
            AND O.ORDERSTATUS = 'F'
```

For this, the query optimizer decided to broadcast ORDERS because applying the selection ORDERSTATUS = 'F' results in broadcasting a smaller set of records.

Let us look at the optimizer messages:

```
                              Display All Messages
                                                                 System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008191

    Optimizer debug messages for distributed query step 1 of 2 follow:  1
    Temporary distributed result file *QQTDF0001 built for query.       2
    Arrival sequence access was used for file ORDERS.
    Optimizer debug messages for distributed query step 2 of 2 follow:  1
    Performing distributed join for query.                              3
    Access path built for file LINEITEM.
    File 0108218001 processed in join position 1.
    File LINEITEM processed in join position 2.
    ODP created.
    Blocking used for query.




                                                                    Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 61. Query Optimizer Messages for Broadcast Join

Message **1** shows that the query was performed in two steps:

- In the first step, records for ORDERS file were broadcast to all of the nodes for file LINEITEM. For this purpose, a temporary file was created **2**.
- Finally, a co-located join was performed for this temporary file and file LINEITEM (**3**).

The following display shows the details for message **2**:

```
                        Additional Message Information

Message ID . . . . . :   CPI4345      Severity . . . . . :   00
Message type . . . . . :   Information
Date sent  . . . . . :   08/30/96     Time sent  . . . . . :   17:45:02

Message . . . . :   Temporary distributed result file *QQTDF0001 built for
  query.
Cause . . . . . :   Temporary distributed result file *QQTDF0001 was created
  to contain the intermediate results of the query for reason code 2. The
  reason codes and their meanings follow:
    1 - Data from member ORDERS of file ORDERS in library A960202B was
  directed to other nodes.
    2 - Data from member ORDERS of file ORDERS in library A960202B was
  broadcast to all nodes.
    3 - Either the query contains grouping fields (GROUP BY) that do not match
  the partitioning keys of the distributed file or the query contains grouping
  criteria but no grouping fields were specified.
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 62 (Part 1 of 2). Temporary Distributed Result File for Broadcast Join

```
                        Additional Message Information

Message ID . . . . . :   CPI4345      Severity . . . . . :   00
Message type . . . . . :   Information

    4 - Query contains join criteria over a distributed file and the query was
  processed in multiple steps.
    A library and member name of *N indicates the data comes from a query
  temporary distributed file.
    File *QQTDF0001 was built on nodes: SYSTEM01, SYSTEM02, SYSTEM03.   4
    It was built using partitioning keys: *N.                           5
    A partitioning key of *N indicates no partitioning keys were used when
  building the temporary distributed result file.
Recovery  . . . :   If the reason code is:
    1 - Generally, a file is directed when the join fields do not match the
  partitioning keys of the distributed file. When a file is directed, the
  query is processed in multiple steps and processed in parallel. A temporary
  distributed result file is required to contain the intermediate results for
                                                                        More...
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 62 (Part 2 of 2). Temporary Distributed Result File for Broadcast Join

Line **3** in Figure 62 shows the nodes where the records from ORDERS were broadcast. *N in line **4** means that no partitioning key was used for the distributed temporary file.

12.6 Implementation of Grouping

Depending on whether or not you use the partitioning key in the grouping criteria, the optimizer can choose between two different types of implementations. Grouping is implemented either in one step or in two steps using an intermediate temporary file.

12.6.1 One-Step Grouping

If all fields from the partitioning key are GROUP BY fields, then grouping can be performed in one step because all of the data for every group reside on the same node. The grouping is performed in parallel on all of the nodes for the distributed file. The coordinator group collects all of the resulting records from all of the nodes and presents them to the application.

To implement one-step grouping, all of the fields of a partitioning key must be in a GROUP BY specification. Additional fields can appear as well without preventing the optimizer from choosing one-step grouping.

If an ordering specification is added to the GROUP BY query, the ordering request is sent to all of the nodes and is performed on all nodes in parallel. The coordinator node merges the ordered resulting records and returns them to the application. Grouping is still performed in one step.

For example, file LINEITEM is distributed over SYSTEM01, SYSTEM02, and SYSTEM03 with SHIPMODE as a partitioning key. The following query is implemented in one step:

```
SELECT SHIPMODE, SUM(PRICE)
    FROM  LINEITEM
       GROUP BY SHIPMODE
          ORDER BY 2
```

If you run this query in debug mode, you find the following messages in the job log:

```
                          Display All Messages
                                                          System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008967

    Performing distributed query.
    Access path built for file LINEITEM.                    1
    Temporary result file built for query.                  2
    Following messages created on target system SYSTEM02.
    Access path built for file LINEITEM.
    Temporary result file built for query.
    Following messages created on target system SYSTEM03.
    Access path built for file LINEITEM.
    Temporary result file built for query.
    ODP created.
    Blocking used for query.




                                                                      Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 63. Query Optimizer Messages for One-Step Grouping

For this query, the optimizer on every node created an access path to perform grouping (**1**). Temporary files were built on every node to perform ordering because grouping and ordering fields were not the same (**2**).

12.6.2 Two-Step Grouping

If the partitioning key is not referenced in the grouping clause, the query requires more than a single step.

These requests are carried out in two separate steps:

1. First, a temporary distributed file is created. This file is distributed over all the participating nodes but the coordinator node holds all of the data; the coordinator node is the data node and the remaining nodes are visibility nodes for the file.

   The query is then run in parallel on all systems. Grouping is performed independently on all of the nodes and the resulting records are placed in the temporary distributed file.

2. The query is then modified and run again on the coordinator node over the temporary distributed file created in the previous step.

If the grouping query contains a HAVING clause or group selection expression on the OPNQRYF command (GRPSLT parameter), the group selection is performed in step 2. The first step always returns all of the groups to the coordinator node.

If final ordering is requested in a query, it is performed as part of step 2 in the coordinator node.

Queries that summarize across the entire file with no group by specifications always require a two-step implementation. The following query is an example of what we call a *whole-file* grouping:

```
SELECT SUM(TOTALPRICE)
   FROM ORDERS
```

In the following query, LINEITEM file is distributed over SYSTEM01, SYSTEM02, and SYSTEM02 with ORDERKEY as partitioning key:

```
SELECT SHIPMODE, SUM(PRICE)
FROM  LINEITEM
GROUP BY SHIPMODE
ORDER BY 2
```

SHIPMODE is not a partitioning key for the file so this query cannot be run in one step.

The optimizer produces the following messages when this query is run:

```
                          Display All Messages
                                                          System:   SYSTEM01
Job . . :   P23YDTYKL     User . . :   A960202B      Number . . . :   008980

    Optimizer debug messages for distributed query step 1 of 2 follow:
    Temporary distributed result file *QQTDF0001 built for query.        1
    Following messages created on target system SYSTEM02.
    Access path built for file LINEITEM.
    Following messages created on target system SYSTEM03.
    Access path built for file LINEITEM.
    Access path built for file LINEITEM.                                 2
    Optimizer debug messages for distributed query step 2 of 2 follow:
    Access path built for file 0108980001.                               3
    Temporary result file built for query.                               4
    ODP created.
    Blocking used for query.



                                                                      Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 64. Query Optimizer Messages for Two-Step Grouping

Message **1** shows that a temporary distributed file was created in step 1 to implement grouping. Message **2** is a local optimizer message on the coordinator node (SYSTEM01 in this example). Similar messages produced on remote nodes can be seen just above this message. An access path was built on the coordinator node to implement grouping in step 2 of the query (**3**). A temporary file (non-distributed) has to be created in step 2 (**4**) to perform ordering of the results because the ordering field is different from the grouping field.

12.6.3 Grouping and Joins

If a query contains a grouping and a join clause, the optimizer makes a decision based on the join specifications. If one or more files and their partitioning keys influence the optimizer, it makes a decision for the *group by* implementation.

Let us look at the modified example from Section 12.5.3, “Re-Partitioned Join” on page 116. LINEITEM has a partitioning key SUPPLIER and ORDERS has a partitioning key CUSTOMER.

```
SELECT L.ORDERKEY, MAX(PRIORITY)
FROM  LINEITEM L, ORDERS O
WHERE L.ORDERKEY = O.ORDERKEY
   AND L.SHIPMODE = 'SHIP'
   AND O.ORDERSTATUS = 'F'
GROUP BY L.ORDERKEY
```

To perform a join, both files were re-partitioned using ORDERKEY as a partitioning key. Now grouping can be performed in one step. Overall, this query requires three steps:

1. Re-partition file LINEITEM.
2. Re-partition file ORDERS.
3. Perform a join of the resulting temporary distributed files. Grouping is performed in the same step.

In the following example, the situation is opposite. LINEITEM has the partitioning key SUPPLIER and ORDERS has the partitioning key CUSTOMER.

```
SELECT L.SUPPLIER, MAX(PRIORITY)
FROM  LINEITEM L, ORDERS O
WHERE L.ORDERKEY = O.ORDERKEY
   AND L.SHIPMODE = 'SHIP'
   AND O.ORDERSTATUS = 'F'
GROUP BY L.SUPPLIER
```

To perform a join, both files were re-partitioned using ORDERKEY as a partitioning key. Now grouping cannot be performed in one step. So an additional step is required to perform grouping. The implementation of this query requires four steps:

1. Re-partition file LINEITEM.
2. Re-partition file ORDERS.
3. Perform a join of the resulting temporary distributed files. The first phase of grouping is performed in the same step, producing another temporary distributed file.
4. Perform the final grouping on the coordinator node.

12.7 Improving Performance of Distributed Queries

The previous sections often mentioned that the execution of distributed queries may frequently lead to moving data across the network. Since communications links generally represent a bottleneck, your queries achieve optimal performance if they are formulated in a way that minimizes data exchange.

For this reason, you should analyze the queries that are run most often and in particular those queries for which response time is critical. You can also operate on the way your data are distributed to minimize data interchange.

Here are some practical recommendations:

1. If your queries refer to data that is located on some specific nodes, you can add selection criteria that help the optimizer in identifying those nodes (see Section 12.2, “Single File Queries” on page 106).
2. Adding an ORDER BY clause to a query that contains a DISTINCT clause generally results in better performance because ordering can be done in parallel on all of the nodes.
3. Queries containing grouping criteria perform best if you request grouping on all of the fields of the partitioning key. These queries can be implemented in a single step and parallelism is exploited.
4. In join queries, specify the entire partitioning key of both files in the join criteria, whenever this is possible. These queries are generally implemented as co-located joins. A co-located join is performed on all of the nodes in parallel and does not require the creation of intermediate temporary files.
5. Avoid join queries with non-equal comparison in the join criteria (non-equijoin queries) over distributed files. These queries always result in a broadcast join being performed, and the whole of one of the files in a query to be sent across communications links. See Section 12.5.4, “Broadcast Join” on page 120 for an example.
6. Use the ALWCPYDTA(*OPTIMIZE) parameter in the STRSQL, OPNQRYF, and SQL precompiler commands whenever possible. By doing so, you allow the query optimizer to consider sorting the result set instead of using an access path to resolve the query. This is a general performance tip that applies to all types of queries.
7. Follow the usual performance guidelines that are stated in *DB2 for OS/400 SQL Programming*, SC41-4611.

12.8 Parallel Support for Distributed Queries

If you look at the overall implementation of distributed queries, you may notice that parallelism can be utilized at several levels:

- If a distributed query is executed in several independent steps, these steps can run in parallel. For example, when a re-partitioned join is performed, the two files can be re-partitioned in parallel (see Section 12.5.3, “Re-Partitioned Join” on page 116).
- In general, distributed queries are broadcast to all the nodes in the network. Each node can perform its own amount of processing independently from the remaining nodes and, therefore, the various nodes can work in parallel.

DB2 for OS/400 addresses all of these areas. In this section, we see which parallel features can be employed to further improve the performance of your queries.

12.8.1 Temporary Result Writers

Temporary result writers (temp writers) are special jobs that can perform some pieces of distributed queries in parallel. On a system where the DB2 Multisystem feature is installed, temp writers are represented by a pair of system jobs, named QQQTEMP1 and QQQTEMP2. See Section 11.7, “Managing Jobs” on page 91 for more information about how the temp writers are handled by the system.

The idea behind temp writers is to let these jobs fill some temporary files with intermediate results. The benefit they provide resides in the fact that many temp writers can work in parallel to create temporary independent result sets that are combined later. The result is an improvement in the overall query run time.

The temporary files created by temp writers reside in the QRECOVERY library and are transparently managed by the system. A unique name is generated for each new temporary file. When the temp writer completes processing a request, it notifies the requesting job, which in turn can use the temporary result file for continuing the query execution.

Let us consider again the example in Section 12.5.3, “Re-Partitioned Join” on page 116.

In this example, the LINEITEM file is distributed using the SUPPLIER field as partitioning key. The ORDERS file is distributed over the same nodes with CUSTOMER as partitioning key.

```
SELECT L.ORDERKEY, O.CLERK
    FROM  LINEITEM L, ORDERS O
       WHERE L.ORDERKEY = O.ORDERKEY
          AND L.SHIPMODE = 'SHIP'
             AND O.ORDERSTATUS = 'F'
```

Note: The join fields are not partitioning keys for either file; therefore, the optimizer chooses to perform a re-partitioned query.

The optimizer takes the following actions:

1. Re-partition records of file LINEITEM using ORDERKEY as partitioning key.
2. Re-partition records of file ORDERS using ORDERKEY as partitioning key.
3. Perform a co-located join of the temporary files produced in the first two steps.

The first two steps do not depend on each other and can be run in parallel.

If you allow the optimizer to use temp writer support, as described in Section 12.8.4, “CHGQRYA Command” on page 132, the following steps are taken:

1. A temporary result file is created to perform the first step (re-partitioning the LINEITEM file).
2. The following query request is placed in the temp writer's queue:

   ```
   INSERT INTO  <TEMPFILE1>
      SELECT ORDERKEY
         FROM LINEITEM
           WHERE SHIPMODE = 'SHIP'
   ```

   We indicated with <TEMPFILE1> the name the system generates for the temporary result file.
3. Similarly, a temporary file is created for the second step.
4. The following query request is placed in the temporary writer's queue:

   ```
   INSERT INTO  <TEMPFILE2>
     SELECT ORDERKEY, CLERK
     FROM ORDERS
     WHERE ORDERSTATUS = 'F'
   ```
5. Once the two requests have been fulfilled, the query optimizer schedules the execution of the following query:

```
SELECT T1.ORDERKEY, T2.CLERK
  FROM <TEMPFILE1> T1, <TEMPFILE2> T2
    WHERE T1.ORDERKEY = T2.ORDERKEY.
```

12.8.2 Parallel Interaction with Remote Nodes

By default, a distributed query is sent to the various nodes of a node group in a serial way. In other words, the job that initiates the query waits until a node has completed its execution before sending the query request to the following nodes. The various nodes are contacted in the same sequence as they are specified in the node group object.

You can enable the query optimizer to send the query requests to every node up front and let the various nodes work in parallel by using the CHGQRYA command and its special *ASYNCJ* parameter, as described in Section 12.8.4, “CHGQRYA Command” on page 132. When the optimizer is so instructed, all the nodes in the node group can work at the same time to process the query. Parallelism can save a lot of time for most complex queries.

For example, if an access path has to be built on each node to process the query, by default, one node at a time creates the temporary access path. When you enable parallelism, the query optimizer initiates the access path creation on all the nodes at the same time. This can greatly reduce the run time for the query.

There are other cases where the benefit may not be so great and this is the reason why this support is not enabled by default. Parallelism can really help queries that process a lot of records in a resource intensive way on all the nodes and that return a relatively small number of records. On the other hand, if you run a plain query from an interactive session (such as, *SELECT * FROM LINEITEM*), you do not really benefit from parallel support. In this simple case, you are interested in receiving the first page of records as quickly as possible and the default serial processing is satisfactory. If you request a large summarization (such as *SELECT SUM(PRICE) FROM LINEITEM*) that is when parallelism can be greatly useful.

12.8.3 Symmetric Multiprocessing

Every node processes a distributed query as if it was a normal local query as far as optimization is concerned.

When the Symmetric Multiprocessing feature is installed on one or more nodes, distributed queries can also take advantage of the parallel support discussed in the first part of this book.

The Symmetric Multiprocessing feature can be controlled the same way as for local queries, by using the QQRYDEGREE system value or the DEGREE parameter of the CHGQRYA command. When the CHGQRYA command is issued for the job where a query is initiated (coordinator job), Symmetric Multiprocessing algorithms are normally enabled for the local node only. However, you can propagate the enablement to the remote nodes; you need to change the QQRYDEGREE system value on the remote nodes **before** DDM jobs are started. Changes to this system value do not have any effect on already active jobs.

For example, file LINEITEM is distributed over three systems (SYSTEM01, SYSTEM02, and SYSTEM03). Before a query is run, the following command is issued:

```
CHGQRYA  DEGREE(*MAX)
```

On remote nodes, the *MAX value is assigned to the QQRYDEGREE system value.

The following query is run on SYSTEM02:

```
SELECT  SUM(PRICE)
FROM LINEITEM
```

This kind of query can benefit from a parallel table scan (see Chapter 3, “Parallel Table Scan” on page 21).

The following messages are sent to the job log on the coordinator node:

```
                          Display All Messages
                                                          System:   SYSTEM02
Job . . :   P23YDTYKE     User . . :   A960202B      Number . . . :   003288

    DDM job started.
    DDM job started.
    Optimizer debug messages for distributed query step 1 of 2 follow:
    Temporary distributed result file *QQTDF0002 built for query.
    Following messages created on target system SYSTEM01.
    Arrival sequence access was used for file LINEITEM.
    Following messages created on target system SYSTEM03.
    Arrival sequence access was used for file LINEITEM.
    Arrival sequence access was used for file LINEITEM.
    2 tasks used for parallel table scan of file LINEITEM.     1
    Optimizer debug messages for distributed query step 2 of 2 follow:
    Arrival sequence access was used for file 0303288002.
    ODP created.
    Blocking used for query.
                                                                        Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 65. SMP Support for Distributed Query on Coordinator Node

Message **1** is produced on the coordinator system and shows that two tasks are scheduled on the coordinator node to process the local part of the file in parallel.

The following figure shows messages sent to the job log of the DDM job on SYSTEM01:

```
                              Display Job Log
                                                              System:   SYSTEM01
Job . . :   M03          User . . :   A960202B      Number . . . :   009008

    Job 009008/A960202B/M02 started on 09/10/96 at 11:31:43 in subsystem QCMN
      in QSYS. Job entered system on 09/10/96 at 11:31:43.
    Target DDM job started by source system.
    Arrival sequence access was used for file LINEITEM.
    DDM job started.
    DDM job started.
    8 tasks used for parallel table scan of file LINEITEM.     2




                                                                        Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F10=Display detailed messages   F12=Cancel
F16=Job menu           F24=More keys
```

Figure 66. SMP Support for Distributed Query on Remote Node

Message **2** shows that eight tasks were used on SYSTEM01 to process the part of the file on this node. This number is higher than for SYSTEM02 (or SYSTEM03 in the following figure) because SYSTEM01 was a larger system (9406 model F95) and the partitioning map for file LINEITEM was configured in such a way that SYSTEM01 contained a larger part of the entire file.

Now we can look at the messages produced on SYSTEM03:

```
                              Display Job Log
                                                              System:   SYSTEM03
Job . . :   DDMTOM03     User . . :   A960202B      Number . . . :   004021

    Job 004021/A960202B/M02 started on 09/10/96 at 11:30:41 in subsystem QCMN
      in QSYS. Job entered system on 09/10/96 at 11:30:41.
    Target DDM job started by source system.
    Arrival sequence access was used for file LINEITEM.
    DDM job started.
    DDM job started.
    2 tasks used for parallel table scan of file LINEITEM.    3




                                                                        Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F10=Display detailed messages   F12=Cancel
F16=Job menu           F24=More keys
```

Figure 67. SMP Support for Distributed Query on Another Remote Node

In Figure 67, message **3** shows that the same number of tasks (two) were used on SYSTEM03 as on SYSTEM02 (in Figure 65 on page 130). This is logical because these were systems of similar performance and the number of records for file LINEITEM on these nodes is almost equal.

12.8.4 CHGQRYA Command

The parallel implementation of queries is controlled by the Change Query Attributes (CHGQRYA) command as it widely discussed in the first part of this book, dedicated to Symmetric Multiprocessing.

In addition to the *DEGREE* parameter, there are two more parameters that can be used in an DB2 Multisystem environment to control parallelism:

- ASYNCJ - asynchronous job usage
- APYRMT - apply remote

In Figure 68, the prompted parameters for this command are shown.

```
                      Change Query Attributes (CHGQRYA)

Type choices, press Enter.

Job name . . . . . . . . . . . . JOB            *_________
  User . . . . . . . . . . . . .                  __________
  Number . . . . . . . . . . . .                 ______
Query processing time limit  . . QRYTIMLMT      *SYSVAL____
Parallel processing degree:      DEGREE
  Processing option  . . . . . .                *SYSVAL__
  Number of tasks  . . . . . . .                ______
Asynchronous job usage . . . . . ASYNCJ         *DIST_
Apply CHGQRYA to remote  . . . . APYRMT         *YES_









                                                                      Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 68. CHGQRYA Command Parameters

- The JOB parameter specifies the job to which this command applies. By default, the CHGQRYA command applies to the current job (where the command is issued).
- The QRYTIMLMT parameter specifies the maximum time you allow your query to run. This parameter has a special meaning for distributed queries, in that it applies to the time it takes to process the query on each node and not to the entire query run time. Although the time limit option can be useful, it should be used with caution with distributed queries. For example, when a multi-step query is performing a directed join (see Section 12.5.2, “Directed Join” on page 113), the time limit is checked **after** directing a file to remote nodes. For a large file, this can be a long time.
- The DEGREE parameter together with the QQRYDEGREE system value controls the parallel execution of local parts of a distributed query on every

node. It has the same affect as for ordinary queries. This parallel support provided by the DB2 Symmetric Multiprocessing for OS/400 feature is discussed in detail in the first part of this book.

- The ASYNCJ parameter is a new parameter introduced by the DB2 Multisystem for OS/400 feature. This parameter controls the usage of the temporary result writers (see Section 12.8.1, “Temporary Result Writers” on page 127). This parameter can have the following values:
  - *NONE - the temporary result writer is never used. In addition, communications with remote nodes are performed synchronously. This can be useful because it allows more performance messages to be returned from remote nodes to the coordinator job. You can easily test how your query is implemented on all nodes without going out to every node to look for performance-related messages.
  - *LOCAL - allows you to use temporary result writers for local queries only. Currently, there is no support in DB2/400 for using temp writers for local queries. This parameter disables the temp writers for distributed queries (just the same as *NONE does), but yet it allows communications with remote nodes to be performed asynchronously (see Section 12.8.2, “Parallel Interaction with Remote Nodes” on page 129).
  - *DIST - allows you to use temporary result writers for distributed queries only.
  - *ANY - allows you to use temporary result writers for all queries (both for local and distributed). Since there is no support for using temp writers for local queries, *ANY allows you to use temporary result writers for distributed queries, and at the same time, enables asynchronous communications with remote nodes.
- The APYRMT parameter is a new parameter introduced by the DB2 Multisystem feature. This parameter can be used to specify whether ASYNCJ parameter values should be propagated to remote system jobs that are used in distributed query processing.

12.8.5 Combining Parallel Options

All of the parallel execution options discussed in Section 12.8, “Parallel Support for Distributed Queries” on page 127 can be used together in different combinations. This gives you a flexible and powerful way to control the amount of system and network resources that are allocated for executing a query.

For example, you can do both of the following:

- Issue the CHGQRYA ASYNCJ(*ANY) APYRMT(*YES) command in a job where a distributed query is run.
- Set the QQRYDEGREE system value to *MAX on all nodes of your network.

Your query is executed with the maximum available degree of parallelism:

- Temporary result writer jobs are used to implement your query. This allows different steps of a multi-step query to run in parallel.
- The local system communicates asynchronously with remote nodes that allow all nodes to work in parallel on their parts of the query.
- On every node, the optimizer can schedule multiple parallel tasks for the execution of the local portion of the query.

The combination of DB2 Multisystem and of Symmetric Multiprocessing leads the AS/400 customers into the arena of massively parallel database processing. A

single query may be carried out by a large number of tasks, scattered across a network of AS/400 systems on each of which multiple CPUs may run concurrently.

These features can strongly improve the run time of an individual query. However, keep in mind that the parallel tasks needed to execute a single request may use a significant percentage of the available system resources:

- CPU cycles
- I/O subsystem
- Communications bandwidth

Other jobs on the systems may be affected and, therefore, resorting to maximum parallelism should be reserved to the execution of critical queries.

Appendix A. Database Files Used in Examples

We describe here the record layout of two files we used in most examples of queries throughout the book.

A.1 File LINEITEM

Table 7. LINEITEM Database File Structure

Field Name	Data Type
ORDERKEY	INTEGER
PART	INTEGER
SUPPLIER	INTEGER
LINENUMBER	INTEGER
QUANTITY	DECIMAL (15,2)
PRICE	DECIMAL (15,2)
DISCOUNT	DECIMAL (15,2)
TAX	DECIMAL (15,2)
FLAG	CHARACTER (1)
LINESTATUS	CHARACTER (1)
SHIPDATE	DATE
COMMITDATE	DATE
RECEIPTDATE	DATE
SHIPINSTRUCT	CHARACTER (25)
SHIPMODE	CHARACTER (10)
COMMENT	VARCHAR (44) ALLOCATE (0)

The total record length for file LINEITEM was 161 bytes.

A.2 File ORDERS

Table 8. ORDERS Database File Structure

Field Name	Data Type
ORDERKEY	INTEGER
CUSTOMER	INTEGER
ORDERSTATUS	CHARACTER (1)
PRICE	DECIMAL (15,2)
ORDERDATE	DATE
PRIORITY	CHARACTER (15)
CLERK	CHARACTER (15)
SHIPPRIORITY	INTEGER
COMMENT	CHARACTER (79)

The total record length for file ORDERS was 142 bytes.

Appendix B. Hash Function Table

You may want to use an additional field in your database file for the purpose of specifying on which node on the system every record should reside. This gives you direct control over the placement of individual records.

The easiest choice is to designate a field of numerical type as a partitioning key. This is simple to manage and the operating system can also efficiently perform the necessary calculations with minimal overhead.

Unfortunately, the operating system cannot use the value of a partitioning key directly. First, this value is fed to the hashing function which returns a partition number. Then this partition number is converted to a node number using the node group information stored within the distributed file object.

When used without proper consideration, many records can end up in the same partition. For example, for numbers between 0 and 1023, 24 different numbers produce the same hash value of 962 (partition number).

For your convenience, we have included two tables of 32 numbers that produce unique partition numbers. This gives you a simple way to assign a record to each node from node 1 through node 32.

Table 9. Partitioning Key to Partition Number Conversion

Key Value	Partition Number	Key Value	Partition Number	Key Value	Partition Number	Key Value	Partition Number
939	75	1037	662	1081	913	1099	9
969	278	1039	288	1083	49	1101	771
1006	85	1043	994	1085	77	1106	493
1013	918	1047	216	1089	333	1107	440
1017	652	1059	472	1091	725	1108	351
1028	1004	1061	267	1092	296	1109	286
1029	37	1063	443	1094	352	1112	1019
1035	544	1075	882	1098	72	1116	745

Table 10. Partition Number to Partitioning Key Conversion

Partition Number	Key Value	Partition Number	Key Value	Partition Number	Key Value	Partition Number	Key Value
9	1099	267	1061	440	1107	745	1116
37	1029	278	969	443	1063	771	1101
49	1083	286	1109	472	1059	882	1075
72	1098	288	1039	493	1106	913	1081
75	939	296	1092	544	1035	918	1013
77	1085	333	1089	652	1017	994	1043
85	1006	351	1108	662	1037	1004	1028
216	1047	352	1094	725	1091	1019	1112

Appendix C. PTF List for DB2 Multisystem for OS/400 Feature

This appendix contains the list of Program Temporary Fixes (PTFs) that at the time of publication were known to be available for the DB2 Multisystem for OS/400 feature.

Be sure to apply the latest cumulative PTF tape to your AS/400 system to ensure you have the latest program level on your system.

C.1 OS/400 Version 3 Release 2 Modification 0

Apply at least cumulative PTF tape **C6233320** plus the following PTFs, which at the time of preparing the book, were available electronically through Electronic Customer Support (ECS):

- SF33123
- SF33313
- SF33554
- SF33617
- SF33717
- SF33766
- SF34094
- SF34259
- SF34259
- SF34310
- SF34427

C.2 OS/400 Version 3 Release 7 Modification 0

OS/400 Version 3 Release 7 Modification 0 was not yet generally available at the time of preparing this book, so there was no information on cumulative PTF tapes for this release. The following list contains PTFs that are available electronically through Electronic Customer Support (ECS) at the time of publication:

- SF33627
- SF33979
- SF34237
- SF34260
- SF34312
- SF34380
- SF34381
- SF34382
- SF34383
- SF34665

Appendix D. Special Notices

This publication is intended to help database administrators, systems administrators, and technical personnel to design and manage solutions based on DB2 Multisystem for OS/400 and DB2 Symmetric Multiprocessing for OS/400. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 Multisystem for OS/400 or DB2 Symmetric Multiprocessing for OS/400. See the PUBLICATIONS section of the IBM Programming Announcement for these products for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

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

IBM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 145.

- *DB2/400 Advanced Database Functions, GG24-4249*

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** *and receive updates 2-4 times a year at significant savings.*

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

E.3 Other Publications

These publications are also relevant as further information sources:

- DB2 for OS/400 Database Programming, *SC41-3701*
- Distributed Database Programming, *SC41-3702*
- DB2 Multisystem for OS/400, *SC41-3705*
- DB2 for OS/400 SQL Programming, *SC41-4611*
- OS/400 Backup and Recovery - Advanced, *SC41-4305*

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269	
Canada	1-403-267-4455	
Outside North America	(+45) 48 14 2207	(long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

APPC	Advanced Program-to-Program Communications
DDM	Distributed Data Management
DRDA	Distributed Relational Database Architecture
NLS	National Language Support
ODBC	Open Database Connectivity
SQL	Structured Query Language
SMP	Symmetric Multiprocessing
VMC	Vertical Microcode

Index

A

abbreviations 149
acronyms 149
Allocate Object (ALCOBJ) command 80
ALWCPYDTA parameter
Apply Journal Changes (APYJRNCHG) command 76

B

bibliography 143

C

CCSID 90
CHGQRYA command
  APYRMT parameter 133
  ASYNCJ parameter 129, 133
  DEGREE parameter 10, 129, 132
Commitment control
  and DB2 Multisystem 76
  and SMP 14, 19
  two-phase protocol 77
Communications
  and distributed queries 104
  changing configuration 68
  choosing an APPC mode 104
  choosing the media 49
  communications failures 73, 77, 107
  configuration 86
  configuring the Relational Database Directory 50
CPU Parallelism 8

D

Database Performance Monitor and SMP 18
DBL3xx VMC tasks 8
DDM jobs 91
Debugger messages 19
DFU (Data File Utility) 49
Disaster recovery 79, 83
DISTINCT SQL clause 108
Distributed files
  and SBMRMTCMD 80
  availability 73, 108
  CCSID 90
  journaling 74
  locking 80
  managing 73
  National Language Support 90
  native I/O processing 108
  redistributing 99
  save/restore 80, 81
  security 85
Distributed queries
  access plan 101
  embedded SQL 102
  group by operations 123
  grouping and join 126
  join operations 110
  on a single file 106
  optimization 101
  parallel support 127
DRDA and SMP 12

E

Embedded SQL and Multisystem 102
Expert Cache and SMP 21

F

FMTDTA utility 20

G

GROUP BY and DB2 Multisystem
  combined with join 126
  one step processing 124

H

Hash function 52, 58, 64, 137
Hash Group By
  example and performance 38
Hash Join
  example and performance 44
HASH scalar function 53, 64

I

I/O Parallelism 7
Index Only Access
  example and performance 34
Isolatable record selection 107

J

Join queries
  broadcast join 120
  co-located join 111
  directed join 113
  Index Join 20
  Nested Loop Join 20, 43
  re-partitioned join 116, 120
Journal receivers 75, 80
Journaling and DB2 Multisystem
  access paths journaling 75
  apply changes 76
  start journaling 74

Journaling and DB2 Multisystem *(continued)*
stopping journaling 75

L
Limitations for SMP 19
Logical files and Multisystem 55

N
National Language Support
for distributed files 90
Node groups
changing 58, 66
creation 55
existence 68
restoring 66
unusable 67
working with 66
NODENAME scalar function 59, 71
NODENUMBER scalar function 59, 71

O
ODBC and SMP 12
OptiConnect 49
ORDER BY operation
and Hash Group By 41
and Symmetric Multiprocessing 14, 41
with DB2 Multisystem 108
with UNION or DISTINCT clauses 108

P
Parallel Index Scan
example and performance 27
Parallel Key Positioning 31
Parallel Table Scan
and Expert Cache 21
example 22
Parallel VMC tasks 10, 24, 29
Partition map 51
Partitioning file 62, 68
Partitioning key
choosing the partitioning key 62, 64
definition and example 52
Performance Tools and SMP 17
Print SQL Information (PRTSQLINF) 19

Q
QQRYDEGREE system value 10, 129

R
Redistributing data 71, 99
Relational Database Directory 50
Restoring distributed files 81

S
Saving distributed files 80
Security 85
user profiles 51
Submit Remote Command (SBMRMTCMD)
command 80

T
Temporary writers 95, 127

U
Uni-processor systems and SMP 15
UNION SQL clause 108
Unique access paths and Multisystem 54

V
Variable record length 49
Visibility node
definition 61
performance 65
usage 64

W
Work with Commitment Definition (WRKCMTDFN)
command 78

Printed in U.S.A.