Teradata
a division of NCR

Teradata® RDBMS
Performance Optimization

V2R5.0

B035-1097-122A
May 2003

The product described in this book is a licensed product of NCR Corporation.

BYNET is an NCR trademark registered in the U.S. Patent and Trademark Office.
CICS, CICS/400, CICS/600, CICS/ESA, CICS/MVS, CICSPLEX, CICSVIEW, CICS/VSE, DB2, DFSMS/MVS, DFSMS/VM, IBM, NQS/MVS, OPERATING SYSTEM/2, OS/2, PS/2, MVS, QMS, RACF, SQL/400, VM/ESA, and VTAM are trademarks or registered trademarks of International Business Machines Corporation in the U. S. and other countries.
DEC, DECNET, MICROVAX, VAX and VMS are registered trademarks of Digital Equipment Corporation.
HEWLETT-PACKARD, HP, HP BRIO, HP BRIO PC, and HP-UX are registered trademarks of Hewlett-Packard Co.
KBMS is a trademark of Trinzic Corporation.
INTERTEST is a registered trademark of Computer Associates International, Inc.
MICROSOFT, MS-DOS, MSN, The Microsoft Network, MULTIPLAN, SQLWINDOWS, WIN32, WINDOWS, WINDOWS 2000, and WINDOWS NT are trademarks or registered trademarks of Microsoft Corporation.
SAS, SAS/C, SAS/CALC, SAS/CONNECT, and SAS/CPE are registered trademarks of SAS Institute Inc.
SOLARIS, SPARC, SUN and SUN OS are trademarks of Sun Microsystems, Inc.
TCP/IP protocol is a United States Department of Defense Standard ARPANET protocol.
TERADATA and DBC/1012 are registered trademarks of NCR International, Inc.
UNICODE is a trademark of Unicode, Inc.
UNIX is a registered trademark of The Open Group.
X and X/OPEN are registered trademarks of X/Open Company Limited.
YNET is a trademark of NCR Corporation.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS-IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT WILL NCR CORPORATION (NCR) BE LIABLE FOR ANY INDIRECT, DIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS OR LOST SAVINGS, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The information contained in this document may contain references or cross references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that NCR intends to announce such features, functions, products, or services in your country. Please consult your local NCR representative for those features, functions, products, or services available in your country.

Information contained in this document may contain technical inaccuracies or typographical errors. Information may be changed or updated without notice. NCR may also make improvements or changes in the products or services described in this information at any time without notice.

To maintain the quality of our products and services, we would like your comments on the accuracy, clarity, organization, and value of this document. Please e-mail: teradata-books@lists.ncr.com

or write:

Information Engineering
NCR Corporation
100 North Sepulveda Boulevard
El Segundo, CA 90245-4361
U.S.A.

Any comments or materials (collectively referred to as "Feedback") sent to NCR will be deemed non-confidential. NCR will have no obligation of any kind with respect to Feedback and will be free to use, reproduce, disclose, exhibit, display, transform, create derivative works of and distribute the Feedback and derivative works thereof without limitation on a royalty-free basis. Further, NCR will be free to use any ideas, concepts, know-how or techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, or marketing products or services incorporating Feedback.

Copyright © 1996-2003, NCR Corporation
All Rights Reserved

Preface

Supported Software Release

This book supports Teradata RDBMS V2R5.0.

For details on how to obtain the latest version of Teradata books, see "Technical Information on the Web" on page x.

Changes to this Book

This book includes the following changes to support the current release:

Date	Description
May 2003	General reorganization of the book. Revised for V2R5.0 performance features. Incorporated performance information from *Best Practices for Teradata Performance.*
October 2001	Revised for V2R4.1 features

Date	Description
September 2000	• Added notation for UNIX only or Windows NT/Windows 2000 only throughout. • Added differences for Windows NT/Windows 2000 throughout. • Added new DBS Control Record fields: • Updated existing DBS Control Record fields with new default values and usage/performance notes, as appropriate • Updated PermDBSize information and upgrade instructions • Added CREATE INDEX changes • Added CREATE INDEX deadlock considerations • Added Lock Display utility • Added host utility (HUT) lock information, including Archive/Recovery processing and RELEASE LOCK command • Added new Windows NT/Windows 2000 utilities
June 2000	Moved Appendix A, Import/Export Utilities, to *Teradata RDBMS Database Administration.* Added usage comments, example updates, and recommendations to: – "HTMemAlloc Performance Field" – "MaxParseTreeSegs Performance Field" – "PermDBAllocUnit File System Field" – "ReadAhead Performance Field" – "Skew Allowance Performance Field" • Added join indexes to "Data Protection Options", "Fallback Option" • Added usage information on "Choosing AMP Clustering Assignments" • Added usage of and release 4.0 enhancements to join index capabilities • Added section: "Reserving Spool Space"

About This Book

Purpose

Teradata RDBMS Performance Optimization provides information that:

- Helps you ensure that Teradata operates at peak performance based on your applications and processing needs.
- Describes specific release-specific performance enhancements that require no user intervention.

Audience

The primary audience includes database and system administors, and applications developers.

The secondary audience is NCR support personnel, including field engineers and local as well as global support and sales personnel.

How This Book Is Organized

This book contains fifteen chapters:

- Chapter 1: "Introduction to Performance Issues" provides an introduction to performance issues, discusses symptoms that impede system performance, suggests recommended system conditions to measure, gives general guidelines for collecting data, lists performance monitoring tools and resources, describes system components and their value in tuning performance, discusses performance reasons to revisit database design, suggests ways to ensure that upgrades do not have adverse effects on performance.
- Chapter 2: "Collecting and Using Resource Usage Data" provides guides for collecting resource usage (ResUsage) data, summarizes ResUsage components, discusses ResUsage disk space overhead, suggests ways to use ResUsage data to help optimize UNIX performance, to analyze traffic flow between the Teradata RDBMS and its clients, to provide data on CPU utilization, disk utilization, and BYNET data, and to aid in capacity planning. The chapter also describes RSSmon and PM/API.
- Chapter 3: "Other Kinds of Data Collection" describes gathering AMPUsage data and provides information on Account String Expansion (ASE), Access Logging, and Database Query Log (DBQL).
- Chapter 4: "Baseline Benchmark Testing" includes guidelines for baseline benchmark testing, baseline profiling, and ways to use system and production canary, or heartbeat, queries, to monitor system performance.
- Chapter 5: "System Performance and the DBS Control Record" describes those fields in the DBS Control Record that may affect system performance.

- Chapter 6: "Tools for Monitoring and Tuning System Performance" provides information on tools that monitor and tune system performance, including System Activity Reporter (sar), xperfstate, TOP, blmstat, ctl and xctl, puma, and Resource Check Tools.
- Chapter 7: "Optimization through Workload and Session Management" describes Priority Scheduler, Priority Scheduler Administrator, and SET SESSION ACCOUNT, discusses job mix tuning, how to identify and handle "killer queries," and Teradata Dynamic Query Manager (TDQM). The chapter also summarizes client-specific monitoring and session control tools and session processing support tools.
- Chapter 8: "Performance Tuning Using Query Analysis Resources and Tools" describes such query analysis resources and tools as Query Capture Facility, Target Level Emulation, Teradata Visual EXPLAIN, Teradata System Emulation Test, Teradata Index Wizard, and Teradata Statistics Wizard, that help tune performance through application and physical database design.
- Chapter 9: "Managing Space" includes discussions of disk space issues, including running out of disk space and free cylinders, freeing cylinders, creating more space on cylinders, and managing spool space.
- Chapter 10: "Using, Adjusting, and Monitoring Memory" suggests ways to determine if the system is using memory effectively and ways to adjust shared memory and free memory, and provides information on File System Segments (FSG) cache and Cylinder Read.
- Chapter 11: "Teradata Manager and Performance" describes Teradata Manager applications (Performance Monitor [PMON] and Priority Scheduler Administrator, for example) and facilities that enable you to monitor real-time system performance, investigate system behavior, investigate the audit log, analyze workload trends as well as historical resource utilization, and set up alerts and events monitoring on the Teradata database.
- Chapter 12: "SQL and System Performance" discusses Structured Query Language (SQL) operations and system performance.
- Chapter 13: "Data Management"discusses data distribution issues, including the impact of uneven data distribution on performance, the way to determine table and database size growth rates, parallel efficiency, primary indexes and row distribution, purging and recommended housekeeping issues, and data protection options.
- Chapter 14: "Database Locks and Performance"provides information on locking levels and locking modes, deadlocks, deadlock handling, avoiding deadlocks, locking and requests, locking and transactions, locking rules, locking and client utilities, and transaction rollback and performance.
- Chapter 15: "General Performance Enhancements" describes release-specific enhancements requiring no user intervention that optimize system performance, including reconfiguration and performance, the effects of roles and profiles on performance, sessions, jobs, and performance, the

"Optimized DROP MACRO" feature, reducing all AMP operations, constraints optimization, update performance, Teradata TPump and join index, Optimizer enhancements, and parallel dump.

- Chapter 16: "Troubleshooting Database Performance" provides tips and techniques to help find the causes of performance problems.

Prerequisites

You should be familiar with your NCR hardware and operating system, your Teradata RDBMS and associated client products, and the utilities you can use to tune Teradata to improved performance.

List of Acronyms

This book uses acronyms, which the following table lists in alphabetical order:

2PC	Two-Phase Commit
AMP	Access Module Process
API	Application Interface
ANSI	American National Standards Institute
ARC	Archive Recovery
ASE	Account String Expansion
AWT	AMP Worker Task
BLMSTAT	BYNET Link Manager Status
BT	Begin Transaction
BTEQ	Basic Teradata Query
BTEQWIN	BTEQ Window
BYNET	Banyan Network
CCL	Customer Care Link
CI	Cylinder Index
CLI	Call-Level Interface
CPU	Central Processing Unit
CS	Correlated Subqueries
DAP	Disk Array Plus
DBA	Database Administrator
DBQL	Database Query Log
DBW	Database Window
DDL	Data Definition Language
DIP	Database Initialization Program
DML	Data Manipulation Language
DSS	Decision Support Systems
DUC	Dynamic Utilization Charting
DWGSC	Data Warehouse Global Support Center
ELA	Error Log Analyzer

ET	End Transaction
FSG	File System Segment
FSP	FreeSpacePercent
GDO	Globally Distributed Object
GMT	Greenwich Mean Time
GSS	Global Sales Support
GUI	Graphical User Interface
HATP	High Availability Transaction Processing
HSI	Host System Interface
HUT	Host Utility
I/O	Input/Output
JDBC	Java Database Connectivity
LAN	Local Area Network
LDV	Logical Device
LSN	Logon Sequence Number
LT/ST	Large Table/Small Table
MPP	Massively Parallel Processing
MVS	Multiple Virtual Storage
NUPI	Non-Unique Primary Index
NUSI	Non-Unique Secondary Index
ODBC	Open Database Connectivity
ODS	Operational Data Store
OLCP	Online Complex Processing
OLTP	Online Transaction Processing
PCI	Peripheral Component Interconnect
PDA	Performance Data Analyzer
PDE	Parallel Database Extensions
PE	Parsing Engine
PI	Primary Index
PJ	Permanent Journal
PM/API	Performance Monitor/Applications Program Interface

PMON	Performance Monitor
PM & PC	Performance Monitor and Production Control
PPI	Partitioned Primary Index
PSA	Priority Scheduler Administrator
PUT	Parallel Upgrade Tool
QCD	Query Capture Database
QCF	Query Capture Facility
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RDBMS	Relational Database Management System
ResUsage	Resource Usage
RI	Referential Integrity
RSS	Resource Sampling Subsystem
SAR	System Activity Reporter
SCSI	Small Computer Systems Interface
SMF	System Management Facility
SMP	Symmetric Multi-Processing
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
TDP	Teradata Director Program
TDPUTCE	TDP User Transaction Collection Exit
TDQM	Teradata Dynamic Query Manager
TGSC	Teradata Global Support Center
TJ	Transient Journal
TLE	Target Level Emulation
TMON	Time-Monitored
TPA	Trusted Parallel Application
TPCCONS	Two-Phase Commit Console Interface
TSET	Teradata System Emulation Test
UPI	Unique Primary Index
USI	Unique Secondary Index

VECOMP	Visual Explain and Compare
vdisk	Virtual Disk
VLC	Value List Compression
VM	Virtual Machine
vproc	Virtual Processor

Technical Information on the Web

The NCR home page (http://www.ncr.com) provides links to numerous sources of information about Teradata. Among the links provided are sites that deal with the following subjects:

- Contacting technical support
- Enrolling in customer education courses
- Ordering and downloading product documentation
- Accessing case studies of customer experiences with Teradata
- Accessing third party industry analyses of Teradata data warehousing products
- Accessing white papers
- Viewing or subscribing to various online periodicals

Contents

Preface

About This Book iii
List of Acronyms vi
Technical Information on the Web x

Chapter 1:
Introduction to Performance Issues

Why Monitor Performance? 1–2
Some Symptoms of Impeded System Performance 1–3
Measuring System Conditions 1–5
Guidelines for Collecting Data 1–7
Performance Monitoring Tools and Resources 1–8
System Components and Performance Monitoring 1–12
Revisiting Database Design 1–14
Performance Considerations When Upgrading 1–16

Chapter 2:
Collecting and Using Resource Usage Data

Collecting Resource Usage Data 2–2
ResUsage Components 2–3
ResUsage Tables 2–4
ResUsage Views 2–8
ResUsage Macros 2–9
Collect and Log Rates 2–12
ResUsage Disk Space Overhead Considerations 2–13
Optimizing ResUsage Logging 2–14
ResUsage and Teradata Manager Compared 2–16
ResUsage and DBC.AMPUsage View Compared 2–17
Extended ResUsage Macros and UNIX Performance 2–18
ResUsage and Host Traffic 2–20
ResUsage and CPU Utilization 2–22
ResUsage and Disk Utilization 2–31
ResUsage and BYNET Data 2–34

ResUsage and Capacity Planning 2–41
Resource Sampling Subsystem Monitor 2–42
PM/API and Performance 2–43

Chapter 3:
Other Kinds of Data Collection
DBC.AMPUsage View 3–2
Account String Expansion 3–4
Access Logging 3–7
Database Query Log 3–9

Chapter 4:
Baseline Benchmark Testing
What is a Benchmark Test Suite? 4–2
Baseline Profiling 4–3
Baseline Profile: Performance Metrics 4–4
Reliable Baseline Profiles 4–5
Canary, or Heartbeat, Queries 4–6
System Canaries 4–7
Production Canaries 4–8

Chapter 5:
System Performance and the DBS Control Record
DBS Control Record 5–3
Cylinders Saved for PERM 5–4
DBSCacheCtrl 5–5
DBSCacheThr 5–6
DeadLockTimeout 5–8
DefragLowCylProd 5–9
DictionaryCacheSize 5–10
DisableSyncScan 5–11
FreeSpacePercent 5–12
HTMemAlloc 5–13
IdCol Batch Size 5–14
JournalDBSize 5–15
LockLogger 5–16
MaxDecimal 5–17
MaxLoadTasks 5–18
MaxParseTreeSegs 5–19
MiniCylPackLowCylProd 5–20

PermDBAllocUnit 5–22
PermDBSize 5–24
PPICacheThrP 5–28
ReadAhead 5–29
ReadAheadCount 5–30
RedistBufSize 5–31
RollbackPriority 5–33
RollbackRSTransaction 5–34
RollForwardLock 5–35
RoundHalfwayMagUp 5–36
RSDeadLockInterval 5–37
Session Mode 5–38
SkewAllowance 5–39
StepsSegmentSize 5–41
SyncScanCacheThr 5–42
TargetLevelEmulation 5–44
WriteDBsToDisk 5–45

Chapter 6:
Tools for Monitoring and Tuning System Performance
How Busy is Too Busy? 6–2
System Activity Reporter 6–4
xperfstate 6–8
sar and xperfstate Compared 6–10
sar, xperfstate, and ResUsage Compared 6–11
TOP 6–13
BYNET Link Manager Status 6–14
ctl and xctl 6–15
puma (NCR UNIX) 6–16
Resource Check Tools 6–21

Chapter 7:
Optimization through Workload and Session Management
Priority Scheduler 7–2
SET SESSION ACCOUNT and Performance 7–8
Accessing Priority Scheduler: Priority Scheduler Administrator, schmon, and xschmon 7–11
Priority Scheduler Simulator 7–12
Job Mix Tuning 7–13
Identifying and Handling "Killer Queries" 7–15

Teradata Dynamic Query Manager 7–16
Client-Specific Monitoring and Session Control Tools 7–17
Session Processing Support Tools 7–19

Chapter 8:
Performance Tuning Using Query Analysis Resources and Tools
Query Analysis Resources and Tools 8–2
Query Capture Facility 8–3
Target Level Emulation 8–4
Teradata Visual EXPLAIN 8–5
Teradata System Emulation Tool 8–6
Teradata Index Wizard 8–7
Teradata Statistics Wizard 8–8

Chapter 9:
Managing Space
Running Out of Disk Space 9–2
Running Out of Free Cylinders 9–3
FreeSpacePercent 9–5
PACKDISK and FreeSpacePercent 9–8
Freeing Cylinders 9–10
Creating More Space on Cylinders 9–14
Managing Spool Space 9–18

Chapter 10:
Using, Adjusting, and Monitoring Memory
Using Memory Effectively 10–2
Shared Memory (UNIX) 10–3
Free Memory (UNIX) 10–5
FSG Cache 10–9
Managing I/O with Cylinder Read 10–10
Monitoring Memory 10–16

Chapter 11:
Teradata Manager and Performance
What is Teradata Manager? 11–2
Performance Analysis and Problem Resolution 11–3
Database Management 11–5
Operational Control 11–6
Using Teradata Manager Applications 11–8

Performance Impact of Teradata Manager 11–11
Monitoring Real-Time Activity 11–12
Investigating System Behavior 11–13
Investigating the Audit Log 11–14
Analyzing Workload Trends 11–15
Analyzing Historical Resource Utilization 11–16
Using Priority Scheduler Administrator to Balance the Workload 11–17
Using Alerts to Monitor the System 11–18
Using Performance Monitor Object 11–20
Teradata Manager and ResUsage Compared 11–21
Teradata Manager and DBC.AMPUsage View Compared 11–22
Teradata Manager Compared with HUTCNS and DBW Utilities 11–23
Teradata Manager and SHOWSPACE Compared 11–25
Teradata Manager and TDP Monitoring Commands 11–26
Teradata Manager and HSI, TDPUTCE, and SMF 11–28
Teradata Manager and Job Control Support Tools 11–29

Chapter 12:
SQL and System Performance

CREATE/ALTER TABLE and Data Retrieval 12–2
Compressing Columns 12–6
Correlated Subqueries 12–8
Concatenation and Correlated Subqueries 12–10
CASE Expression 12–11
Ordered Analytical Functions 12–15
Extending DATE with the CALENDAR System View 12–18
Optimized INSERT SELECTs 12–19
SQL Techniques that Reduce Row Redistribution 12–22
Merge Joins 12–25
Hash Joins 12–26
Secondary Indexes 12–27
Join Indexes 12–31
Joins and Aggregates On Views 12–38
Joins and Aggregates on Derived Tables 12–40
Derived Table Optimization 12–42
GROUP BY Operator and Join Optimization 12–43
Outer Joins 12–44
Large Table/Small Table Joins 12–46
Volatile and Global Temporary Tables 12–48
Partitioned Primary Index 12–50
Identity Column 12–52
Collecting Statistics 12–53

Referential Integrity 12–62
2PC Protocol 12–66
Updatable Cursors 12–67
Sparse Indexes 12–68
EXPLAIN Feature and the Optimizer 12–69

Chapter 13:
Data Management
Data Distribution Issues 13–2
Identifying Uneven Data Distribution 13–3
Parallel Efficiency 13–7
Primary Index and Row Distribution 13–8
Recommended Housekeeping 13–10
Data Protection Options 13–13
PDE-Dump and Disk Array Plus (NCR UNIX) 13–20
Determining Table and Database Size Growth Rates 13–21

Chapter 14:
Database Locks and Performance
Locking Overview 14–2
What Is a Deadlock? 14–5
Deadlock Handling 14–7
Avoiding Deadlocks 14–9
Locking and Requests 14–10
Locking and Transactions 14–11
Locking Rules 14–12
LOCKING ROW/NOWAIT 14–13
Locking and Client (Host) Utilities 14–14
Transaction Rollback and Performance 14–16

Chapter 15:
General Performance Enhancements
Reconfiguration and Performance 15–2
Effect of Roles and Profiles on Performance 15–4
Sessions, Jobs, and Performance 15–6
"Optimized DROP MACRO" Feature 15–7
Reducing All AMP Operations 15–8
Constraints Optimization 15–12
Update Performance Features 15–13

Teradata TPump and Join Index 15–16
Optimizer Enhancements 15–19
Parallel Dump (UNIX) 15–21

Chapter 16:
Troubleshooting Database Performance
Troubleshooting a Hung or Slow Job 16–2
Identifying Peak Periods and Bottlenecks 16–6
Job Scheduling Around Peak Utilization 16–7
Expanding the System 16–8
How to Catch a "Killer Query" 16–12
Running Out of TDP Memory 16–17
Lock Problems 16–18
Lack of Disk Space 16–19
Hardware Faults 16–20
Skewing 16–21

Index Index–1

Chapter 1:

Introduction to Performance Issues

This chapter provides an introduction to Teradata performance issues.

Topics include:

- Why monitor performance?
- Some symptoms of impeded system performance
- Measuring system conditions
- Guidelines for collecting data
- Performance monitoring tools and resources
- System components and performance monitoring
- Revisiting database design
- Performance considerations when upgrading

Why Monitor Performance?

To Identity System Problems

Monitoring and tracking system performance makes identifying system problems easier to accomplish. Setting up the system environment, for example, to enable system administrators to react quickly to a performance issue protects the availability of the system.

For Performance Tuning

In addition to making the identification of system problems easier, monitoring system performance also provides necessary data to support performance tuning. Such tuning can yield more efficient use of existing system resources than might otherwise result through workload management or through an increase in system resources brought about by capacity planning.

Managing resources, for example, which includes job mix tuning in order to use available idle cycles effectively and to meet pent-up demand that may peak during prime hours, ensures that the system operates efficiently.

For Capacity Planning

If performance degradation is a gradual consequence of increased growth and shrinking capacity, all data collected over time can be used for capacity or other proactive planning. Since the onset of growth-related performance degradation can often be insidious, taking measurements and tracking both data and usage growth can be very useful.

Some Symptoms of Impeded System Performance

System Saturation and Resource Bottlenecks

All database systems, including Teradata, reach saturation from time to time, particularly in ad-hoc query environments where end-users may saturate a system unless you control elements such as spool space or job entry.

System saturation and bottlenecks are interrelated. When the system is saturated, the bottleneck is usually some key resource, such as a CPU or disk. Looking at how often a resource or session is in use during a given period and asking such questions as the following help, for example, identity resource bottlenecks:

- How intensively was the AMP CPU used?
- Are all AMPs working equally hard?
- What are input/output counts and sizes for disks, BYNET, and client channel connections or Ethernet?

You can use the information obtained from resource usage, as well as system monitoring tools, to find where, and when, bottlenecks occur. Once you know which resource is frequently the bottleneck in certain applications, you can, for example, modify your job entry or scheduling strategies and justify system upgrades or expansions or tune your workloads for more efficient use of resources.

Processing Concurrency: Lock Conflicts and Blocked Jobs

Transaction locks are used to control processing concurrency. The type of lock (exclusive, write, read, or access) imposed by a transaction on an entity (database, table, rowhash rank, or rowhash) determines whether subsequent transactions can access the same entity.

A request is queued when a lock it needs cannot be granted because a conflicting lock is being held on the target entity. Such lock conflicts can hamper performance. For example, several jobs could be blocked behind a long-running insert into a popular table.

To resolve lock conflicts, you need to identify what entity is blocked and which job is causing the block. Then you may want to abort the session that is least important and later reschedule the long job to run in off hours.

Deadlocks

A deadlock can occur when two transactions each need the other to release a lock before continuing, with the result that neither can proceed. This occurrence is rare because Teradata uses a pseudo table locking mechanism at the AMP

level (see "AMP-Level Pseudo Locks and Deadlock Detection" on page 14-7), but it is possible. To handle the exceptions, the dispatcher periodically looks for deadlocks and aborts the longest-held session.

You can control the time it takes for a deadlock to detect and handle a deadlock automatically by shortening the cycle time of the Deadlock Detection mechanism. You can modify this value in the tunable Deadlock Timeout field of the DBS Control Record.

Measuring System Conditions

Tabular Summary

The following table summarizes recommended system conditions to measure:

System Conditions	What to Use	What to Collect	How to Use
Response time	Canary queries (also known as "heartbeat queries")	• Response time samples • System level information	Saved samples can be used to: • Track trends. • Monitor and alert • Validate Priority Scheduler configuration
	Baseline testing	• Response time samples • Execution plans	Check for differences +/- in response time Check EXPLAINs if degradation is present Collect / keep information for comparison when there are changes to the system
	DBQL	Application response time patterns	Track trends: • To identify anomalies • To identify performance tuning opportunities • For capacity planning
Resource utilization	ResUsage	• ResNode Macro set • SPMA summarized to one row per node	Look for: • Peaks • Skewing, balance
	AMPusage	CPU and I/O for each unique account	Use this to quantify heavy users
Data growth	• Script row counts • Permspace	Summary row per table once a month	Look at trends

System Conditions	What to Use	What to Collect	How to Use
Changes in data access	Access log	Summary row per table and the number of access once a month	Look for increases in access, trends
Increase in the number of active sessions	Logonoff, acctg	Monthly and summarized number of sessions	Look for increases in concurrency and active users
Increase in system demand	DBQL	Query counts and response times, plus other query information	Look for trends, including growth trends and measure against goals

Guidelines for Collecting Data

In collecting performance data, the following general guidelines are recommended:

Data Collection Infrastructure

Establish a data collection infrastructure. To do so, ensure the proper scope and granularity based on:

- Time
- Workload grouping

Use Accounts and Account Strings Expansion (ASE) to assure data collections for each query executed is associated to a particular time and world group. See Chapter 3: "Other Kinds of Data Collection" for information on Accounts IDs and ASE.

Primary Collection Points

Collect data at primary collection points. These include:

- CPU, memory, and I/O data collected by node and by AMP
- CPU and I/O data collect by account, user, workgroup, and time of day
- Growth data, a minimum of monthly data by table
- Spool usage data by system usage and by workload and user
- Canary/heartbeat response times
- Query logging

  Use query logging to capture query/statement counts and response times, to discover potential application improvements and further refinements to workload groupings and scheduling.

Other Collection Points

Collect data at additional collection points, using, for example:

- Access logging.

  Use access logging to discover table access for application improvements and other possible tuning such as DBScache adjustments.

Performance Monitoring Tools and Resources

You can use the following Teradata performance tools and resources to analyze and improve system performance.

Note: Tools that are available only on one type of operating system are identified as either "NCR UNIX" or "Windows 2000". If a tool is available on both platforms, no operating system is noted.

Tool	Description	For more details, see...
Access controls	• Use the REVOKE statement to revoke an explicitly granted privilege from the specified user • Use the REVOKE LOGON statement to revoke access from one or more specified client platforms	*Teradata RDBMS SQL Reference* *Teradata RDBMS Security Administration*
Access Logging facility	Suite of tools enabling C2 security that log the results of checks for access privileges. Log information can include user or session, privilege checked, and database entity. **Note:** This feature incurs processing overhead, but can be useful for identifying bottlenecks or lock conflicts.	*Teradata RDBMS Security Administration*
ctl (Windows 2000)	Tool you can use to display or modify the fields of the Parallel Database Extension (PDE) control parameters.	*Teradata RDBMS Utilities*
Database Window (DBW)	GUI (Graphical User Interface) that runs on the AWS (Administrative Workstation) available with MPP platforms. With DBW, you can execute commands via the Console Subsystem, view the status of AMPs and disks, and otherwise run other administrative tools for your Teradata RDBMS.	*Teradata RDBMS Database Windows*

DBC.AMPUsage	System view that shows AMP usage by user and account.
	Use this view to determine which users are submitting jobs that are CPU- or I/O intensive[1]; for example:
	`.set retlimit 10`
	`select username,processor,sum (cputime),sum(diskio)`
	`from dbc.ampusage`
	`where processor ='1-0'`
	`order by 2,3 desc`
	`group by 1,2;`
		Teradata RDBMS Data Dictionary

Tool	Description	For more details, see...
DBC.DiskSpace	System view that provides information by AMP on disk space usage, including permanent and spool data by database or account for each AMP. Use this view to track large spool usage.[1]	*Teradata RDBMS Data Dictionary*
DBC.LogOnOff	System view that provides historical information about logon and logoff activity. Use this view to identify users logged on at some time in the past.[1]	*Teradata RDBMS Data Dictionary*
DBC.SessionInfo	System view that provides information about users who are currently logged on. Use this view to determine the logon times of current sessions.[1]	*Teradata RDBMS Data Dictionary*
DBC.Software_ Event_Log	System view that provides information about hardware and software failures and severity, diagnostic information, and so on. Use this view to identify errors or system failures.[1]	*Teradata RDBMS Data Dictionary*

DBC.TableSize	System view that provides information about disk space usage (excluding spool), by database and table, for each AMP.
	You can query this view to identify tables with skewed data distribution;[1] for example:
	`select AMP, CurrentPerm`
	`from DBC.TableSize`
	`where DatabaseName =`
	`'`*`databasename`*`'`
	`and TableName = '`*`tablename`*`'`
	`order by 1;`
		Teradata RDBMS Data Dictionary

dbscontrol	Command-line utility that allows you to display and modify the tunable parameters in the DBS Control Record.	*Teradata RDBMS Utilities*
DEFRAGMENT	Ferret tool that combines free sectors on a cylinder.	*Teradata RDBMS Utilities*
Database Initialization Program (DIP)	A suite of script files containing SQL statements that are packaged with the RDBMS. Execute one or more files to create special system tables, views, and macros that you use to monitor system and user activity.[1]	*Teradata RDBMS Utilities*
dipviews	Script file (one of a suite comprising the DIP) that contains the SQL definitions for the system views. Execute this script to create the views that provide access to commonly referenced information in the underlying system tables. System views are created in system database DBC; thus, a fully-qualified reference is DBC.*viewname*.	*Teradata RDBMS Data Dictionary*

Tool	Description	For more details, see...
EXPLAIN	SQL statement that allows you to see the steps and access path that would be used to process the following SQL statement. EXPLAIN is useful for constructing complex queries, especially those planned for repetition, and for evaluating the syntax of slow or problem queries.	*Teradata RDBMS SQL Reference*
Lock Display	Utility that displays active transaction locks that control concurrent processing and that can be applied to the rowhash level. You can specify the display by: • A specific AMP • A group of AMPs • All AMPs This utility is useful for finding a session that is blocking others, particularly when you need to break a deadlock.	*Teradata RDBMS Utilities*
PACKDISK	Ferret tool that moves data blocks to create free cylinders	*Teradata RDBMS Utilities*
Priority Scheduler	Facility based on weighted priorities that lets you fine-tune the allocation of CPU resources among concurrent sessions.	*Teradata RDBMS Utilities*
qrysessn	Tool that provides information on the processing state for both SQL and utility sessions. State details can include Parsing, Active, Blocked, Response, whether stored procedures are being processed, and so on.	*Teradata RDBMS Utilities*
Recovery Manager	Utility that provides a way to measure and predict the progress of down-AMP recovery time.	*Teradata RDBMS Utilities*
ResUsage	Macros, tables, and views that monitor, record, and report the use of your Teradata resources, including CPU and I/O consumption, during transaction processing.	*Teradata RDBMS Resource Usage Macros and Tables*
RSSmon (NCR UNIX)	Utility that displays the UNIX and PDE resource usage data that was sampled and stored by the RSS (Resource Sampling Subsystem).	*Teradata RDBMS Resource Usage Macros and Tables* *Teradata RDBMS Utilities*
sar	Tool that monitors and records the use of system resources.	*Teradata RDBMS Resource Usage Macros and Tables*
ScanDisk	Command in Ferret Utility that checks File System integrity (integrity of internal data block structures and the consistency between cylinder indexes and data blocks). ScanDisk speed is affected by the data block pre-load factor you set in the ReadAhead field of the DBS Control GDO.	*Teradata RDBMS Utilities* *Teradata RDBMS Database Administration*

Tool	Description	For more details, see...
ShowLocks	Tool that identifies and displays all active HUT (Host Utility) locks placed on databases and tables by Table Rebuild or Archive Recovery (ARC) operations. To release them, use the RELEASE LOCK option or command of the utility.	• *Teradata Archive/Recovery* • *Teradata RDBMS Utilities*
SHOWSPACE	Function running under the Ferret utility that displays disk cylinder utilization and available free cylinders.	*Teradata RDBMS Utilities*
stune file (NCR UNIX)	File in which you can modify the LOTSFREE, DESFREE, and MINFREE parameters.	
Teradata MultiTool (Windows 2000)	GUI (Graphical User Interface) that enables you to invoke command-line utilities and tools used to administer Teradata and its sessions on a Windows 2000 platform. Allows up to four application windows which can communicate with Teradata RDBMS simultaneously.	*Teradata RDBMS Utilities*
Teradata Manager	Performance monitoring system that displays historical and near real-time performance data.	Teradata Manager online help
TOP (NCR UNIX)	Node-level performance tool for UNIX that displays the top 15 processes by percentage of CPU usage.	
TPCCONS (Two-Phase Commit Console Interface)	Command-line interface you can use to investigate and resolve Two-Phase Commit (2PC) in-doubt transactions.	*Teradata RDBMS Utilities*
Vproc Manager	Utility that allows you to view the status of the AMPs and perform such administrative functions as initializing an AMP or disk or forcing a manual Teradata restart. Use this utility to recover the system if an AMP stays down.	*Teradata RDBMS Utilities*
xctl (NCR UNIX)	Tool that allows you to display or modify the PDE control parameters, specifically FSG Cache Percent.	*Teradata RDBMS Utilities*
xperfstate (NCR UNIX)	PDE tool that monitors the state of system components in real time.	*Teradata RDBMS Utilities*

[1] If Teradata reports that the view is not found, check the contents of database DBC to verify that your system views have been created. If they have not, run the dipviews script file (see Database Initialization Program (DIP) and dipviews, above).

System Components and Performance Monitoring

Tabular Summary

The following table explains the place of system components in monitoring and tuning performance:

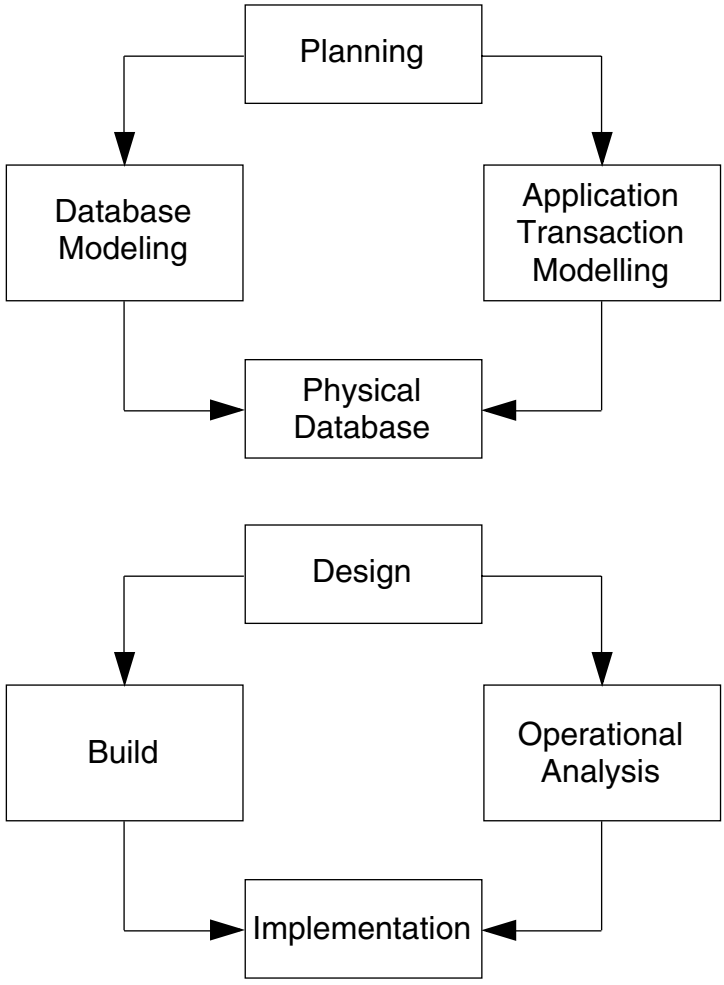
System Component	Performance Considerations
CPUs	Because all servers used with Teradata have multiple Central Processing Units (CPUs), you should know the number of CPUs and their speeds.
Vdisks	If your database is out of space, you need more Virtual Disks (vdisks). You can archive and delete data from existing vdisks, or add disk arrays.
Vprocs	Virtual Processes (vprocs), which include Access Module Process (AMP) and Parsing Engine (PE) vprocs, should be balanced to have the same processing power, memory allotment and I/O capacity.
Memory	Increased memory often improves performance for certain applications. Always ensure that memory is adequate when increasing the number of vprocs. Prepare for such contingencies as vprocs migrating when a node is down in a clique.
AMPs	An AMP vproc, which executes I/O tasks, should be balanced to have the same processing power, memory allotment and I/O capacity.
PEs	120 concurrent sessions can be supported per PE. You can, therefore, increase the number of sessions that run concurrently by adding PEs to a single node. Be sure, however, that the gateway on each node affected can support the total number of sessions.
Gateway	Because the gateway has a limit of 1200 sessions per node, be sure that the session limit can accommodate the number of PEs on that node.
LAN connection	You must have a separate Local Area Network (LAN) connection for each logical LAN. A group of LAN PEs service requests for the LANs connected to your server.

System Component	Performance Considerations
Channel connection	A channel connects a Teradata platform to a mainframe host. A separate channel is needed for each host. For example, an IBM Virtual Machine (VM) requires one channel, and an IBM Multiple Virtual Storage (MVS) requires another. You should have multiple connections per channel for redundancy.
Channel driver	Each node can accommodate one channel driver. The driver has no session limit.
NCR Banyan Network (BYNET) hardware/ software	The BYNET, a high-speed node interconnect designed for optimum Multiple Parallel Processing (MPP) operation, increases Teradata RDBMS availability and performance with: • Wide bandwidth and quad redundancy • Dynamic, automatic load balancing of node traffic and vproc reconfiguration • Broadcast or point-to-point communication, as appropriate for fastest query performance
System disks	In addition to the operating system and your applications, system disks store Teradata executables. Teradata also obtains dump and swap space from the system disks. Be sure to plan for this consumption.

Revisiting Database Design

Design Phases

The following diagram illustrates the phases of database design:



After initial database design implementation, you may experience:

- Expanded or additional applications
- Different data demographics
- Different processing windows
- Different transaction frequencies
- Different access paths

Many database designs are denormalized to support one or two particular applications. When you add new applications, the denormalization that helped

the performance of earlier applications may actually hinder the performance of later applications.

If you add new applications, or if you gather any other information during the design phase changes, you may need to revert to the Application Transaction Modeling phase of database design and proceed from there. Reiteration is a necessary fact of life in system design and development.

Basic Rules

The best way to choose indexes is to follow all the steps outlined in the above figure. This will ensure the best performance across all applications and transactions.

At times you may need to take shortcuts. Either you have inherited a poor design, or the requirements have changed and you have only a short time frame in which to make performance improvements. In these cases, the following basic rules may help:

- You can improve DSS (Decision Support System) performance by reconsidering Primary Indexes (PIs) to improve join performance for large tables.
- Common join columns usually make good PIs:
  - They make it possible for Teradata to perform PI-based merge joins.
  - For multi-column joins, a good PI may be all join columns or some subset of the join columns.
- To minimize searching through non-qualifying columns, the choice of PIs should be through the most frequent table access path.
- Common selection columns usually make good secondary indexes for DSS. For multi-column accesses, a good secondary index may be all access columns or some subset of the access columns.

  Common selection columns can also make good PIs, especially if one column also serves as a join column.

Caveats

Be aware of the following caveats if you decide to change your PIs based on join requirements:

- If you include too many columns in the PI, those queries which do not specify ALL columns will not access the table via the PI but rather via a scan, eliminating PI access or merge join from the Optimizer's plan. Alternative access or joins for big tables are expensive.
- If you include too few columns in the PI, each PI value may correspond to thousands of rows. This may cause data skewing, as well as performance degradation, on operations (such as data maintenance) that must touch all rows in a row hash.

Performance Considerations When Upgrading

From time to time, you will be faced with an upgrade, either to accommodate your data warehouse growth, or for some other reason.

Recommendations

To ensure performance after a major/minor software or hardware upgrade (new hardware costs will affect Optimizer plans), we recommend that you perform the following tasks on the production system:

- Recollect statistics where possible.
- Create a baseline test suite.
- Run the baseline on the production system before the upgrade.
- Save the EXPLAINs for the base queries.
- Run the test bed again after the upgrade, and get new EXPLAINs.
- Compare the two EXPLAINS.
- Check for faster/slower response on any of the test queries.
- If slower, look at the before and after EXPLAIN.
- Check statistics.
  - If stale, recollect.
  - If not, report this as a performance regression problem to Teradata Support immediately.

Performing these simple activities, even with a small group of queries (5-10 minimum are recommended), will help validate your performance results and expectations following any sort of important system change.

Chapter 2:

Collecting and Using Resource Usage Data

This chapter discusses collecting and using resource usage (ResUsage) data.

Topics include:

- Collecting resource usage data
- ResUsage components
- ResUsage tables
- ResUsage view
- ResUsage macros
- Collect and log rates
- ResUsage disk space overhead considerations
- Optimizing ResUsage logging
- ResUsage and Teradata Manager compared
- ResUsage and DBC.AMPUsage view compared
- Extended ResUsage macros and UNIX performance
- ResUsage and host traffic
- ResUsage and CPU utilization
- ResUsage and disk utilization
- ResUsage and BYNET data
- ResUsage and capacity planning
- Resource Sampling Subsystem Monitor (RSSmon)
- PM/API and performance

Collecting Resource Usage Data

Introduction

In order to understand system performance, resource usage (ResUsage) data must be collected over time.

Such data is useful in understanding current performance and growth trends. It can also be used for troubleshooting.

What Resource Usage Data Can Tell You

ResUsage data is useful for:

- Measuring system benchmarks
- Measuring component performance
- Assisting with on-site job scheduling
- Identifying potential performance impacts
- Analyzing performance degradation and improvement
- Identifying problems such as bottlenecks, parallel inefficiencies, down components, and congestion
- Planning installation, upgrade, and migration

For complete information on how to collect ResUsage data, see *Teradata RDBMS Resource Usage Macros and Tables*.

ResUsage Components

The following table describes the main components of ResUsage monitoring:

Component	Description	See
Tables	ResUsage actually consists of seven tables. You can specify the intervals at which resource utilization data is collected and logged in the ResUsage tables. You can also determine which tables are to be logged.	page 2-4
Views	Five views act as windows into the ResUsage tables.	page 2-8
Macros	Eight sets of ResUsage macros report on all nodes in your system. Use ResUsage macros to select data from the ResUsage tables. Each macro produces a ResUsage report.	page 2-9

For full information on the ResUsage tables, view, and macros, see *Teradata RDBMS Resource Usage Macros and Tables*.

ResUsage Tables

Controlling Tables

The system automatically maintains ResUsage tables that you can control via the logging interval setting in one of the following:

- Resource Sampling Subsystem (RSS) settings in the xctl utility
- SET commands in DBW
- PMON in Teradata Manager

Types of Tables

The system maintains two types of ResUsage tables:

1. One type contains data about the node, including CPU utilization, BYNET activity, memory, external connections, and so on.
2. A second type contains data about the vprocs. Data in this table type describes logging activity for the AMPs and PEs.

System Activities

The following table describes the system activities for which the ResUsage tables provide data.

System Activity	Description
Process scheduling	CPU time, switching, interrupts
Memory	Allocations, deallocations, logical memory reads and writes, physical memory reads and writes
Network	BYNET traffic patterns, messages (size and direction)
General concurrency control	User versus system
Teradata File System	Logical and physical disk, locking, cylinder maintenance
Transient journal management	Purging activity
Physical and logical disks	I/O reads and writes, KB transferred, I/O times

Table Descriptions

The following table describes each of the eight ResUsage tables:

Table	Type of Information	Description of Contents
ResUsageSpma ResUsageIpma	General node-level Internal node-level	These tables log one record for each node in the system for each node interval, and record: • vprocs • Process scheduling and pending • Process blocking and wait times • CPU utilization • Memory management • Node backup activity • BYNET traffic (point-to-point and broadcast) • Merge processing • Physical and logical database I/O. The Ipma table is used internally.
ResUsageScpu	Individual CPU/node	The system logs these tables at the node interval. These tables record CPU utilization by CPU, detailing the idle, wait, OS, and user execution.
ResUsageSvpr ResUsageIvpr	Individual vproc (AMPs and PEs) Internal vproc	ResUsageSvpr logs one record/log interval for each vproc in the system including PEs, AMPs, and node vprocs. They record: • CPU utilization by processor partition • Process pending • Memory allocations (perm data, Cylinder Index [CI], spool data, Transient Journal [TJ]) in size, aging frozen and so on • Node backup activity • BYNET activity • Disk activity • Cylinder management overhead (migrates, new cylinders, cyl packs, defragmentations). The Ivpr table is used internally.
ResUsageSldv	Logical device I/O	The system logs this table at the node interval. This table records data, including reads, writes, and response times, for logical devices, including SCSI controller activity per node and Pdisk activity.

Table	Type of Information	Description of Contents
ResUsageShst	Host traffic	The system logs this table at the node interval. This table records channel and LAN host data, including traffic in number and size.
ResUsageSobj		This table is for database object information, and is not implemented.

Kinds of ResUsage Data

The system gathers ResUsage information in four kinds, depending on the nature of the data being collected:

- Counted, or the number of times an event happens. The live buffer is updated at each event. For example, a disk write.
- Countshift, a special case of count data in which the gathering code counts the elements of a specified byte size in a power of 2. The resulting count is shifted the appropriate number of bits to convert it into a larger grain.

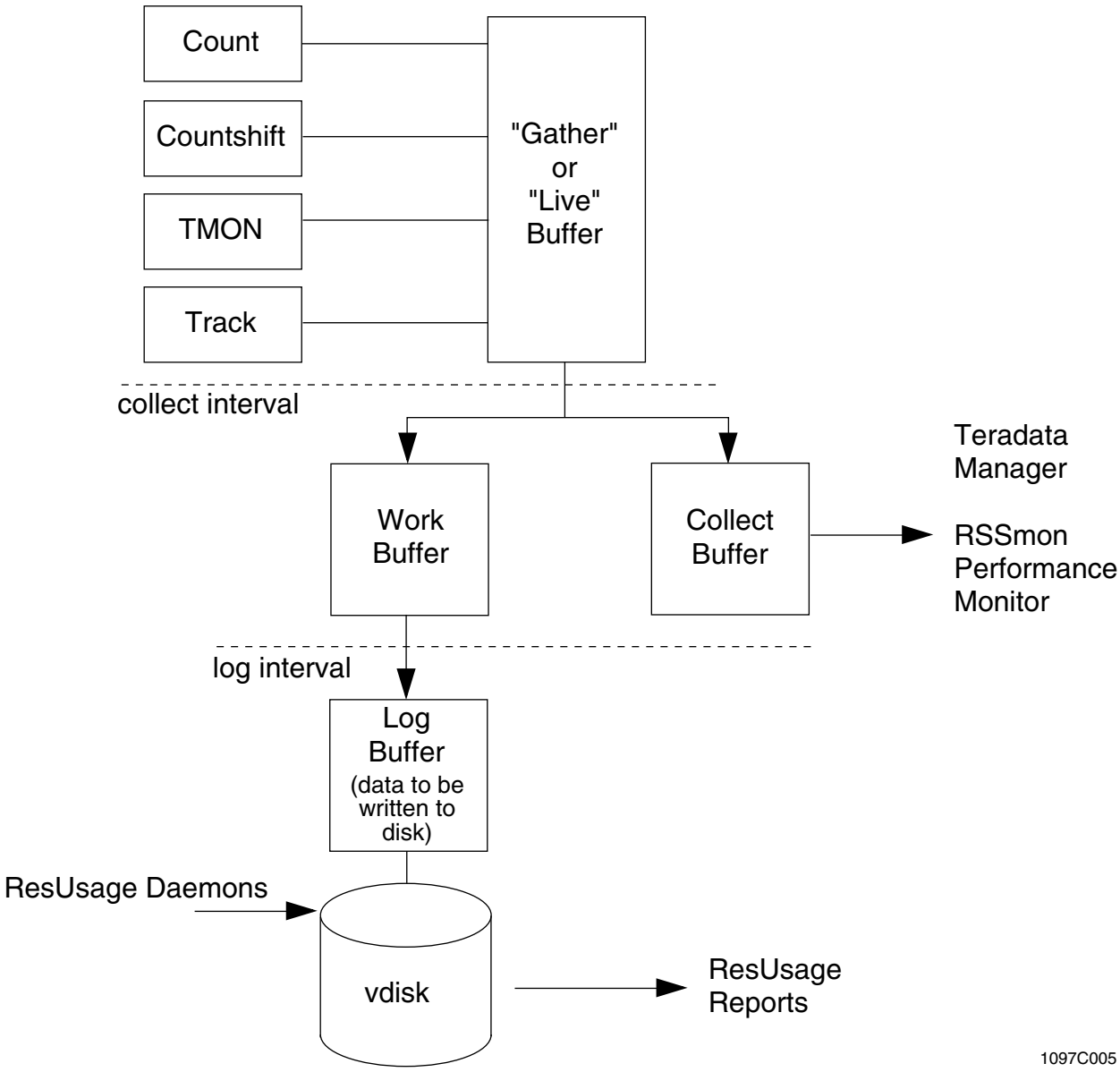
  For example, the message subsystem may measure message size by tallying 16-byte grains, and RSS (Resource Sampling Subsystem) might convert the accumulated value to kilobytes by shifting it right by 6 bits.
- Time-monitored (TMON), or how much time was spent in a particular state. The live buffer is updated at each state change.
- Tracked, that is, using a snapshot of a queue length at the collect period.

  For example, at the time of the collect, the number of memory texts resident. This information goes directly into the collect buffer.

Populating ResUsage Tables

The system populates in ResUsage tables as follows:

Stage	Process
1	The system stores counted and TMON events in the live buffer as events occur.
2	At the collect interval (that you specify), the system moves the data to the collect buffer. The system joins this data with tracked data.
3	The system summarizes data from the collect buffer, and moves the data into the work buffer. The work buffer accumulates this data until the log interval, at which time the data moves into the log buffer.
4	Data is stored in the log buffer until the system can write it to the vdisks, in the ResUsage tables that you specify.


Count
Countshift
TMON
Track
"Gather"
or
"Live"
Buffer
collect interval
Work
Buffer
Collect
Buffer
Teradata
Manager
RSSmon
Performance
Monitor
log interval
Log
Buffer
(data to be
written to
disk)
ResUsage Daemons
vdisk
ResUsage
Reports
1097C005

ResUsage Views

Each row in the ResUsage tables represents activity during one logging period. The same is true of each row in the views.

ResUsage includes the following views:

This view...	Accesses this table...	And provides...
ResGeneralInfoView	Spma	CPUs, disks, and BYNET information.
ResCPUUsageByAMPView	Svpr	details on the ways AMPS use the CPUs.
ResCPUUsageByPEView	Svpr	details on the ways PEs use the CPUs.
ResShstGroupView	Shst	details on ways host channels and LANs communicate with the RDBMS
ResSldvGroupView	Sldv	logical device information

ResUsage Macros

You can use ResUsage macros to generate reports based on ResUsage table data. Each macro provides a report on one table. Run macros after collecting statistics on a specific job or set of jobs.

This macro...	Accesses this table...	And provides...
ResCPUByAMP	SVPR	details on CPU utilization for each AMP vproc.
ResCPUByPE	SVPR	details on CPU utilization for each PE vproc.
ResCPUByNode	SPMA	details on CPU utilization for each node.
ResHostByLink	SHST	host traffic for each individual communication link.
ResLDVByNode	SLDV	a summary of all logical device activity for each node.
ResMemMgmtByNode	SPMA	memory management activity information for the node, including memory allocation and paging/swapping.
ResNETByNode	SPMA	information about BYNET traffic for the node.
ResNode	SPMA	a summary of ResUsage for the node. The report includes statistics about: • CPUusage • Logical device interface • Memory interface • Host interface • Net interface • General node process scheduling • Lock blocks encountered

Executing the Macros

To execute the ResUsage macros, see *Teradata RDBMS Resource Usage Macros and Tables*.

For example, executing the following statement

```
EXECUTE ResCPUByAmp('2001-12-25','2001-12-31', 080000,
240000,'123-02','125-04');
```

produces a report for the period beginning 8:00 a.m. on December 25, 2001 and ending 12:00 p.m., midnight, on December 31, 2001. It includes data for nodes 123-02 through 125-04.

```
01/07/12                                  CPU USAGE BY AMP                                  Page   1

                                                      Awt      Misc      Awt      Misc     Total     Total     Total
                          Vproc    Node      Node     User      User     User      User      User      User      Busy
      Date      Time       Id      Id        CPUs     Serv%     Serv%    Exec%     Exec%     Serv%     Exec%       %
--------  --------  -----  ------  --------  -------  -------  -------  -------  -------  -------  -------
01/07/12  09:57:00      0  001-01         4    0.36%    0.00%    0.05%    0.00%    0.36%    0.05%    0.41%
                        1  001-01         4    0.26%    0.00%    0.12%    0.00%    0.30%    0.12%    0.42%

          09:57:20      0  001-01         4    0.41%    0.00%    0.12%    0.00%    0.45%    0.12%    0.58%
                        1  001-01         4    0.34%    0.00%    0.05%    0.00%    0.38%    0.05%    0.42%

          09:57:40      0  001-01         4    0.25%    0.00%    0.18%    0.00%    0.28%    0.18%    0.45%
                        1  001-01         4    0.19%    0.00%    0.06%    0.00%    0.29%    0.06%    0.35%

          09:58:00      0  001-01         4    0.38%    0.00%    0.08%    0.00%    0.45%    0.08%    0.52%
                        1  001-01         4    0.31%    0.00%    0.09%    0.00%    0.34%    0.09%    0.42%

          09:58:20      0  001-01         4    0.31%    0.00%    0.08%    0.00%    0.34%    0.08%    0.41%
                        1  001-01         4    0.36%    0.00%    0.09%    0.00%    0.40%    0.09%    0.49%

          09:58:40      0  001-01         4    0.39%    0.00%    0.11%    0.00%    0.41%    0.11%    0.52%
                        1  001-01         4    0.32%    0.00%    0.12%    0.00%    0.36%    0.12%    0.49%

          09:59:00      0  001-01         4    0.29%    0.00%    0.11%    0.00%    0.30%    0.11%    0.41%
                        1  001-01         4    0.21%    0.00%    0.09%    0.00%    0.22%    0.09%    0.31%

          09:59:20      0  001-01         4    0.30%    0.00%    0.06%    0.00%    0.31%    0.06%    0.38%
                        1  001-01         4    0.30%    0.00%    0.19%    0.00%    0.32%    0.19%    0.51%

          09:59:40      0  001-01         4    0.40%    0.00%    0.09%    0.00%    0.46%    0.09%    0.55%
                        1  001-01         4    0.26%    0.00%    0.08%    0.00%    0.38%    0.08%    0.45%

          10:00:00      0  001-01         4    0.32%    0.00%    0.08%    0.00%    0.34%    0.08%    0.41%
                        1  001-01         4    0.28%    0.00%    0.09%    0.00%    0.31%    0.09%    0.40%
```

Column...	Reports the percent of time AMPs were busy doing user...
Awt User Serv%	service for the AMP Worker Task (Awt) partition.
Misc User Serv%	service for miscellaneous (all other except Partition 0) AMP partitions.
Awt User Exec%	execution within the AMP Worker Task (Awt) partition.

Column...	Reports the percent of time AMPs were busy doing user...
Misc User Exec%	execution within miscellaneous (all other except Partition 0) AMP partitions.
Total User Serv%	service[a] work. This is the sum of the Awt User Serv%, the Misc User Serv%, and AMP Partition 0 user service%.
Total User Exec%	execution[b] work. This is the sum of the Awt User Exec%, the Misc User Exec%, and AMP Partition 0 user execution.
Total Busy%	service and execution work. This is the sum of the Total User Serv% and the Total User Exec% columns.

a. Service is the time that a CPU is busy executing user service code, which is privileged work performing system-level services on behalf of user execution processes that do not have root privileges.

b. Execution is the time a CPU is busy executing user execution code, which is the time spent in a user state on behalf of a process.

Collect and Log Rates

Introduction

You must set the collect and log rates to enable RSS to collect data and log it into the ResUsage tables.

IF the setting is...	AND the rate is...	THEN the system...
600 seconds (default)	RSS Collect	collects statistics for the node and for each vproc at this interval.
	Node Log	writes collected statistics to the database for the node at this interval.
	Vproc Log	writes collected statistics to the database for each vproc at this interval.

Guidelines

Follow these guidelines to set the collect and log rates:

- The ranges for all settings are 0 to 3600 seconds.
- The logging rates must be an even multiple of the RSS collect rate.
- The collect rate must be less than or equal to the log rate.
- To disable logging of the data to the database, either deselect the tables or set the log rates to zero.

For information on setting collect and log rates, see *Teradata RDBMS Resource Usage Macros and Tables*.

ResUsage Disk Space Overhead Considerations

Some overhead is associated with turning on ResUsage logging as rows are written to the ResUsage tables. At a log rate of 600 (seconds, which is the default rate), the following numbers show the approximate system overhead for each ResUsage table.

Table	# Columns	Bytes/Row (approx)	1 Row Per	# Rows/ Node	KB/Hour/ Node *(1)	MB/Month/ Node *(1)
SPMA	195	1520	Node	1	18	5
IPMA	98	750	Node	1	9	6
SCPU	25	210	CPU *(2)	4	10	7
SVPR	357	2550	2 * Vproc *(3)	24	387	272
IVPR	184	1520	Vproc *(3)	12	214	150
SLDV	26	240	Ldv-id *(4)	32 x 4 = 128	90 x 4 = 360	63 x 4 = 253
SHST	61	490	*(5)	5	29	20

Notes:

1. Assume Fallback on all ResUsage tables, and 10 minute logging intervals.
2. Assume four CPUs/node.
3. Assume eight AMPs and three PEs/node. Additionally, there is one node vproc/node.
4. Logical device/node varies with # ranks/vproc and disk size; here, assume 1 rank/virtual AMP and 4 LUNs/rank. Logical device data is recorded in every node in the clique. Assuming four nodes in a clique, logical device data is recorded four times for the same logical device.
5. Two rows for each channel connect, one row for each network connect.

Typically, you always want ResUsageSPMA turned on. You can turn other tables on and off as necessary.

Optimizing ResUsage Logging

Follow the procedure below to optimize the performance and reduce the cost of ResUsage logging on Teradata:

Step	Action
1	Select the summary mode in the xctl utility RSS Settings window to reduce database I/O operations by consolidating and summarizing ResUsage data. But note that the trade-off is less detail in the output report.
2	If ResUsage logging terminates due to a lack of table space: **a** Delete rows from the appropriate table or make more space for it in user DBC **b** Restart ResUsage logging via xctl settings, Teradata Manager, or the DBW SET RESOURCE command in reclaim rows written to the database. If space is never reclaimed, it will eventually grow to consume all available space in user DBC.
3	To minimize extra overhead: • Never enable logging on tables that you do not intend to use. • Always disable ResUsage logging or decrease the collection rate when you do not need it. Writing to the database adds to system I/O load. On a heavily loaded system, this could affect the production workload throughput. To disable all ResUsage logging, either disable all tables or set both logging rates to zero. If problems arise, enable other tables and increase frequency only as needed.
4	Do not use small logging rates. Instead, use the largest rates that provide enough detailed information for your purposes. Generally, use a logging rate no smaller than 60. If logging is enabled on all the ResUsage tables, set logging rates to a value no smaller than 300.

Step	Action
5	Initially, set all 4 collection and logging rates to 600. You can adjust these values any time, even when the database system is busy. New values take effect as soon as the adjustment command is issued.

6	Periodically, purge old data from ResUsage tables.
	
	Each day, for example, during low system activity, execute the following Structured Query Language (SQL) statement to remove data more than thirty days old from the Spma table:
	
	`DELETE FROM ResUsage Spma`
	`WHERE TheDate < DATE - 30;`

Note: In an extremely loaded system, RSS can fall behind in writing data to the database. Although it caches such data, and will eventually catch up if given a chance, the RSS will be forced to start discarding rows if the system load persists and its cache capacity has been exceeded.

ResUsage and Teradata Manager Compared

ResUsage Advantages

ResUsage data has the following advantages:

- The ResUsage reports are more comprehensive than data Teradata Manager displays.

  For example, BYNET data overruns and high-speed Random Access Memory (RAM) failures are not reported in Teradata Manager.
- The system writes ResUsage data to tables. The system does not write Teradata Manager data to tables; therefore, the resource usage data you do not examine at the end of the sample interval is lost and overwritten.
- Because of the historical nature of ResUsage data (that is, a large amount of data accumulated over a long period of time), it is best used for the following:
  - Determining trends and patterns
  - Planning system upgrades
  - Deciding when to add new applications to systems already heavily utilized

ResUsage Disadvantages

ResUsage data has the following disadvantages:

- ResUsage reports are less convenient and less user-friendly for real-time on-the-fly analysis than data Teradata Manager displays.
- ResUsage reports do not provide session-level resource usage data, application locks data, and data on the application being blocked.

ResUsage and DBC.AMPUsage View Compared

The DBC.AMPUsage view displays CPU usage information differently than that way such information is displayed in ResUsage data.

This facility...	Provides...
ResUsage data	metrics on the whole system, without making distinctions by individual user or account ID.
DBC.AMPUsage view	AMP usage by individual users or account ID.

See "Teradata Manager and DBC.AMPUsage View Compared" on page 11-22.
For more information on the DBC.AMPUsage view, see *Teradata RDBMS Data Dictionary*.

Extended ResUsage Macros and UNIX Performance

This section provides an overview of extended ResUsage macros to help you troubleshoot UNIX performance.

Macro	Macro Variable(s)	Report Column/Graph	Problem
Res5100Cpu Res5150Cpu		Avg & Max AMP CPU Utilization	High AMP skewing
ResCylPack	• HourTotal • Total	Splits & Migrates	FreeSpacePercent (FSP) may need adjustment.
ResNet	• HourTotal • Total • BySec	Backup Reads	Updates/inserts processing workload. When point-to-point exceeds backups by more than 2 to 1, the excess is row redistribution for joins and aggregations.
		I/O Wait vs. BYNET I/O	BYNET I/O is/is not/may be the cause of the I/O wait. Compare I/O Wait against broadcast or point-to-point or, on a UNIX system, against buddy backup.

Macro	Macro Variable(s)	Report Column/Graph	Problem
ResPMA	• HourTotal • Total • BySec	Available Free Memory	• Too much/too little available free memory. Tune FSG Cache Percent (see Chapter 10). • Minimum often goes below 40 MB. Tune UNIX LOTSFREE, DESFREE, and MINFREE parameters (see "Adjusting for Low Available Free Memory" on page 10-5). • Difference between average and minimum. • Available free memory. A non-Trusted Parallel Application (TPA) is running.
		CPU vs. I/O Wait	Significant I/O bottleneck exists. Either the system is I/O bound or not enough tasks are running in the system.
		Disk Read & Writes	Disk pre-reads indicate full table scans. Disk writes indicate spool or data table updates.
		Page/Swap	High values for extended periods. Possibly tune FSG Cache Percent. Tune LOTSFREE, DESFREE, and MINFREE (see "Adjusting for Low Available Free Memory" on page 10-5).
		Parallel Efficiency	• Nodes (CPUs) at maximum efficiency (out of capacity) • Skewing • Periodic workload (workload profile)
		I/O Wait vs. Disk I/O	Disk I/O is/is not/may be the cause of the I/O wait. Bottleneck could be BYNET I/Os or a combination of Disk and BYNET I/Os.
ResSvprRead	• HourTotal • Total	Types of I/O	Table data, spool, TJ and Permanent Journal (PJ) reads.

For more information on using ResUsage macros to troubleshoot performance, see:

- "ResUsage and Host Traffic" on page 2-20
- "ResUsage and CPU Utilization" on page 2-22
- "ResUsage and Disk Utilization" on page 2-31
- "ResUsage and BYNET Data" on page 2-34

ResUsage and Host Traffic

You can use the following macros to analyze the traffic flow between the Teradata database and the channel or network client.

Macro	Description	Purpose
ResHostHourTotal	Hourly totals	Check workloads over a weekly period (extended set)
ResHostTotal	Log period totals	Problem analysis of daily workloads (extended set)
ResHostByLink	By second averages	General analysis

These macros contain the following host I/O columns:

Columns	Description
Host Type	Type of host link • Network • Mainframe channel connect
Host Id	ID of host
MBs (KBs) Read	MB (KB) read from host
MBs (KBs) Write	MB (KB) written to host
Blks Read	Number of I/O blocks read from host
Blks Write	Number of I/O blocks written to host
KBs/Blk Read	Average block size per read I/O from host
KBs/Blk Write	Average block size per write I/O to host

Host Communications by Communication Link

To understand how to use ResHost, the next table provides information to help you examine macro output reflecting three different types of workloads. Normal DSS activity = no loads and approximately 10KB reads or writes.

Workload # 1	Workload # 2	Workload # 3
> 100 KBs read/sec	>100 KBs write/sec Divide by 1024 to get the true number for network host types	KBs read/write = 10 to 19 and # Blks read/ write 30 to 70
FastLoad MultiLoad	Basic Teradata Query (BTEQ) answer set return (one host connect) Fast Export with high volume of data (in parallel through multiple host connects)	Primary Index (PI) workload

Following is sample output based on the above workloads.

Wrkld	Time	Node Id	Vproc Id	Host Type	Host Id	KBs Read /Sec	KBs Write /Sec	Blks Read /Sec	Blks Write /Sec
#1	18:34	018-00	1018	IBMMUX	9	130.0	0.2	12.8	4.3
	18:35	018-00	1018	IBMMUX	9	136.5	0.2	13.5	4.5
	18:36	018-00	1018	IBMMUX	9	135.5	0.2	13.4	4.5
	18:37	018-00	1018	IBMMUX	9	113.1	0.2	11.2	3.8
#2	13:05	019-00	65535	NETWORK	0	1065	151,349	18.4	18.4
	13:06	019-00	65535	NETWORK	0	1063	150,940	18.3	18.3
	13:07	019-00	65535	NETWORK	0	1071	152,314	18.5	18.5
	13:08	019-00	65535	NETWORK	0	1055	149,702	18.2	18.2
#3	22:42	018-00	1018	IBMMUX	9	10	14	48.0	44.2
		02	65535	IBMMUX	9	11	14	43.4	35.9
		019-00	65535	IBMMUX	5	11	15	55.1	32.1
		019-02	65535	IBMMUX	5	11	15	46.7	33.8
	22:43	018-00	1018	IBMMUX	9	14	19	64.5	57.0
		018-02	65535	IBMMUX	9	14	19	62.7	48.6
		019-00	65535	IBMMUX	5	15	19	72.1	46.5
		019-02	65535	IBMMUX	5	15	19	63.1	45.3
	22:44	018-00	1018	IBMMUX	9	8	11	35.3	36.1
		018-02	65535	IBMMUX	9	8	11	34.6	31.8
		019-00	65535	IBMMUX	5	8	11	40.0	31.4
		019-02	65535	IBMMUX	5	8	11	34.3	29.0

ResUsage and CPU Utilization

Teradata is designed to give users all of the system resources they need. This is different from a typical timesharing environment where the users are limited to a maximum CPU utilization threshold that may be very small depending on predefined user privileges.

CPU Busyness

The following macros report CPU busyness:

Macro	Description	Purpose
ResNode	(System provided) by second averages	General system analysis
ResPmaCpuDayTotal	Daily totals	Capacity planning and tracking long term trends
ResPmaHourTotal	Hourly totals	Check workloads over a weekly period
ResPmaTotal	Log period totals	Problem analysis of daily workloads
ResPmaBySec	By second averages	Detailed problem analysis
ResPmaByNode	By node details	Node-level problem analysis

The above macros contain the Avg CPU Busy column, which is the average CPU utilization for all nodes. Avg CPU Busy % is a measure of how often multiple CPUs in a node were busy during a log period.

- In a DSS environment, a small number of jobs can easily bring the CPU close to 100% utilization.
- High CPU utilization consists of a Avg CPU Busy % of 70% over significant periods of time.
- The CPU is stressed differently in DSS and transaction processing environments.

DSS Environment

The following table lists how CPU tasks are carried out on the node during DSS operations.

Stage	Process
1	Prepare for read: **a** Memory management allocates memory for the data block. **b** Database software communicates with the file system. **c** File system communicates with the disk controller.
2	Qualify rows. Determine if the row satisfies the WHERE clause condition(s). Most DSS operations require full-table scans in which the WHERE clause condition check is relatively time-consuming. Full-table scans generally result from SQL statements whose WHERE clause does not provide a value for an index.
3	Process rows: • Join • Sort • Aggregate
4	Format qualifying rows for spool output.

Transaction Processing Environment

Transaction processing is:

- Table maintenance, such as UPDATE, INSERT, MERGE, and DELETE, or SELECT by PI value
- Access by UPI, NUPI, or USI

  Updates and deletes are usually indexed operations with access by a Unique Primary Index (UPI), Non-Unique Primary Index (NUPI), or Unique Secondary Index (USI). Non-unique Secondary Indexes (NUSIs) are not suitable, particularly in on-line transaction processing, because they are not node-local and require all nodes to be involved in the search for the row.

  UPI and NUPI accesses are one-node operations. USI accesses are two-node operations. In addition to updates, inserts and deletes also are common in the PI access environment.
- OLTP or batch maintenance
- Small amounts of data access at one time
- Frequent one-row requests via PI selects

The following table describes how the CPU tasks are carried out on the node during a transaction processing activity. The table also applies to batch maintenance processing.

Notice that the qualify rows activity is missing from the table. In transaction processing, it is more common for the WHERE clause to provide a value for the PI or USI. The read itself qualifies rows. Transaction processing typically avoids

further conditional checks against non-indexed columns. All of these CPU tasks occur on the nodes.

Stage	Process
1	Prepare for read: **a** Memory management allocates memory for the data block. **b** Database communicates with the file system. **c** File system communicates with the disk controller.
2	Update row: **a** Database locates row to be updated. **b** Memory management allocates memory for the new data block to be built. **c** Database updates the changed row and copies the old rows. **d** Database communicates with the file system. **e** File system communicates with the disk controller.

Parallel Efficiency

Node parallel efficiency is a measure of how evenly the workload is shared among the nodes. The more evenly the nodes are utilized, the higher the parallel efficiency.

Node parallel efficiency is calculated by dividing average node utilization by maximum node utilization. Parallel efficiency does not consider the heaviness of the workload. It only looks at how evenly the nodes share that workload.

The closer node parallel efficiency is to 100%, the better the nodes work together. When the percentage falls below 100, one or a few nodes are working much harder than the others in the time period. If node parallel efficiency is below 60% for more than 1 or 2 10-minute log periods, Teradata is not getting the best performance from the parallel architecture.

Possible causes of poor parallel node efficiency include:

- Down node
- Uneven number of AMPs per node
- Skewed table distribution
- Skewed join or aggregation processing
- Non-Teradata application running on a TPA node
- Coexistence system with different speed nodes

Poor parallel efficiency can also occur at the AMP level.

Common causes of poor parallel efficiency include:

- Poor table distribution (you can check this in DBC.Tablesize)
- Skewed processing of an SQL statement

– User CPU (you can check this in DBC.AMPusage)
– Spool (you can check this in DBC.Diskspace)

The following table lists common mistakes that can cause skewing and poor parallel efficiency, and solutions.

Mistake	Solution
A user did not define a PI. The system uses the first column of the table as the default P1.	Define a PI with good data distribution.
A user used a null value as PI for the target table in a left outer join.	Perform any of the following: • Choose a different PI. • Handle NULL case separately. • Use a multi-set table.

A user performed a join on column with poor data distribution. For example, the user entered:
`SELECT A.colname, B.x, B.y`
`FROM A, B`
`WHERE A.colname = B.colname;`
	• Identify column value and counts. For example, enter:
	`SELECT colname, count(*)`
	`FROM T`
	`HAVING count(*) > 1000`
	`GROUP BY 1 ORDER BY 1 Desc;`
	The following displays:
	`colnamecount(*)`
	`codeXYZ720,000`
	`codeABC1,200`
	• Break the query into two separate SQL statements.For example, handle codeXYZ only in 1 SQL statement; handle all other cases in another SQL statement.
	• Collect statistics on the join column.

CPU Use

The following macros provide information on CPU use:

Macro	Table	Purpose	See
ResCPUByNode	SPMA	Report of how each individual node is utilizing CPUs	following subsection
ResCPUByAMP	SVPR	Report of how each AMP utilizes the CPUs on the respective node	page 2-27
ResCPUByPE	SVPR	Report of how each Parsing Engine (PE) utilizes the CPUs on irrespective node	page 2-26

Macro	Table	Purpose	See
ResCpuByCpu	SCPU	Report of how each individual CPU is executing within a node (Expanded set)	page 2-28
ResSvpr5100Cpu/ ResSvpr5150Cpu	SVPR	Summary report of how all PEs and AMPs utilize CPUs throughout the whole system (Expanded set)	page 2-29

ResCPUByNode

The ResCPUByNode macro selects data from the ResUsageSpma table. You must enable logging to the ResUsageSpma table to use this macro.

The following columns are the averages for all CPUs on the node.

This column...	Lists percentage of time spent...
I/O Wait %	idle and waiting for I/O. The I/O wait time is time waiting for disk, BYNET, or any other I/O device.
Total User Serv %	performing user service work.
Total User Exec %	performing user execution work.
Total Busy %	performing either service or execution work. Sum of the Total User Serv % and Total User Exec % columns.

where:

This variable...	Describes the time a CPU is busy executing...
User service	user service code, which is privileged work performing system-level services on behalf of user execution process that do not have root privileges.
User execution	user execution code, which is the time spent in a user state on behalf of a process.

ResCPUByPE

The ResCPUByPE macro normalizes data from the ResusageSvpr table. You must enable logging on this table to use the ResCPUByPE macro.

CPU statistics in this macro represent the aggregate of all time spent by all CPUs in the node. Because there are multiple CPUs, the theoretical maximum percent is the 100 times the number of CPUs in the node. In most cases (4700/5150/4400), the maximum Total Busy % for 4 CPUs would be 100%.

The following table describes the ResCPUByPE macro columns.

Column	Description
Pars User Serv%	Service for the parser partition of the PE.
Disp User Serv%	Service for the dispatcher partition of the PE.
Ses User Serv%	Service for the session control partition of the PE.
Misc User Serv%	Service for miscellaneous (not classified as parser, dispatcher, session control, or partition 0) PE partitions.
Pars User Exec%	Execution within the parser partition of the PE.
Disp User Exec %	Execution within the dispatcher partition of the PE.
Ses User Exec %	Execution within the session control partition of the PE.
Misc User Exec %	Execution within the miscellaneous partition of the PE.
Total User Serv%	Total service work. The sum of the user service columns above plus PE partition 0 user service.
Total User Exec %	Total execution work. The sum of the user execution columns above plus PE partition 0 user execution.
Total Busy %	Total service and execution work. The sum of the Total User Serv% and Total User Exec % columns.

ResCPUByAMP

The ResCPUByAMP macro normalizes data from the ResusageSvpr table. You must enable logging on the ResusageSvpr table to use this macro.

The CPU statistics in this macro represent the aggregate of all time spent by all CPUs in the node. Because there are multiple CPUs, the theoretical maximum percent is 100 times the number of CPUs in the node. In most cases (4700/5150/4400), the maximum Total Busy % for 4 CPUs would be 100%.

The following describes the ResCPUByAMP macro columns.

Column	Description
Awt User Serv%	Service for the AMP Worker Task (AWT) partition.
Misc User serv%	Service for miscellaneous (not classified as AWT or Partition 0) AMP partitions.
Awt User Exec%	Execution within the AWT partition.
Misc User Exec%	Execution within miscellaneous AMP partitions.
Total User Serv%	Total service work. The sum of the Awt User Serv%, Misc User Serv%, and AMP Partition 0 user service %.

Column	Description
Total User Exec%	Total execution work. The sum of the Awt User Exec%, Misc User Exec%, and AMP Partition 0 user execution%.
Total Busy%	Service and execution work. The sum of the Total User Serv% and Total User Exec% columns

ResCpuByCpu

The ResCpuByCpu macro accesses the ResusageSCPU table to report the busyness of each CPU within a node (the CPU parallel efficiency of a node). The ResCPUByCPU macro displays the following columns for each CPU/node.

Column	Description
Total Busy %	Sum of Total User Serv% and Total User Exec %
I/O Wait %	Waiting for I/O
Total User Serv %	Execution of system services
Total User Exec %	Execution of database code
CPU Eff %	Average busy of all CPUs in a node divided by max busy CPU

Run this macro to verify that the system is using all CPUs. The ResusageSCPU table is not usually turned on; however, you should turn it on to flag the condition of a single CPU in a node doing most of the work in that node. A specific occurrence: when a system algorithm causes the work to go to a few CPUs while some remained idle. Use this macro if all the following are true:

- ResNode macro shows your system never reaches 100% busy even under the heaviest user workload.
- You do not have a hardware problem.
- You do not have an I/O bottleneck.

Following is sample output of the ResCpuByCpu macro:

```
Res      Res     Res    CPU    Total    CPU I/O  Total   Total
Scpu     PMA     CPU    Effic  Busy     Wait     Hser    User
Time     ID      Id     #      #        #        Serv%   Exec%
17:24    18-02   0             92.4     2.3      77.3    15.1
                 1             81.6     4.5      67.4    14.2
                 2             93.2     2.8      76.8    16.4
                 3             89.2     3.8      68.5    20.7
                 4             83.6     3.2      69.1    14.5
                 5             93.8     2.5      80.2    13.6
                 6             90.5     2.6      74.6    15.9
                 7             80.3     3.2      66.4    13.9
                               ------
                               93.9%
```

ResSvpr5100Cpu/ResSvpr5150Cpu

Keep the following in mind for the ResSvpr5100Cpu and ResSvpr5150Cpu macros. For AMP Virtual Processors (vprocs):

vproc Type	Considerations
AMP	• In a node, an AMP does NOT have exclusive use of a CPU. • An AMP can run on any CPU in a node. • Any number of AMPs can run on a given node. • Hash buckets are assigned to AMPs. • Portions of disk (slices) are assigned to an AMP to become the Virtual Disk (vdisk). • A vproc is always associated with a given node (except in the case of vproc migration) and the installation controls this association through pdeconfig.
PE	• In a node, a PE does NOT have exclusive use of a CPU. • A PE can run on any CPU in a node. • Any number of PEs can run on a given node. • A PE can run in the same node as the AMPs. • Local Area Network (LAN) PEs will migrate if the node goes down; channel PEs will not migrate.

The ResSvpr5100Cpu macro shows the average and maximum AMP utilizations on an 8 CPU/node system.

The ResSvpr5150Cpu macro shows the average and maximum AMP utilizations on a 4 CPU/node system. This macro works for the 4700, 4800 and 5200.

These macros provide a vproc summary report. Because the number of CPUs is not recorded in the table, the number is hard coded directly inside the macros.

The following table describes ResSvpr5100Cpu/ResSvpr5150Cpu macro columns.

Column	Description
Avg # AMPs/Node	Average number of virtual AMPs/node
Avg # PEs/Node	Average number of virtual PEs/node
Avg CPU%/Node	Average CPU utilization by all vprocs in a node
Avg AMP %/Node	Average CPU utilization by all Virtual AMPs in a node
Avg PE %/Node	Average CPU utilization by all Virtual PEs in a node
Avg NVpr %/Node	Average CPU utilization by the node vproc in a node

Column	Description
Avg AMP%/CPU	Average AMP vproc utilization for all CPUs
Max AMP%/CPU	Maximum AMP vproc utilization for all CPUs
Avg PE %/CPU	Average PE vproc utilization for all CPUs
Max PE %/CPU	Maximum PE vproc utilization for all CPUs
Avg NVpr %/CPU	Average node vproc utilization for all CPUs
Max NVpr %/CPU	Maximum node vproc utilization for all CPUs

A node vproc handles operating system functions not related directly to AMP or PE work. These functions include the disk I/O and BYNET I/O drivers.

Following is sample ResSvpr5100Cpu/ResSvpr5150Cpu output:

Date	Time	Avg # AMPs /Node	Avg # PEs /Node	Avg CPU% /Node	Avg AMP% /Node	Avg PE% /Node	AVG NVpr% /Node	Avg AMP% /CPU	Max AMP% /CPU	Avg PE% /CPU	Max PE% /CPU	AVG NVpr% /CPU	MAX NVpr% /CPU
99/06/01	15:00	6	2	19	18	0	1	24	43	0	2	7	92
99/06/01	15:10	6	2	47	46	0	1	61	78	1	24	11	94
99/06/01	15:20	6	2	58	57	0	2	75	115	0	2	13	90
99/06/01	15:30	6	2	80	76	0	4	102	139	1	9	30	36
99/06/01	15:40	6	2	85	81	0	3	108	175	1	30	25	30
99/06/01	15:50	6	2	82	80	0	2	106	147	1	31	19	22
99/06/01	16:00	6	2	85	83	0	2	111	180	1	25	18	21
99/06/01	16:10	6	2	84	82	0	3	109	159	1	3	21	26
99/06/01	16:20	6	2	58	57	0	2	75	173	0	4	13	87
99/06/01	16:30	6	2	62	60	0	2	80	125	1	6	15	73
99/06/01	16:40	6	2	60	59	0	1	78	136	1	7	11	71
99/06/01	16:50	6	2	68	65	0	2	87	110	1	22	20	59

ResUsage and Disk Utilization

The following macros report disk utilization:

Macro	Description	Purpose
ResNode	(System provided) by second averages	General system analysis
ResIODayTotal	Daily totals	Capacity planning and tracking long term trends
ResPmaHourTotal	Hourly totals	Checking workloads over a weekly period
ResPmaTotal	Log period totals	Problem analysis of daily workloads
ResPmaBySec	By second averages	Detailed problem analysis
ResPmaByNode	By node details	Node-level problem analysis

Disk I/O Columns

All disk I/O columns in this subsection are physical I/Os.

Column	Description
Position Rds	Full tables scans use position reads to position to the first data block on the cylinder, while the pre-reads are for sequential full table scans of table blocks and spool blocks. If there are no pre-reads, the system is performing a full table scan operation. Transaction processing uses position reads to read individual table data blocks, TJ blocks, PJ blocks, and Cylinder Indexes (CIs) for the transaction.
Pre-Rds	
Total Disk Rds	Position reads and pre-reads total the number of disk reads.
DB Wrts	Average number of writes to disk per second. Disk writes can be table data blocks, spool blocks, CIs, and TJ and PJ blocks.
Disk RdMB(KB)	MB (KB) transferred for disk reads, including both position reads and pre-reads.
Disk WrtMB(KB)	MB (KB) transferred for disk reads.
PgSw IOs	Paging and Swap I/Os per second. If the swap I/O count is high, you can adjust the amount of free memory by reducing FSG CACHE PERCENT via the xctl utility.

The following table shows the block size for disk reads and writes for the system:

Block Type	Size (KB)	Comments
CI	8	
TJ	4	Tunable parameter
PJ	4	Tunable parameter
Permanent tables	63.5	Tunable parameter and option in the CREATE TABLE statement
Spool tables	63.5	

Table data blocks are NOT a fixed size:

Stage	Process
1	Table data blocks start at maximum data block size.
2	When a row is added, the block is split into two smaller pieces.
3	As data is added over time, the data blocks will grow to the maximum size before splitting again.

I/O Wait % Column

I/O Wait % helps to identify situations where the system CPU capacity is under-utilized because it is waiting for I/O (it is I/O bound).

The system has more CPUs than disk controllers, so CPUs have to share their I/O resources. CPUs may have to wait for I/O.

When the system is I/O bound, improving CPU speed will not help your performance. Teradata may be I/O bound because of:

- BYNET activity
- Disk activity

With multiple CPUs, it is possible to saturate the I/O subsystem. For example, a site was CPU-bound with four CPUs/node. When the site upgraded to eight CPUs/node the system became I/O bound! The site solved the I/O problem by adding a second disk array controller.

Suppose a node includes two disk controllers and four CPUs. If both controllers are in use when a CPU makes an I/O transfer request, that CPU must wait until one of the controllers becomes available.

The system can go into a CPU Idle Waiting for I/O state due to:

- Disk I/O bottleneck
- BYNET I/O bottleneck

- Combination of disk and BYNET I/O
- Not enough jobs in the system

I/O Wait % Versus Disk I/O or BYNET I/O

When I/O Wait % and Disk I/O patterns match up, the I/O bottleneck is probably due to disk I/O.

When I/O Wait % and BYNET I/O patterns match up, the I/O bottleneck is probably due to BYNET I/O.

ResNode Macro Equivalent Disk I/O Columns

The following CPU-related columns in the ResNode macro have been discussed previously with slight modifications:

Column	Equivalent to
WIO %	I/O Wait %.
Ldv IO/Sec	Combined disk position reads, pre-reads and writes/ second.
Ldv Eff %	Parallel efficiency of the logical disk I/Os. It is the average number of I/Os per node divided by the number of I/Os performed by the node with the most I/Os. Similar to node parallel efficiency for CPU Busy %.
P+S % of IOs	Percentage of logical disk reads and writes that are for paging or swapping purposes.
Read % of IOs	Percentage of logical disk reads and writes that are reads.
Ldv KB/IO	Average size of a disk I/O. Includes the composite average of disk reads and writes

ResUsage and BYNET Data

The following macros report BYNET data:

Macro	Description	Purpose
ResNode	(System provided) by second averages	General system analysis
ResPmaIODayTotal	Daily totals	Capacity planning and tracking long term trends.
ResNetHourTotal	Hourly totals	Checking workloads over a weekly period
ResNetTotal	Log period totals	Problem analysis of daily workloads
ResNetBySec	By second averages	Detailed problem analysis

These macros include the following columns.

Column	Description
Date	Date of statistics
Hour	Hour of statistics
I/O Wait %	Average % of time the CPU was idle waiting for I/O
Bkup Rds	{Total, average} buddy node backup reads
PtP Rds	{Total, average} BYNET point-to-point reads
PtP Rd MB	{Total, average} point-to-point MB read
Brd Rds	{Total, average} BYNET broadcast reads
Brd Wrts	{Total, average} BYNET broadcast writes
Brd Rd MB	{Total, average} broadcast MB read
Brd Wrt MB	{Total, average} broadcast MB written
PtP + Brd Rds	{Total, average} sum of point-to-point and broadcast reads
Text Alocs	{Total, average} text (code) memory allocations in free memory
VPR Alocs	{Total, average} vproc (data) memory allocations in free memory

BYNET I/O Types

There are three types of BYNET I/O:

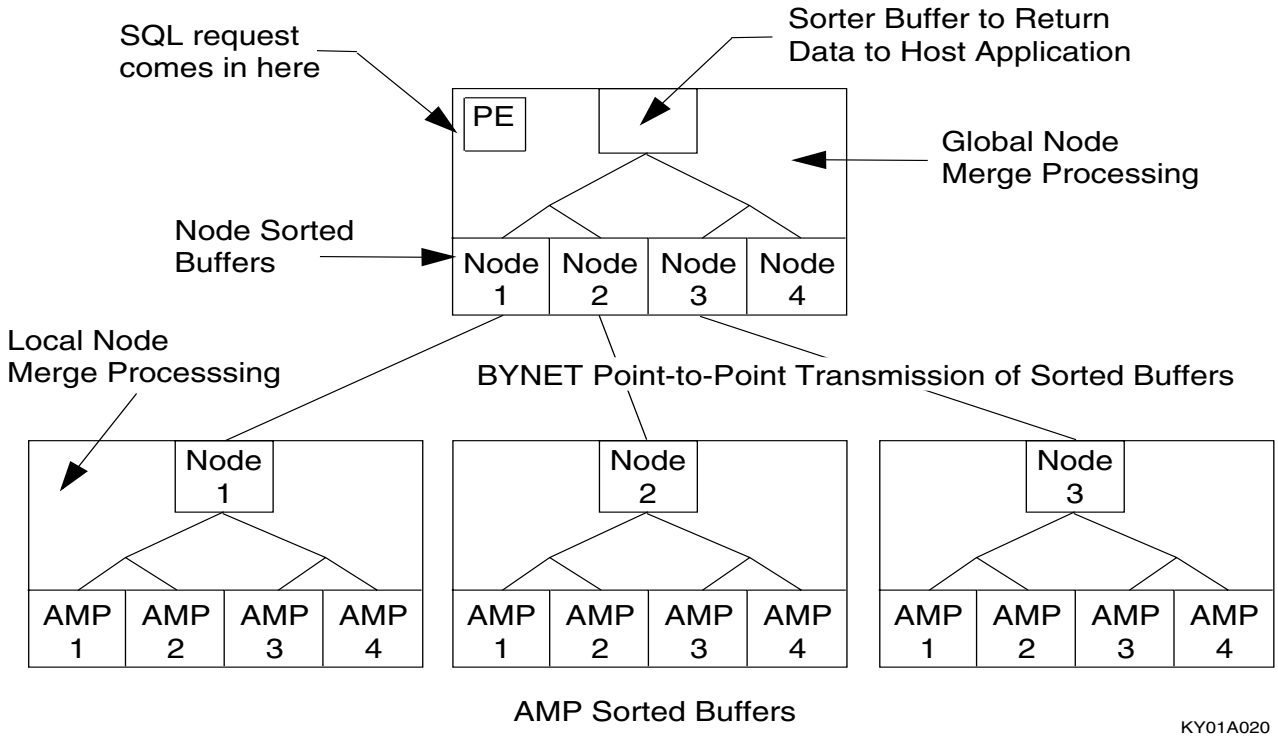
Type	Description
Point-to-point	Point-to-point messages have one sender and one receiver. The total point-to-point reads equals the total point-to-point writes. Point-to-point is used for: • Row redistribution between AMPs • Communication between PEs and AMPs on a single AMP operation Operations that cause point-to-point include: • Joins, including merge joins and exclusion merge joins, some large table/small table joins, and nested joins • Aggregation • FastLoad • Updates to fallback tables • MulitLoads • INSERT SELECTs • FastExports • Create USI • Create fallback tables • On UNIX, Buddy Backup AMP traffic • Create Referential Integrity (RI) relationship • USI access
Broadcast	Broadcast transmits a message to multiple AMPs at the same time. It is used for: • Broadcasting an all-AMP step to all AMPs • Row duplication Multicast is a special case of a broadcast where only a dynamic group of vprocs (subset of all vprocs) process the message. This avoids: • Sending point-to-point messages to many vprocs • Involving a large majority of vprocs that have nothing to do with a transaction A vproc can send a message to multiple vprocs by sending a broadcast message to all nodes. The BYNET software on the receiving node determines whether a vproc on the node should receive the message. If not, it is discarded. Teradata allows a limited number of broadcast messages if traffic is high and limited to point-to-point messages.
Merge	BYNET merge is used only for returning a single answer set of a single SQL select statement. Merge writes per second is the average number of merge rows sent to the PE by the AMP. **Note:** FastExport output is not counted in the merge statistics.

BYNET Merge Processing

BYNET merge processing involves the following steps.

Stage	Action
1	One PE, the coordinator, broadcasts the message to start the merge.
2	The virtual AMPs sort their respective spool files and create a sort key for each row.
3	Each node sorts all spools from respective vprocs, (a vproc sort/merge).
4	The node builds an intermediate buffer with sort keys and sorted rows from all virtual AMPs on the node. The sort key information identifies the node and AMP for the respective row.
5	Each node does a merge move, a point-to-point move of the first buffer to the coordinating PE.
6	The coordinator PE does a set-up. It builds: • A heap (buffer) for each node on the system • A "to be sent to host" buffer • An additional heap where it merges pointer information If there are 8 nodes on the system, the PE sets up 10 buffers: • 1 corresponding to each of the nodes on the system • 1 for sort key pointers • 1 for the data rows to be returned to the host
7	The coordinator PE sleeps until the virtual AMPs are ready.
8	The coordinator PE gets sort key/row information from all nodes and builds a tree (in the sort key pointer heap) from all keys. The coordinator PE knows how many nodes to expect, so it knows when it has all the information it needs.
9	The coordinator does a heap sort.
10	In the steady state the coordinator looks at the heap, sends a (point-to-point) request, and gets the first buffer.
11	The coordinator gets the next highest and does a sift up.

The following figure shows 16 virtual AMPs, 4 on each node. Each AMP does its own sort, and merge is done in the node. The nodes send buffers to the receiving node where buffers are processed.



BYNET Merge Output

The output in the following table is of a sort with rows returned. The output has been cropped to show times with high merge activity to very low merge activity. For space considerations, the following columns are not displayed:

- OS as % CPU
- I/O Wait %
- Total Disk RdMB
- Total Disk WrtMB
- Page Swap I/Os
- Avg MB Free
- Min MB Free

Date	Hour	Avg CPU Bsy	Max CPU Bsy	Total Position Rds	Total Pre Rds	Total Disk Rds	Total DB Wrts	Total Merg Wrts
98/08/05	15	67	25	3,167,295	2,674,880	5,842,175	2,430,761	631,881
98/08/05	16	70	20	4,012,771	3,709,689	7,722,460	2,303,323	1,517,651
98/08/05	17	91	6	1,705,905	5,477,610	7,183,515	1,228,384	45,965
98/08/05	18	68	9	1,607,554	1,359,511	2,967,065	1,458,861	55,712
98/08/05	19	43	12	1,424,290	663,117	2,087,407	952,338	249,184
98/08/05	20	40	11	3,133,453	671,244	3,804,697	3,019,145	4,738
98/08/05	21	41	7	3,831,702	323,095	4,154,797	3,282,489	109,957
98/08/05	22	47	9	4,859,217	328,993	5,188,210	3,126,918	1,074
98/08/05	23	43	10	6,032,973	328,317	6,361,290	3,026,434	705

Buddy Backup (UNIX)

Although it is much faster to write to memory than I/O to disk, if a node fails in an MPP system, the database restarts on the remaining available nodes, ignoring the failed node. Any data from committed transactions that is in memory on the node that stayed down is not written to disk.

A buddy backup node prevents this on UNIX MPP configurations. When the system writes segments to memory in one node, it also sends those segments over the BYNET and writes them to memory on another node in the clique (the backup node). When a node fails, the system writes the data to disk from the copy on the backup node.

When a data block is allocated or changed (causing CI changes) the backup node receives the:

- Data block
- Row
- CI information

When a row in an existing data block is updated, and there are no CI changes, the backup node receives the:

- Data block header and trailer
- Row

Note: Although the terms Backup Node and Backup AMP may be used synonymously, there is no such thing as a Backup AMP. All AMPs on a node are backed up by the same node. Backup nodes are assigned at database startup.

The system writes the following types of file system segments to the buddy backup node:

File System Segment	Description
New data blocks	Data blocks that are newly allocated.
Modified data blocks	Inserts, updates, or deletes to an existing data block. (The CI is not changed on fixed length updates or variable length updates that do not require a change in the data block size.)
CIs	If the CI is updated, a copy is sent to the backup node. The CI is changed if: • Data block size changes (smaller or larger) • A new data block is allocated • A data block is deleted • The first or last row ID is changed • The number of rows in a data block changes

File System Segment	Description
TJ data blocks	Rows added or deleted from the TJ.
PJ and Append data blocks	Inserts, updates, or deletes to PJ data blocks.

The system does not write the following types of file system segments to the buddy backup node, because they are released during database recovery and do not impact data integrity:

File System Segment	Description
Spool data blocks	Data blocks used for to spool answer sets for query requests
Spool CI	CI for spool data blocks

Monitoring Backup Activity

The ResUsageSPMA table contains a detailed breakdown of backup node activity, including:

Column	Description
MemsegBackCReads	Number of complete disk segments read for backup storage.
MemsegBackPReads	Number of partial disk segments read for backup storage.
MemsegBackReadKB	KB of disk segments read for backup storage.
MemsegBackFlushes	Number of backup storage disk segments flushed from memory after requesting and assuring that the associated vproc wrote it to disk.
MemsegBackFlushKB	KB of backup storage disk segments flushed from memory.

where *seg* can be replaced as follows:

seg	Description
PDb	Permanent data block disk segments
Pci	Permanent CI disk segments
SDb	Regular or restartable spool data block disk segments
Sci	Regular or restartable spool CI disk segments
TJt	Transient journal table data block or CI disk segments
APt	Appended table, PJ table data block, or CI disk segments

Using the ResNode Macro

Output from the ResNode macro includes the Bkup rds/sec column, which reports the number of segments read by a backup node.

ResUsage and Capacity Planning

Using the ResNode Macro

For capacity planning, generally only ResUsageSPMA is required. This is the ResNode macro set. Important information from ResNode includes:

- CPU utilization
- Parallel efficiency to show hot nodes or AMPs.
- CPU to I/O balance
- OS busy versus DBS busy to see the characteristics of the workload
- Memory utilization
- Availability and process swapping (paging).
- Network traffic

Observing Trends

ResUsage data is most useful in seeing trends when there is a reasonably long history (more than a year) available for comparison. Use this data to answer questions, such as:

- How heavily is the system used at different times of the day or week?
- When are there peaks or available cycles in utilization?
- Are there data skewing issues (as shown in parallel efficiency calculations) that need to be accounted for in the capacity planning cycle?
- How balanced is the system in terms of CPU versus disk I/O?
- How is utilization growing?

Resource Sampling Subsystem Monitor

On Teradata for UNIX, Resource Sampling Subsystem Monitor (RSSmon) displays UNIX and PDE resource usage.

The performance impact of using RSSmon to view ResUsage data is very minimal. RSS has already sampled the resource data and stored the information in memory buffers. RSSmon simply reads and displays the data.

For instructions on how to operate RSSmon, see *Teradata RDBMS Utilities.*

For more information on the ResUsage feature, see *Teradata RDBMS Resource Usage Macros and Tables*.

PM/API and Performance

Introduction

The Performance Monitor / Application Programming Interface (PM/API) provides access to Performance Monitor and Production Control (PM & PC) functions resident within Teradata Database.

PM & PC data is available through a log-on partition called MONITOR using a specialized PM/API subset of the Call-Level Interface Version2 (CLIv2).

The PM/API uses RSS to collect performance data and set data sampling and logging rates. Collected data is stored in memory buffers and is available to the PM/API with little or no performance impact.

Through PM/API commands, performance data is collected on:

- Current system configuration, status and utilization
- Resource usage and status of an individual AMP or PE or node
- Resource usage and status of individual sessions
- Identification and analysis of problem SQL requests

PM/API data may be used to show how efficiently the Teradata database is using its resources, to identify problem sessions and users, and to abort sessions and users having a negative impact on system performance.

For information on PM/API, see *Teradata RDBMS PM/API Reference*.

Chapter 3:

Other Kinds of Data Collection

This chapter describes collecting data associated with system performance other than ResUsage data.

Topics include:

- The DBC.AMPUsage view
- Account String Expansion
- Access logging
- Database Query Log (DBQL)

For information on collecting and using ResUsage data, see Chapter 2: “Collecting and Using Resource Usage Data”.

DBC.AMPUsage View

ResUsage and AMPUsage

While ResUsage is the primary data to use in assessing system usage at the system, node and AMP levels, AMPUsage data is required to see what users are doing.

With DBC.AMPUsage, one can identify:

- Heavy users of the system, over time and at the moment
- Users running skewed work
- Usage trends over time, by group or individual

DBC.AMPUsage View

DBC.AMPUsage is actually a view. The base table is DBC.acctg.

AMPUsage will accumulate CPU and logical disk I/O usage by user and account from day one, even with no additional account identifiers. It writes at minimum, one row per account per AMP.

The data is logged cumulatively, not in intervals as it is with ResUsage. Since data is recorded after a completed step, the one exception would be aborted queries, which would not include the accumulated usage in the step that was actually aborted.

Using the DBC.AMPUsage View

The DBC.AMPUsage view provides cumulative information about the usage of each AMP for each user and account. The system collects and continually adds data to this table until you reset the counters to zero.

Many sites zero the information on a per week, day, or shift basis so that users can make a determination of the resources used during the day and during the night, by UserId and account.

Since the system maintains data on a per-AMP basis, you can check the DBC.AMPUsage table, if processing on your system is skewed, to determine the user that is consuming all the resources on that AMP and may be causing performance problems.

You can use DBC.AMPUsage with ASE (see page 3-4).

For more information on DBC.AMPUsage, see:

- "ResUsage and DBC.AMPUsage View Compared" on page 2-17
- *Teradata RDBMS Data Dictionary*
- *Teradata RDBMS SQL Reference*

Recommendations

Since this data has a tendency to build up quickly, we recommend that you save a summary of this data weekly, or monthly if space is available. Use the DBC.AMPUsage view to update or remove rows in the DBC.Acctg table.

For example, to reset counters for ALL rows or selected rows:

```
UPDATE DBC.AMPUsage
Set CPUTime = 0
,DiskIO = 0
,ALL ;
```

For more information, see the *Teradata RDBMS Database Administration*.

Account String Expansion

Introduction

The Account String Expansion (ASE) feature lets you establish one or more account identifiers when users create or modify other users. ASE allows a more precise measurement of an individual SQL statement execution.

You can use ASE to increase the granularity at which the system takes AMP usage measurements. The system inserts collected information into DBC.AMPUsage.

How ASE Works

Each time the system determines that a new account string is in effect, it begins collecting new AMP usage and I/O statistics. The system stores the accumulated statistics for a user/account string pair as a row in the DBC.AMPUsage view. Each user/account string pair results in a new set of statistics and an additional row.

You can use this information in capacity planning or in chargeback and accounting software. ASE uses the AMPUsage mechanism, but by adding in the substitution variables, the amount of information recorded can greatly increase for the purpose of capacity planning or performance analysis.

At the finest granularity, ASE can generate a summary row for each SQL request. You can also direct ASE to generate a row for each user, each session, or for an aggregation of the daily activity for a user.

The measurement rate may be specified by date (&D), time (&T), or a combination of both. Information can be written to AMPUsage based on the time the user logged on (&L). It can be directed to generate a row for each user, each session, or for an aggregation of the user's daily activities. At the finest granularity, ASE can generate a summary row for every SQL request.

If the user account has a priority associated with it ($L, $M, $H, $R), the priority must appear as the first two positions in the account string. The priority variable must be preceded with a "$". If the variable exceeds a single character, it must be terminated with a "$".

ASE Notation

Following the priority, workload group name and the ASE parameters may be used in any combination and in any order, subject to the constraints on length and position. The expanded account string will be truncated after 30 characters.

The following table provides general ideas on uses of ASE parameters:

ASE Parameter Use/Purpose	Using specified name of Work Group (For example, CRM, cust-supt)
&D	Usage trends by day
&H	Usage trends by hour
&L	Useful for differentiating between multiple session logons
&I	Useful to analyze individual queries
&S	Useful to analyze sessions
&T	For highly granular trend or performance analysis

Using AMPUsage Logging with ASE Parameters

AMPUsage/acctg logging may have both performance and data storage impacts.

The following table summarizes potential impacts:

ASE Parameter	Performance Impact	Data Capacity Impact
None	Negligible	1 row per account per AMP
&D	Negligible	1 row per account per day per AMP
&H	Negligible	1 row per account per hour per AMP
&D&H	Negligible	1 row per account per hour per day per AMP
&L	Negligible	1 row per session pool
&I	Negligible	1 row per SQL request
&S	Negligible	1 row per session
&T	Potentially non-negligible	1 row per query per AMP

Example

Here's an example:

```
Definition:
MODIFY USER AM1 ACCOUNT = ('$M1$&D&H&Sdss','$H1$&D&H&Soltp');
```

```
Usage:

.logon AM1, mypassword, '$M1$&D&H&Sdss'
or during the session:
SET SESSION ACCOUNT = '$H1$&D&H&Soltp'
Breakdown:

For the account string
$M1$&D&H&Sdss = acctid (13 characters, unexpanded)
It is expanded to:
$M1$YYMMDDHHSSSSSSSSSdss = acctid (24 characters, expanded)
$M1$ = session priority
&S&D&H = ASE variable
dss = workgroup / worktype
```

Performance Impact

ASE has little impact on PE performance. The cost incurred for analyzing the account string amounts to only a few microseconds.

The AMP has the burden of additional DBC.AMPUsage logging. Depending on the number of users and the ASE options selected, the added burden may vary from very slight, to enough to degrade performance. In general, the &D, &H, and &L options do not have major effects on performance.

Because the &T option can generate an AMPUsage row for virtually every Teradata SQL request, it can have a much greater effect on performance. Therefore, do not use the &T option:

- In default account ID strings
- In conjunction with OLTP
- With BulkLoad or TPump

The &T option should not be a problem for long-running DSS requests, but could be a performance issue if users are running numerous small requests. &T is site dependent; it should generate no more than 10 to 20 AMPUsage rows per minute.

Note that because ASE causes the system to write more entries to DBC.AMPUsage, you must manage the table more often.

For more information on ASE, see *Teradata RDBMS SQL Reference*.

Access Logging

Introduction

Access Logging enables the system to generate a log entry each time specific privileges are checked for one, some, or all users against one or more database entities.

Implement Access Logging if you need to know which users are requesting access to what entities under what security privileges. For example, if you need to resolve lock conflicts that often occur on the same table, you can track every transaction that required an UPDATE or INSERT privilege on the table, and whether access was granted or denied.

Note: With the Access Logging feature, the Teradata database supports C2 security logging, which complies with United States government security requirements.

Enabling and Disabling Logging

For Access Logging to occur, first you must enable the feature by creating the DBC.AccLogRules macro. Then you activate checking by defining the rules in one or more BEGIN LOGGING statements. After the statements have been processed, you can use the DBC.AccLogRules view to display the stored rules.

Teradata generates a log entry every time it performs a check for a specified privilege. Use the DBC.AccessLog view to see the contents of the log.

You terminate checking for any rules currently in force by defining an END LOGGING statement.

Caution: To disable Access Logging completely, you must drop the DBC.AccLogRules macro and then force a TPA reset.

For details on how to implement C2 security with Access Logging, see *Teradata RDBMS Security Administration.*

Example

The following query provides a summary report on Access Logging data stored in Teradata:

```
Select DT.ld (format 'mm/dd/yyyy', title 'Date')
, DT.st (title '//Statement//Type')
, DT.db (format 'X(13)', title 'DatabaseName')
, DT.cnt (title 'Number//of//Statements')
FROM
(Select LogDate
```

```
, StatementType
, DatabaseName
, count(*)
FROM dbc.accesslog
WHERE StatementType not in ('select', 'Begin/End Logging',
'Show','Exec')
GROUP BY 1,2,3) DT(ld, st, db, cnt
GROUP BY 1,2,3,4
WITH sum (DT.cnt)(title 'Total') BY DT.ld
ORDER BY 1,3,2;
```

Here is a sample of the output:

```
Date       Type                 DatabaseName  Statements
---------- -------------------- ------------- -----------
10/29/2001 Create User DBC 1
----------
Total 1
10/30/2001 Grant dbc 1
10/30/2001 Replace Macro DBC 6
10/30/2001 Replace View DBC 2
10/30/2001 Collect Statistics dss_tables 97
10/30/2001 Create Index dss_tables 41
10/30/2001 Create Table dss_tables 12
10/30/2001 Delete dss_tables 26
10/30/2001 Drop Index dss_tables 15
10/30/2001 Drop Statistics dss_tables 3
10/30/2001 Drop Table dss_tables 12
10/30/2001 Help dss_tables 72
10/30/2001 Insert dss_tables 114
10/30/2001 Rename Table dss_tables 6
10/30/2001 Update dss_tables 542
----------
Total 949
```

Performance Impact

Use Access Logging only as needed. Although this feature can provide a great deal of information, it can increase processing overhead significantly, especially for very short running queries such as prime index selects.

The following results were obtained in-house on a standalone 2-node 4800:

- Prime index selects (worst case) showed as much as a 5x increase in CPU cost.
- DDS operations (best case) showed little to no impact.

Your results will vary due to workload mix, system type, system configuration, and so on.

Database Query Log

Introduction

You can use Database Query Log (DBQL) to log query processing activity for later analysis. Query counts and response times can be charted and SQL text and processing steps can be analyzed to fine-tune your applications for optimum performance.

DBQL provides a series of predefined tables that can store, based on rules you specify, historical records of queries and their duration, performance, and target activity.

DBQL is flexible enough to log information on the variety of SQL requests that run on the Teradata database, from short transactions to longer-running analysis and mining queries. You begin and end collection for a user or group of users and/or one or a list of accounts.

Collection Options

Collection options include:

- Default logging, which reports for each query at least the leading SQL characters, the time of receipt, the number of processing steps completed, the time the first step was dispatched, and the times the first and last response packets were returned to the host.
- Summary logging, which reports at each logging interval the count of all queries and their response time sums that completed processing time within the specified time intervals per active session.
- Threshold logging, which logs a combination of default and summary data:
  - Default data for each query that ran beyond the threshold limit
  - Summary counts of all queries that ran within the threshold time
- Detail logging, which includes:
  - Default data
  - Any or all of the following:
    - Step level activity, including parallel steps
    - Object usage per query
    - Full SQL text

DBQL and Access Logging Overhead Compared

Comparison tests of DBQL and access logging indicate that DBQL logging has significant performance cost savings over access logging.

For detailed information on DBQL, see *Teradata RDBMS Database Administration*.

Chapter 4:

Baseline Benchmark Testing

This chapter discusses Teradata performance optimization through baseline benchmark testing.

Topics include:

- What is a benchmark test suite?
- Baseline profiling
- Baseline profile: performance metrics
- Reliable baseline profiles
- Canary, or heartbeat, queries
- System canaries
- Production canaries

What is a Benchmark Test Suite?

Introduction

A benchmark test suite is really nothing more than a group of queries. The queries picked for this purpose are more accurate and useful if they come from actual production applications.

Test beds such as these can be used to:

- Validate hardware and software upgrade performance
- Measure performance characteristics of database designs and SQL features such as join indexes, temporary tables, or analytical functions
- Assess scalability and extensibility of your solutions architecture (process and data model)
- Distinguish between problems with the platform or database software versus problems introduced by applications

Tips on Baseline Benchmarking Tests

1. Tests should reflect characteristics of production job mix.

   Choose a variety of queries (both complex and simpler ones that are run frequently by users). Remember to include samples from applications that may only be run seasonally (end of year, end of month, and so on)

2. Tests should be designed to scale with the system. Don't expect tables with 200 rows to scale evenly across a system with over a thousand AMPs.

3. Run the benchmark directly before and after upgrades, deployment of new applications and expansions. Comparing a test run to a baseline taken months earlier is not an accurate comparison.

4. Run the benchmark under the same conditions each time. Response times are relative to the amount of work being executed on the system at any particular time. If the system was otherwise idle when the first test was run, it should also be so when executing subsequent runs as well.

Baseline Profiling

Introduction

Baseline profiles provide information on typical resource usage. You can build baseline resource usage profiles for single operations (such as FastLoad, full table scans, primary index INSERT SELECTs, select joins) and also for multiple, concurrently run jobs.

Maintaining profiles of resource usage by user and knowing how patterns of resource usage fluctuate at the site simplifies performance evaluation.

Baseline Profile: Performance Metrics

The following types of performance metrics are recommended:

Metric	Description
Elapsed time	Time for a job or transaction to run from beginning to end (usually in seconds).
I/O rate	Average number of I/O operations per transaction
Throughput rate	Any of the following: • Transaction: total number of transactions in a job divided by job elapsed time • Rows: total number of rows in a table divided by elapsed time of an all-rows transaction • Parallel processing: rows per second per AMP or PE.
Resource utilization	Percentage of time a resource (for example, CPU, disk, or BYNET) is busy processing a job. For example, for a full-table scan, CPU usage may be 30% busy and disk usage may be 70% busy.
Path time	Time a resource spends per transaction or row, which you can calculate as resource utilization divided by throughput rate. For example, a CPU utilization of 70% means the CPU is busy 70% of 1 second, or 0.7 of a second, or 700 milliseconds. If the processing throughput rate is 10 transactions per AMP per second, calculate the path time by dividing 700 milliseconds by 10 transactions. The result is 70 milliseconds per transaction.

Reliable Baseline Profiles

When you have built detailed and accurate baseline profiles, you can:

- Compare current performance metrics against the appropriate baseline profile
- Examine profiles of resource usage to determine which user is at variance with a given standard profile. For example, variance from the norm points to a runaway job or an application change that is inefficient.

Follow the step given below to build reliable baseline profiles:

Step	Action
1	Record all baseline test data, including: • TPA hardware configuration and settings (such as memory per node) • Teradata settings from xclt • Operating system settings • Settings of performance and memory fields in the dbscontrol GDO • Application syntax and workload volume • Mix of application types (OLTP and DS) • Parameters of the target table(s) that may affect performance, such as: – Indexes. For example, uniqueness of PI and SIs, presence of JIs and HIs and whether they are used for query resolution – DDL options affecting I/O, such as BLOCKSIZE, FALLBACK, JOURNAL
2	Run each key application on a dedicated system (where no other jobs or running). If a dedicated NCR server is not available, you can still establish a baseline profile for each job, as long as the workload is consistent and known.
3	Run ResUsage reports for that period, storing the data hourly.
4	Chart resource usage by day, hour, and user.
5	Organize the resulting ResUsage reports so that historical information for each job is represented by time period
6	Develop usage patterns on a daily, weekly, and/or monthly basis.

Canary, or Heartbeat, Queries

Introduction

Using a "canary query," also know as "heartbeat query," as a simple, automated form of monitoring. (A canary query takes it names from the caged canaries miners used to detect poisonous gases. If a canary taken into a mine, usually ahead of the miners, suddenly died, it was a signal not to enter the mine.)

A canary query can be any SQL run at specific intervals whose response time is being monitored.

Use a canary query to:

- Measure response time as an indicator of system demand or system / database hangs.
- Initiate an alert system if response time degrades so that appropriate action can be taken.

Classifying Canary Queries

Although there are many possible actions that can be taken in response to a stalled or "dead" canary, you must first decide what it is you want to measure.

Generally, canary queries can be classified as either:

- System, or
- Production

System Canaries

Introduction

"System canaries," used to check for overall system/database hangs, should be written to take some kind of action when response times reach certain thresholds, or when stalled, such as send alert and/or capture system level information.

More than just a heartbeat check, a system canary should execute diagnostics that capture the state of the system if performance stalls.

System canaries are intended specifically to focus on the Teradata core system. They should be short-running (1 second), low impact queries on tables that are normally not write-locked.

System canaries should be run at frequent intervals (2-3 minutes). They should logon, execute, log off.

They should be run on a system node. This will eliminate other factors such as middle tiers, network connections.

Sample System Canary

The system canary available from the Teradata Global Support Center (TGSC) is the following one-second query:

```
select * from dbc.dbcinfo
```

As the canary runs, a monitor script watches the canary, logging start and end times. If the canary begins to stall, the monitor logs thresholds in 10 second increments beginning at 30 seconds, and then fires a trigger at 90 seconds, which executes hang scripts on each TPA node.

The hang script executes several system level diagnostic tools whose output is saved on every node. The information captured includes, for example:

- Information on AWTs
- Message queue lengths
- System Activity Report (sar) output
- process queue lengths
- BYNET stats

See “System Activity Reporter” on page 6-4 and “puma (NCR UNIX)” on page 6-16.

Production Canaries

Introduction

"Production canaries" may be used to:

- Take response time samplings, storing them for tracking purposes, or
- Monitor the expected response times of specific groups of queries, such as short-running tactical queries running in high priority.

Response times are an indicator of system demand. When system demand is high, canary, or heartbeat, response is high and we can expect all other queries running in the same performance group ti display similar elongations in response time.

From a user perspective, a sudden deviation in response times would have an immediate impact on those users, since users of consistently short running queries would be the first to notice performance degradation.

Using Production Canaries

Production canaries have wider uses than system canaries, and can be used in a variety of ways. For example, they:

- Can be run on production user tables.
- Could be run from other endpoints in the system architecture, such as a network client PC, or MVS client to expand scope of monitoring.
- Monitor overall response.
- Store response times or take alert action. If storing response times, consider overhead of inserts.
- Monitor specific area of the job mix.
- Can be more complex and similar in nature to a particular type of production query, running in the same Priority Scheduler performance group.
- Can be run in multiple performance groups to monitor the system demand of each performance group.

In using a production canary from a non-TPA node location, other things such as network and middle-tier monitoring is also covered, but when it stalls, further investigation must be performed to determine where the bottleneck is located.

Once the response time for a canary is stored in a table, it can be summarized for use in tracking trends.

Chapter 5:

System Performance and the DBS Control Record

This chapter describes the use of those DBS Control Record fields whose values may affect performance. These include:

- Cylinders Saved for PERM
- DBSCacheCtrl
- DBSCacheThr
- DeadLockTimeout
- DegragLowCylProd
- DictionaryCacheSize
- HTMemAlloc
- IdCol Batch Size
- JournalDBSize
- FreeSpacePercent
- LockLogger
- MaxDecimal
- MaxLoadTasks
- MaxParseTreeSegs
- MiniCylPackLowCylProd
- PermDBAllocUnit
- PermDBSize
- PPICacheThrP
- ReadAhead
- ReadAheadCount
- RedistBufSize
- RollbackPriority
- RollbackRSTransaction
- RollForwardLock
- RoundHalfwayMagUp
- RSDeadLockInterval
- SessionMode
- SkewAllowance
- StepsSegmentSize
- SyncScanCacheThr

- TargetLevelEmulation
- WriteDBsToDisk

For information on how to run the dbscontrol utility, see *Teradata RDBMS Utilities*.

DBS Control Record

Introduction

The DBS Control Record stores various fields used by the Teradata RDBMS for the following:

- Debugging / diagnostic purposes
- Establishing known global system values
- Performance tuning

Cylinders Saved for PERM

The value in Cylinders Saved for PERM indicates the number of cylinders saved for permanent data. The value limits the free cylinders for query tasks.

The value in Cylinders Saved for PERM causes spool file management routines to stop short of using the entire system free space. The cost is reduction of the space available per AMP for spool files.

If the number of free cylinders falls below the value in this field, any attempt to allocate cylinders for spool data results in an abort of the requesting transaction.

The default is 10. The range on UNIX is 1 to 100 cylinders. The range on Windows 2000 is 0 to 5000 cylinders.

DBSCacheCtrl

The value in DBSCacheCtrl enables or disables the performance enhancements associated with the DBSCacheThr field.

The default is `TRUE`. This enables the DBSCacheThr setting to control the caching of data blocks.

If you change the value to `FALSE`:

- Data blocks read during sort operations are not cached
- All other data blocks are cached using the least recently used algorithm.

Carefully consider the behavior resulting from the DBSCacheThr setting before making a decision about DBSCacheCtrl.

DBSCacheThr

The value in DBSCacheThr specifies the percentage to use for calculating the cache threshold when DBSCacheCtrl is set to TRUE.

Depending on the size of File System Segments (FSG) Cache and the size of the tables in the databases, the value in this field can make a big difference to how much useful data the system actually caches. Using cache saves on physical disk I/Os, which implies that caching the smaller and more frequently-accessed tables (usually reference tables) is recommended. You can use the DBSCacheThr value to encourage these smaller tables to stay in memory longer.

Use DBSCacheThr to prevent a large, sequentially read or written table from pushing other data out of the cache. Since the system probably will not access table data blocks again until they have aged out of memory, it does little good to cache them, and may cause more heavily accessed blocks to age out prematurely.

Large history tables are not the primary tables to cache. In the case of multiple users that access the same table at the same time, the system can do a synchronized scan (sync scan) on the table.

Before making a decision about changing the default, review the description of DBSCacheThr in the chapter titled “DBS Control Utility” in *Teradata RDBMS Utilities*.

Recommendation

Set this field to as small a value as possible to keep out the smallest large data table, but larger than the largest reference table or the largest spool to be kept in memory.

Because the system also uses this field as a threshold for keeping spools in memory, do not make this field value too small. The larger the memory (for example 2 GB), the smaller the value of DBSCacheThr.

Use the following formula:

DBSCacheThr = (SizeOfTable/NumberOfNodes) / AdjustedFSGCache

where AdjustedFSGCache = FSG cache x FSG Cache Percent.

For example, to keep a reference table in cache, assume that you have:

- A 1 million row table with 100 byte row = 100 MB table
- 10 nodes at 10 MB/node

System	Adjusted FSG Cache	Recommended DBSCacheThr
1GB	500MB	2% or greater
2GB	1.5 GB	1% or greater
4 GB	3.5 GB	1% or greater

where 1% is the smallest value you can specify.

To keep out the smallest large data table, assume, for example, that you have:

- A 10 million row table with 100 byte row = 1000 MB table
- 10 nodes at 100 MB/node

System	Adjusted FSG Cache	Recommended DBSCacheThr
1GB	500MB	20% or less
2GB	1.5 GB	6% or less
4 GB	3.5 GB	2% or less

DeadLockTimeout

The value in DeadLockTimeout specifies the time-out value for jobs that are locking each other out in different AMPs. When the system detects a deadlock, it aborts one of the jobs.

Pseudo table locks reduce deadlock situations for all-AMP requests that require write or exclusive locks (see"AMP-Level Pseudo Locks and Deadlock Detection" on page 14-7.) However, deadlocks still may be an issue on large systems with heavy concurrent usage. In batch operations, concurrent jobs may contend for locks on the Data Dictionary tables.

Recommendation

Reduce the value in this field to cause more frequent retries with less time in a deadlock state.

Faster CPU chips significantly reduce the system overhead for performing deadlock checks, so you can set the value much lower than the current default of 240 seconds. In general:

IF your applications...	THEN you should...
incur some dictionary deadlocks	set the value to between 30 and 45 seconds
incur few dictionary deadlocks	retain the default value of 240 seconds
incur many true deadlocks	set the value as low as 10 seconds
are predominantly Online Transaction Processing (OLTP) applications	set the value as low as 10 seconds

DefragLowCylProd

The value in DefragLowCylProd specifies the threshold at which to perform a cylinder defragmentation operation. The system dynamically keeps cylinders defragmented.

If the system has less than the specified number of free cylinders, defragmentation occurs on cylinders with at least 25% free space, but not enough contiguous sectors to allocate a data block.

Recommendation

Set this field higher than MiniCylPackLowCylProd (page 5-20) because defragmentation has a smaller performance impact than cylinder pack.

DictionaryCacheSize

The maximum size of the dictionary cache depends on the value in the DictionaryCacheSize field.

Recommendation

The recommended value is 1024 KB. This allows more caching of table header information and reduces the number of I/Os required, which is especially effective when the workload is accessing many tables (more than 200) or when the workload generates many dictionary seeks.

For OLTP and Online Complex Processing (OLCP) type workloads, a better response time of even a few seconds is important. For query workloads with a response time of more than one minute, there is no measurable difference when this field is set to a higher value.

DisableSyncScan

The value in DisableSyncScan allows you to enable (set to FALSE) or disable (set to TRUE) synchronized full file scans. When enabled, DisableSyncScan works with SyncScanCacheThr to specify the percentage of free memory for synchronized full-file scans.

Synchronized table scans:

- Allow multiple scans to share I/Os by synchronizing reads of a subtable. There is no limit to the number of users who can scan data blocks in sync.

  If the database receives multiple requests to scan the same table, it can synchronize, or share I/Os, among such scans. Teradata starts a new scan from the current position of an existing scan, and records where the second scan starts. When the second scanner reaches the end of the table, it automatically starts over at the beginning of the table and proceeds until it reaches its original starting position, thereby completing the scan.

  Teradata synchronizes a new scan with the existing scan that has accessed the least amount of data and, therefore, has the most left to do. This way, the two scans can share I/Os for a long time. The scans are weakly synchronized, that is:

  – Even though Teradata initially synchronizes one scanner with another, the scanners do not proceed in lock step but remain independent from each other.
  – Two synchronized scanners may do different amounts of work when processing rows, so one may be slower than the other. Therefore, it is possible for them to diverge over time. If scanners diverge too much, scans are no longer synchronized, and the system discards the data blocks immediately upon release.
- Are used in a decision support environment for full-table scans that do not fit into the existing memory cache.

If the system is already I/O-bound, the reduced I/O from sync scan is quite noticeable. The system keeps data blocks in memory as long as space is available, as defined by SyncScanCacheThr.

FreeSpacePercent

The value in FreeSpacePercent specifies the default amount of space on each cylinder to be left unused during certain operations. Use this field to reserve space on a cylinder for future updates and avoid moving data to other cylinders in order to make room. (See "PACKDISK and FreeSpacePercent" on page 9-8 for more information.)

Recommendation

Use a higher value if most of your tables will grow and a lower value if you expect little or no expansion. If you have a variety of tables that will and will not grow, set this field for the majority. Use the CREATE or ALTER TABLE statements to set other values at the table level.

HTMemAlloc

The value in HTMemAlloc specifies the percentage of memory to be allocated to a hash table for a hash join. The hash join occurs as an optimization to a merge join under specific conditions. A hash join can save the time it takes to sort the right-hand (usually the larger) table of two tables in a join step. The saving can occur under the following situations:

- The left table is *duplicated* and the join is on non-Primary Index (PI) columns. For the merge join to take place, the right table must be sorted on the row hash value of the join columns.

  The hash join replaces this sort by maintaining the left table as a hash table with hash values based on the join columns. Then, the hash join makes a single pass over the right table, creating a row hash on the join column values and doing a table lookup on the (left) hash table.
- Both the left and right tables are *redistributed*. For a merge join to occur, both tables must be sorted based on the row hash value of the join columns; when the right table is large enough, sorting the table requires multiple passes over the data.

  The hash join makes only a single pass over the data, hence producing the savings and the value of the hash join.

However, a hash join works well only as long as the smaller hash table remains in memory and if no AMP has a high skew rate:

IF...	THEN...
the hash table is too large to remain in memory	the hash join makes multiple passes of the larger right table.
there is a high skew in an AMP	the hash table for the AMP may not fit in the HTMemAlloc size, and the multiple pass situation may provide a poorer query response time.

Recommendation

If your system is using large spool files and the Optimizer is not using the hash join because of the HTMemAlloc limit, increase HTMemAlloc and see if performance improves.

This field works with SkewAllowance (see page 5-39).

See additional information on this field, on hash table size calculations, and possible values under "Hash Joins" on page 12-26 and under HTMemAlloc in the chapter titled "DBS Control Record" in *Teradata RDBMS Utilities*.

IdCol Batch Size

The IdCol Batch Size field specifies the size of a pool of numbers reserved by a vproc for generating numbers for a batch of rows to be bulk-inserted into a table with an identity column.

When the initial batch of rows for a bulk insert arrives on a PE/AMP vproc, the following occurs:

- First, a range of numbers is reserved before processing the rows.
- Then, each PE/AMP retrieves the next available value for the identity column from the IdCol table.
- Finally, each PE/AMP immediately updates this value with an increment equal to the IdCol Batch Size setting.

The valid range of values is 1 to 1 million.

The default is 100,000.

The new setting becomes effective after the DBS Control Record has been written or applied.

Note: The IdCol Batch Size field settings survive system restarts.

The IdCol Batch Size setting makes a trade-off between performance and numbering gaps that can occur in a restart. A larger setting might improve the performance of bulk-inserts into an identity column table, since there will be less updates of DBC.IdCol in reserving batches of numbers for a load. However, since the reserved numbers are kept in memory, unused numbers will be lost if a restart occurs.

When setting the IdCol Batch Size, consider the following:

- The data type of the identity column
- The number of vprocs serving the bulk insert.

Note: An INSERT-SELECT should base the IdCol Batch Size setting on the number of AMPS. Other bulk insert statement should base the IdCol Batch Size setting on the number of PEs.

JournalDBSize

The value in JournalDBSize determines the maximum size of Transient Journal (TJ) and Permanent Journal (PJ) table multi-row data blocks, written during insert, delete, and update operations.

The absolute maximum journal block size is 255 sectors on UNIX and 127 sectors on Windows NT/Windows 2000. The default is 12.

Recommendation

For applications using permanent journaling, and for applications adding many rows to populate permanent or temporary tables, try setting this field to 32 sectors (16 KB).

If the rows involved in these applications are very long, or many rows are being manipulated, try increasing JournalDBSize accordingly. A larger size also can produce significant savings if the system is I/O bound.

In general, the maximum multi-row data block size for journals should agree with the data row length. If the modified rows are short, the journal data block size can be small. If the modified rows are long, the journal data block size can be large.

If you base data block size on processing activity, the following rules are generally successful for good performance when the workload is mixed:

- PermDBSize (page 5-24) should be a large number to optimize decision support, especially queries involving full-table scans.
- JournalDBSize should be a low number to benefit analytic functions and High-Availability Transaction Processing (HATP) operations.

LockLogger

The value in LockLogger defines the system default for the Locking Logger, and allows you to log delays caused by database locks and help identify lock conflicts. Locking Logger runs as a background task, recording information in a circular buffer on each AMP. It then writes the data to the Lock Log table, which you must have already created.

LockLogger is useful for troubleshooting problems such as determining whether locking conflicts are causing high overhead.

Some values in the Lock Log table represent internal IDs for the object on which the lock was requested. The Lock Log table defines the holder and the lock requester as transaction session numbers. You can obtain additional information about the object IDs and transaction session numbers by joining your Lock Log table with the DBC.DBase, DBC.TVM, and DBC.EventLog tables.

MaxDecimal

The value in MaxDecimal defines the number of decimal digits in the default maximum value used in expression data typing.

MaxLoadTasks

The value in MaxLoadTasks controls the number of load tasks such as FastLoad, MultiLoad, and FastExport that can run on the system simultaneously. The default is 5. A zero value means none of these tasks are allowed.

MaxParseTreeSegs

The parser allocates 64 KB tree segments when parsing a Teradata SQL request. MaxParseTreeSegs defines the maximum number of 64 KB tree segments the parser allocates while parsing a request. This is an enabling field rather than a performance enhancement field.

Recommendation

If you are using a front-end tool to generate complex SQL queries, set this field to 246 to allow for 64 table joins.

The more complex the queries, the larger you need to set this field for code generation. Ordinarily, users do not need to change this field unless their queries run out of memory (3710/3711 errors). If you want to limit the query complexity, you can set this field as low as 32.

MiniCylPackLowCylProd

The value in MiniCylPackLowCylProd specifies the number of free cylinders below which an anticipatory mini-cylinder pack (mini-cylpack) operation begins. The mini-cylpack operation performs in the background.

Stage	Process
1	Mini-cylpack pack scans the Master Index, a memory-resident structure with one entry per cylinder, looking for a number of logically adjacent cylinders with a lot of free space.
2	When mini-cylpack finds the best candidate cylinder, it packs these logically adjacent cylinders to use one less cylinder than is currently being used. For example, mini-cylpack packs 4 cylinders that are each 75% full into 3 cylinders that are 100% full.
3	The process repeats on pairs of cylinders until mini-cylpack successfully moves all the data blocks on a cylinder, resulting in a free cylinder. This whole process continues until either: • No additional cylinders can be freed. • The number of free cylinders reaches the value in MiniCylPackLowCylProd.

By running in the background and starting at a threshold value, a mini-cylpack minimizes the impact on response time for a transaction requiring a new cylinder. Over time, however, though, mini-cylpack may not be able to keep up with demand, due to insufficient free CPU and I/O bandwidth, or to the increasing cost of freeing up cylinders as the demand for free cylinders continues.

The following table provides information on the results of setting MiniCylPackLowCylProd to a nonzero or zero value.

IF you set MiniCylPackLowCylProd to…	THEN...
a nonzero value	mini-cylpacks run in anticipation of the need for free cylinders. When running in this mode, each mini-cylpack scans and packs a maximum of 20 cylinders. If mini-cylpack cannot free a cylinder, further anticipatory mini-cylpacks do not run until another cylinder allocation request notices that the number of free cylinders has fallen below MiniCylPackLowCylProd. If you set this field to a low value, you reduce the impact of anticipatory mini-cylpacks on performance. However, there is a risk that free cylinders will not be available for tasks that require them. This will cause mini-cylpacks to run while tasks are waiting, thereby seriously impacting the response time of such tasks.
zero	the anticipatory mini-cylpack operation is disabled. A mini-cylpack is run only when a task needs a cylinder and none are available. The requesting task is forced to wait until the mini-cylpack is complete. When a mini-cylpack runs while a task is waiting, the number of cylinders that mini-cylpack can scan is unlimited. If necessary, mini-cylpack scans the entire disk in an attempt to free a cylinder.

Recommendation

Set this value to no more than 20 free cylinders.

PermDBAllocUnit

The value in PermDBAllocUnit specifies the allocation unit for the multi-row data blocks of the permanent table.

If PermDBAllocUnit is not an integer factor of the absolute largest data block, multi-row data blocks are always smaller than the maximum, as the following table illustrates:

IF you set PermDBAllocUnit to...	THEN the largest multi-row data block is...		
	V2R2.x.x (default maximum, 63 sectors)	V2R3.0.x and V2R4.0.x (default maximum, 127 sectors)	V2R4.1 (default maximum, 255 sectors)
4 (even if PermDBSize is the default)	60	124	252
16	48	112	240

Changing this field from the default of one sector affects the maximum size of the permanent data blocks for non-read-only tables.

When FastLoad or an INSERT SELECT initially populates an empty table, the system packs the rows into the maximum data block size.

For FastLoaded tables and tables you modify via ALTER TABLE with a block size clause and the IMMEDIATE option, blocks are created with sizes nearly equal to the value of the multi-row block size. The blocks remain that size until you insert, delete, or modify rows.

For tables that are heavily modified, the blocks tend toward a size of 75% of the maximum multi-row size. You can see this by looking at the normal growth cycle of blocks as rows are inserted. As you add rows to the table, the maximum block size is split into 2 16-KB blocks. Thereafter, the block size grows by PermDBAllocUnit.

If you set the value to 8 sectors (4 KB), the block grows from 16 KB to 20 KB, 24 KB, and 28 KB, respectively. Since another 4 KB makes the block larger than the maximum block size of 31.5 KB, the block size remains at 28 KB, or is split into a 14 and a 14.5 KB block. So the average over time is halfway between 1/2 and 1 times the multi-row block size.

If a modification causes the amount of free space to exist in a block, the block shrinks to the minimum number of integral sectors required, and the extra sectors are freed.

If you change this field, you will not see a significant boost in performance.

Recommendation

The file system can sometimes perform modification operations more efficiently if the size of a data block does not change. Potentially, then, you can optimize performance by setting PermDBAllocUnit to values higher than 1.

But setting the value higher than 1 can increase the required disk utilization required. Therefore, do not set PermDBAllocUnit to an arbitrarily high value.

PermDBSize

The value in PermDBSize specifies the default maximum size, in consecutive 512-byte sectors, of a permanent multi-row data block. (Also see "JournalDBSize" on page 5-15.)

PermDBSize works in conjunction with the value in the PermDBAllocUnit field (page 5-22). For example, if PermDBAllocUnit is not an integer factor of 127 (the absolute largest data block), then the largest multi-row data blocks are always smaller than 127.

PermDBSize affects all tables; however, you can override this value on an individual table with the DATABLOCKSIZE option of the CREATE TABLE and ALTER TABLE statements.

Note: If your workload varies table by table, always specify data block size at the table level instead of using PermDBSize.

General guidelines for setting PermDBSize to suit your applications (also see JournalDBSize and PermDBAllocUnit) are as follows:

IF you...	AND you...	The initial cylinder size default is...	The initial PermDBSize default is...	AND...	THEN...
upgrade from V2R3.x.x to V2R4.0.x	do not want to run Sysinit and reload your data	the V2R3.x.x value at time of upgrade	the V2R3.x.x value at time of upgrade	your tables are large and rarely updated, and used mainly for selecting data, especially selects that cause full-table scans	1 Leave PermDBSize at its current value. 2 Use ALTER TABLE ... BLOCKSIZE to set a low maximum, perhaps 14 sectors, for tables having many update operations.
upgrade from V2R3.x.x to V2R4.0.x	do not want to run Sysinit and reload your data	the V2R3.0.x.x value at time of upgrade	the V2R3.x.x value at time of upgrade	your tables contain historical data, or data used for OLTP applications, and are subject to many inserts, deletes, and updates	1 Leave PermDBSize at its current value. 2 Use ALTER TABLE ... BLOCKSIZE to set a high maximum, from 63 to 127 sectors, for read-only tables, especially those used in full-table scans.

IF you...	AND you...	The initial cylinder size default is...	The initial PermDBSize default is...	AND...	THEN...
upgrade from V2R4.0.x to V2R4.1	do not want to run Sysinit and reload your data	the V2R4.0 value at time of upgrade	the V2R4.0 value at time of upgrade	your tables are large and rarely updated, and used mainly for selecting data, especially selects that cause full-table scans	**1** Leave PermDBSize at its current value. **2** Use ALTER TABLE ... BLOCKSIZE to set a low maximum, perhaps 14 sectors, for tables having many update operations.
upgrade from V2R4.0.x to V2R4.1	do not want to run Sysinit and reload your data	the V2R4.0 value at time of upgrade	the V2R4.0 value at time of upgrade	your tables contain historical data, or data used for OLTP applications, and are subject to many inserts, deletes, and updates	**1** Leave PermDBSize at its current value. **2** Use ALTER TABLE ... BLOCKSIZE to set a high maximum, from 127 to 255 sectors, for read-only tables, especially those used in full-table scans.

IF you...	AND you...	The initial cylinder size default is...	The initial PermDBSize default is...	AND...	THEN...
upgrade from V2R3.x.x to V2R4.0.x, or upgrade from V2R4.0.1 (MP-RAS) to V2R5.0, or upgrade from V2R4.1.0 (Windows 2000) to V2R5.0	still have 1488 sectors per cylinder and are willing to run Sysinit and reload your data	before Sysinit, 1488 sectors per cylinder	the V2R3.x.x value at the time of upgrade	most of your tables are OLTP, so you need to set PermDBSize to a high value; but you also want to: • Avoid losing disk space through fragmentation • Increase the AMP size configuration	**1** Unload your data **2** Upgrade to V2R4.0.x **3** On UNIX, you need to run pdeconfig **4** Run Sysinit and change the cylinder size to 3872 sectors. This increase: – Mitigates a possible increase in cylinder fragmentation – Allows a maximum AMP configuration of 1.3 TB. (For 1488 sectors, maximum AMP configuration on R4.0.1 is 520 GB) – Resets PermDBSize to the initial default of 127 sectors **5** Reload your data **6** Check PermDBSize: – If your tables are often updated, such as with OLTP, set PermDBSize low. – If your tables involve selects and all-rows scans, use default. – If applications are mixed, use ALTER TABLE with DATABLOCKSIZE for tables of the opposite type. **Note:** Contact your NCR support representative to change the cylinder size.

IF you…	AND you…	The initial cylinder size default is...	The initial PermDBSize default is...	AND…	THEN…
have a new V2R4.0.x or above installation	retain the pre-defined default values	3872 sectors	127 sectors		• If your tables involve mainly selects and all-rows scans, leave PermDBSize high. • If your tables are often updated, such as with OLTP and HATP, set PermDBSize low. • If your applications are mixed, use ALTER TABLE with DATABLOCKSIZE for tables of the opposite type of application.

Performance Impact of Larger Datablock Size

In general, datablock size should be kept to less than, or equal to, 64KB (127 sectors), and increasing the datablock size to 128KB (255 sectors) should only be done after careful evaluation of the system workloads.

For example, when the workload is mainly DSS and very few single-row access operations are performed, the datablock size can be set to 255 at the system level for all the tables.

PPICacheThrP

The value in PPICacheThrP specifies the percentage to be used to calculate the cache threshold for operations dealing with multiple partitions.

The PPICacheThrP value controls the memory usage of PPI operations. Larger values improve the performance of these PPI operations, as long as the following occur:

- Data blocks can be kept in memory (at which point performance might degrade)
- The number of partitions in a table is not exceeded (in this case, increasing the value does not improve performance)

You should decrease the value if memory contention occurs because of the preceding operations. The value is specified in units of 1/10th of a percentage. The valid range of values is 0 to 500. The default is 10 (1%).

ReadAhead

ReadAhead is useful for sequential-access workloads. When this field is set to `TRUE`, Teradata issues a ReadAhead I/O to load the next sequential block, or group of blocks, into memory when a table is being scanned sequentially.

Loading data blocks in advance reduces I/Os and can improve processing time significantly, especially when running tools such as ScanDisk.

The default is TRUE because without at least one pre-load data block in memory, usually throughput suffers and I/O completion takes longer during sequential-access operations.

The number of blocks to be pre-loaded is determined by the ReadAheadCount performance field.

ReadAheadCount

ReadAheadCount defines the number data blocks to be pre-loaded during sequential-access scanning operations. The default is 1.

In general, CPU throughput should exceed the time it takes to read the number of blocks you specify. Thus, the slower the CPU throughput, the less blocks should be pre-loaded; the faster the CPU throughput, the more blocks should be pre-loaded.

For example, if most of your applications use large data blocks, the default should suffice. If most use small data blocks, you should benefit by increasing ReadAheadCount to 25 or higher.

Setting ReadAheadCount to `0` is not usually beneficial.

RedistBufSize

RedistBufsize determines row redistribution buffer size. For the redistribution of data from AMP to AMP, the system reduces message overhead by grouping individual rows (or messages) into blocks before sending.

Each AMP has *N* buffers for managing redistribution data. If there are *N* AMPs in the system, there are N^2 total buffers in the system.

To illustrate memory usage, let's say that your system has 8 AMPs/node, which means there are 8 *N* buffers/node, each at RedistBufSize. As a consequence, the amount of memory used for redistribution in a node grows as the system size grows.

The information in the following table illustrates this growth in redistribution buffer size for a single distribution.

IF the system configuration is...	THEN the number of buffers is...	AND required memory (MB) is...	Comment
8 nodes with 8 AMPs/node	512 ((8*8)*8)	2	The default RedistBufSize is 4 KB.
12 nodes with 8 AMPs/node	768 ((8*12)*8)	3	The memory requirement grows a proportional 50%.
48 nodes with 8 AMPS/node	3072 ((8*48)*8)	12	In single user environment, this is not a problem. But with many concurrent users, this could use up all available free memory and put the system into a swapping state.

Recommendation: If you have many AMPs per node, a small buffer size is generally better for performance. If you have few AMPs per node, a large buffer size is generally better for performance. A large buffer size will also benefit joins with a small spool row size.

Conservatively, maintain the RedistBufSize at 4 KB for up to 48 nodes with 8 AMPs/node. As the system configuration grows larger, you can compensate by doing one or more of the following:

- Set RedistBufSize smaller in proportion to the increase in the total number of AMPs (that is, send more smaller messages)
- Add more memory to increase the total memory in the system to accommodate the redistribution buffers up to 4 GB per node
- Increase the amount of free memory available for redistribution buffers by setting FSG Cache Percent smaller

To help determine if RedistBufSize is too high, see if the minimum available free memory consistently goes below 40 MB during heavy periods of redistribution. Also, check for significant swapping (more than 10/second) during this period. If this is the case, reduce RedistBufSize an incremental value lower, for example, 4 to 3 KB.

RollbackPriority

The value in RollbackPriority defines at which priority transaction rollbacks are executed.

- Setting RollbackPriority to FALSE means that subsequent transaction aborts will be rolled back at the RUSH priority.
- Setting it to TRUE means subsequent transaction aborts will be rolled back at the priority of the aborted job (LOW, MEDIUM, or HIGH, as applicable).

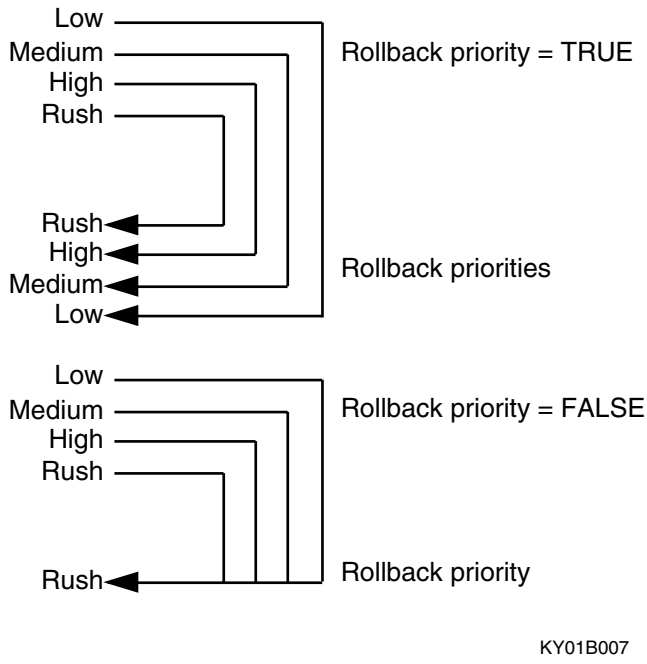
Therefore, to cause rollbacks to occur as fast as possible, set the value to FALSE.

Note: A change to RollbackPriority does not take effect until the next restart. To make a change effective immediately, do a tpareset.

Aborting user transactions at the RUSH level releases the lock on the affected table(s) more quickly than at other priority levels, as illustrated below.

Recommendation

To lessen the impact on users whose transactions are normally assigned a RUSH priority, leave RollbackPriority set to TRUE unless a critical situation requires a quick release of a transaction lock held by a non-rush aborted transaction.



RollbackRSTransaction

The value in RollbackRSTransaction controls which transaction is rolled back when a user transaction and a subscriber-replicated transaction are involved in a deadlock. `TRUE` rolls back the subscriber-replicated transaction.

RollForwardLock

The value in RollForwardLock defines the system default for the RollForward using the Row Hash Locks option.

During a RollForward operation, you can use RollForwardLock to specify whether or not to use row hash locks.

Row hash locks reduce lock conflicts, making users more likely to be able to access data during the RollForward operation.

RoundHalfwayMagUp

The value in RoundHalfwayMagUp specifies how rounding should be performed when computing DECIMAL values. The default is `FALSE`, which specifies the rounding semantics traditional for Teradata. If you prefer standard rounding, in which halfway values are rounded away from zero, set this value to `TRUE`.

RSDeadLockInterval

The value of RSDeadLockInterval determines the interval, in seconds, between detection cycles. If the value is 0, then the Deadlock Time Out (page 5-8) value is used.

RS deadlock checking is used only if your system is configured with RSG vprocs and the Relay Services Gateway (RSG) is up.

Session Mode

The value in Session Mode defines the global system default for transaction mode, case sensitivity, and character truncation rule. The default is zero (Teradata mode). If you want to run transactions only in ANSI mode, reset this value to 1.

The global value can be overridden during logon at either the user or the session level. For details, see the SET SESSION statement and the ACCTID clause of the CREATE USER statement in the *Teradata RDBMS Reference*.

SkewAllowance

The value in Skew Allowance specifies a percentage factor used by the Optimizer in deciding on the size of each hash join partition. Skew allowance reduces the memory size for the hash join specified by HTMemAlloc. This allows the Optimizer to take into account a potential skew of the data that could make the hash join run slower than a merge join.

Consider an example with the following configuration and field settings:

- 1 GB of memory/node
- Memory for UNIX, 8 AMPs and 2 PEs/node, where (340 MB = 10 vprocs *32 MB + 20 MB)
- FSG Cache Percent at 80%
- HTMemAlloc at 1% (see page 5-13 for more information)
- SkewAllowance 75%

Intermediate calculations are as follows:

FSG Cache =1 GB-340 MB
=1024-340 MB
= 684 MB/node

Available Memory (AdjustedFSGCache) = FSG Cache *
FSG Cache Percent
= 684 MB * 0.80
= 547 MB/node

Hash Table Memory Allocation = 547 MB * 0.01
= 5.5 MB/node or 700 KB/AMP (for 8 AMPs/node)

Adjustment for Skew = Hash Table Memory Allocation
* (1 - SkewAllowance)
= 5.5 MB * 0.25
= 1.4 MB/node

Optimizer Threshold Value = 1.4 MB / # AMPs per node
= 1.4 MB/8 (for 8 AMPs/node)
= 175 KB/AMP

Thus, if the size of a spool is less than 175 KB/AMP, the Optimizer considers using the hash join instead of a merge join.

If the SkewAllowance is 0%, the Optimizer uses 700 KB/AMP as the threshold to determine whether to use the hash join.

Following is another example:

System (GB)	AdjustedFSGCache	HTMemAlloc (MB) (Default 1%)	SkewAllowance (MB) (Default 75%)	8 AMPS/Node (KB)
1	550 MB	5.5	1.4	175
2	1.5 GB	15	3.75	470
4	3.5	35	8.75	1094

Note: The default value of 75% is a conservative number that permits a skew four times the limit of what the Optimizer uses to consider using the hash join. If you set SkewAllowance too low and there is a high skew of the data, the hash join could take longer than a merge join.

For example, if you set SkewAllowance to 0%, and the skew is 4 times the size the size of HTMemAlloc, the hash join makes 4 passes over the right table.

The hash join works optimally when the hash table fits in memory all at once, so that the hash join makes only one pass over the right table. If the hash table is four times larger than HTMemAlloc, only one-fourth of the hash table can be kept in memory at a time, and each time the has join makes a full pass on the right table.

Recommendation

The default of 75% is the recommended value. If you know your data very well and do not expect skewing at this extreme, you can set this value to 50%, which still allows for a skew that is double the size the Optimizer uses in its estimates. Consider a different setting if data is so badly skewed that hash join degrades performance. In this case, you should turn the feature off or try increasing the SkewAllowance to 80. Set this field together with HTMemAlloc (see page 5-13).

StepsSegmentSize

The value in StepSegmentSize defines the maximum size (in KB) of the plastic steps segment.

When decomposing a Teradata SQL statement, the parser generates plastic steps, which the AMPs then process. StepsSegmentSize defines the maximum size of each plastic steps segment.

Recommendation

Large values allow the parser to generate more SQL optimizer steps than the AMPs use to process more complex queries. Set this field to a small number to limit the query complexity. Set this field to 1024 KB for the maximum size allowable for plastic steps.

SyncScanCacheThr

The value in SyncScanCacheThr indicates how much memory the system can use to keep scans for large tables in synchronization scan (sync scan) mode. Sync scan can occur when two or more queries perform a full table scan on the same large table that exceeds DBSCacheThr. Multiple tables can also be in sync scan mode at the same time.

WHEN...	THEN...
the queries that are in sync scan mode process the data blocks at differing speeds	a gap appears between the fastest and the slowest query on the table.
the combined gap of all tables in sync scan mode, in total size in bytes, exceeds SyncScanCacheThr	at least one query falls out of sync scan mode.
three or more queries are in sync scan mode on the same table	the farthest query behind falls out of sync scan mode.

Recommendation

The recommended value for this field is 5%-10%.

Note: If you set SyncScanCacheThr too high (for example, 50%), smaller reference tables will age out and negate the benefits of the DBSCacheThr.

Compute the amount of memory available to cache data for all tables involved in full-table sync scans similar to DBSCacheThr:

Threshold = (SyncScanCacheThr * AdjustedFSGCache)/100

where AdjustedFSGCache = FSG cache * FSG Cache Percent.

System (GB)	AdjustedFSGCache	SyncScanCacheThr (MB)		
		10%	5%	1%
1	500 MB	50	25	5
2	1.5 GB	150	75	15
4	3.5 GB	350	175	35

The system divides this value by the number of tables, each with multiple scanners, which are participating in the various sync scans. (As long as there is more than one, the number of scanners per table is irrelevant.) The system uses the result to determine if multiple scanners of a table are still synchronized.

For example, assume that a table has two scanners. If the amount of disk that must be scanned for the lagging scanner to catch up to the leading scanner is less than this value, the system considers the two scans synchronized.

However, if it is more than this value, the system no longer considers the scans synchronized, and both scanners cease to cache their data. (The actual computation is more sophisticated, since there can be multiple independent synchronization points on the same table for four or more scanners, but the essence of the computation remains the same.)

Note: When two tasks are accessing the same block and one ages the block normally (caches it) and the other discards it, the block is aged normally, that is, the higher age wins.For example:

TargetLevelEmulation

NCR does not recommend enabling Target Level Emulation on a production system. The default is FALSE.

A value of TRUE enables a set of diagnostic SQL statements that NCR support personnel can use to set costing parameters the Optimizer considers. For more information, see *Teradata RDBMS SQL Reference*.

WriteDBsToDisk

The value in WriteDBsToDisk forces the writing of data blocks directly to disk rather than committing them to a backup node.

Note: Backup nodes are present only on UNIX systems that are configured with cliques.

This field only affects permanent tables other than the transient journals.

The PDE FSG subsystem manages a data block cache containing both primary and backup data. When the number of backup blocks reaches a threshold limit, the PDE FSG subsystem releases a number of least-recently-used segments from the primary cache, including unmodified segments, which are simply discarded. Modified segments are written to disk and then discarded.

This allows the system to reclaim memory used for the backup cache, and reduces the number of blocks the system stores on the backup node.

When the backup cache size falls below a lower threshold, the PDE FSG subsystem allows the primary cache to grow. The difference between the upper and lower threshold limits is carefully chosen to avoid thrashing.

Recommendation

You can try setting this field to `TRUE` for applications where a large percentage of modifications to permanent data blocks affect only write per block. In this case, the overhead of committing updates to the buddy node overrides the benefit of keeping the block in memory.

Caution: Exercise care before changing the WriteDBsToDisk default value to FALSE. The extra work to send a row to the buddy node is only beneficial to overall performance if it actually saves physical I/Os on the originating node. Enhancements to Teradata software (full table modifications operating on datablocks at a time rather than rows at a time) greatly reduce the likelihood that datablocks can be updated multiple times in the buddy node before requiring a physical disk I/O on the originating node.

Chapter 6:

Tools for Monitoring and Tuning System Performance

This chapter provides information on tools that monitor and tune Teradata performance.

Topics include:

- How busy is too busy?
- System Activity Reporter (sar)
- xperfstate
- sar and xperfstate compared
- sar, xperfstate, and ResUsage compared
- TOP
- BYNET Link Manager Status (blmstat)
- ctl and xctl
- puma
- Resource Check Tools

How Busy is Too Busy?

CPU Saturation

Systems that run frequently at or near capacity are often the subject of assessments in which attempts are made to determine the extent of resource exhaustion.

When CPU is the binding factor and the scarce resource on a system, it is useful to determine the systems' highest level of saturation, that is, the point to which one can drive the system before it becomes so overloaded that it appears to be hanging.

Once a system reaches this level of saturation, it should still be able to work itself out, but it takes extra time to do so.

Without appropriate performance monitoring, users may start, once the system appears to be hanging, to:

- Abort jobs that could cause rollbacks, or
- Submit duplicate queries that create more work on an already-exhausted system.

While ResUsage data provides the bulk of the information needed to know that a system is at 100% busy with respect to CPU or CPU+I/O Wait, other information may also be needed in order to examine the extent of CPU saturation and system congestion. In other words, one can know that the system is running at capacity, but not the extent of the overload.

When the system becomes so busy that logons become slow or hung, performance monitoring is not able to determine whether the system is actually hung or simply overloaded without using other tools.

Suggested Monitoring Techniques

Since the goal is to be able to drive the system as hard as possible without overloading it, some techniques for assessing the "level of busy" can be used:

When CPU usage is high:

1. Check AWT utilization (use puma -c). If the number is constantly at or near maximum, then
2. Check the message flow control (use puma –m). If there are tasks apparently in flow control, then
3. Check UNIX run queue (sar –q). If the run queue is growing longer and longer, the system is too busy and will slow down dramatically.

See "System Activity Reporter" on page 6-4 and "puma (NCR UNIX)" on page 6-16.

While it is not unusual to see a busy system with high AWT counts (the total is 80 per AMP), if flow control is widely present, it means that some tasks are currently being blocked from sending more work to the AMPs who are very busy. If the run queue is growing longer and longer, it means that more work is in the CPU run queue, and each process will have to wait longer in turn before it is again switched in.

High I/O Wait

NCR recommends configurations with CPU to I/O bandwidth ratios according to typical Teradata workload demands in order to avoid CPU starvation. If these guidelines are not followed, or customer workload is unusually heavy with respect to I/O, it is possible that CPU starvation may still occur, as reflected by high I/O WAIT on the system.

If, on the one hand, ResUsage data shows that I/O Wait is rising while CPU Busy is falling, it may mean that the system is not as busy as it was, and there are more CPU cycles available. On the other hand, it may mean that there is indeed an I/O bottleneck that must be investigated.

If the onset of high I/O wait is sudden:

1. Determine if the high I/O wait is due to disk I/O, to waiting for other nodes (because of skewing or coexistence imbalance), to low system demand, or (least likely) to BYNET I/O.

   CPU+WIO less than 90% may suggest low system demand without a true I/O bottleneck. Look at node efficiency to determine if the I/O wait is due to node waiting.

2. Look at the actual disk I/O wait queue using sar –d, and examine:
   - avwait, - average time in ms. that transfer requests wait idly on queue for response (in the SCSI driver queue or in the disk's queue)
   - avserv, - average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

If I/O Wait and the disk I/O queue length are both increasing, it indicates a true I/O bottleneck.

System Activity Reporter

Introduction

The System Activity Reporter (sar) is a command-line utility that allows you to monitor hardware and software information for a system running UNIX. It is a node-local, generic UNIX tool, providing reports that are applicable to Teradata.

Note: The xperfstate utility displays the similar information for reports applicable to Teradata. For a description of differences, see "sar and xperfstate Compared" on page 6-10. For more information on sar and its options, refer to the UNIX man page.

sar Reports

This report...	Displays...
CPUs	CPU utilization for all CPUs at same level of detail as xperfstate (i.e., idle, user, system, I/O wait). Note that the manpage refers to this as processor utilization.
Buffers	all buffer activity, including buffer access, transfers between buffers, and cache hit ratios.
Block device	disk and tape drive activity. Disk activity is the same information you get using xperfstate.
Swapping and switching	• the number of transfers, and units transferred for swapping in and out. • a report of process switches.
Queue length	the average-run queue length while occupied and percent of the queue that is occupied for processes in memory that can be run.
Paging	all paging activities, including page-in and page-out requests, allocated pages, pages available for use, page faults, and pages-per-second scanned. Note that the generated report may be misleading.
Kernel memory	the allocation of the memory pool reserving and allocating space for small requests, including the number of bytes to satisfy the request.

CPU Utilization Example

For example, if you enter:

```
sar -uD
```

where:

Option	Description
-u	CPU utilization
-D	percent of time for system use classified as local or remote

you might see the following:

```
00:00:01      %usr     %sys     %sys     %wio     %idle
                       local    remote
01:00:00      42       44       0        1        13
02:00:00      38       44       0        2        16
03:00:00      36       39       0        2        23
04:00:00      40       35       0        2        24
05:00:00      38       37       0        2        23
06:00:00      38       38       0        2        22
07:00:00      39       36       0        1        24
08:00:00      38       37       0        1        23
08:20:00      39       36       0        1        23
08:40:00      40       35       0        1        24
09:00:00      37       35       0        2        26
09:20:00      39       39       0        1        21
09:40:00      39       37       0        1        23
10:00:00      34       37       0        1        27
10:20:00      40       37       0        2        22
10:40:00      38       36       0        2        24
11:00:00      35       35       0        2        28
11:20:00      40       37       0        1        21
11:40:00      38       37       0        2        23
12:00:00      38       38       0        1        22
12:20:00      37       36       0        2        25
12:40:00      40       36       0        1        23
13:00:00      40       36       0        1        22
13:20:00      37       35       0        2        26
13:40:00      40       3        0        2        23
14:00:00      40       38       0        1        22
14:20:00      35       35       0        1        28

Average       38       38       0        2        22
```

where the meaning of each column is as follows:

Column	Description
%usr	Percent of time running in user mode
%sys local	Percent of time servicing requests from local machine
%sys remote	Percent of time servicing requests from remote machines

Column	Description
%wio	Idle with some process waiting for block I/O
%idle	Percent of time the CPU is idle

Queueing Example

If you enter

```
sar -q -f /var/adm/sa/sa09
```

you might see the following:

```
00:00:00   runq-sz    %runocc
08:00:00     1.3         23
08:20:00     1.5         14
08:40:00     2.1         33
09:00:00     2.6         97
09:20:00     2.6         84
09:40:0      6.2        100
10:00:00     6.6        100
10:20:00     7.3        100
10:40:00     7.0        100
11:00:00     7.5        100
```

where the meaning of each column is as follows:

Column	Description
`runq-sz`	The average number of processes queued up to run on CPU during the sample period.
`%runocc`	The average amount of time the runq processes queued during the sample period.

Note: sar no longer reports statistics for the following two columns: swpq-sz and %swpocc

It is normal for the runq-sz can be approximately 100 on a very busy system. If it begins to reach 150, or if it is constantly growing, it should be investigated. The %runocc will always be about 100% on a busy system.

The -f option indicates the sar -q will run against the sa09 file rather than today's current sar data. sar is continuously running and, at the end of each day (24:00), the sar file is truncated and moved to the /var/adm/sa directory with a name reflecting the day of the month. The sa09 file contains sar data for the 9th day of the current month.

Running a sar -q with no other options will return today's queue data.

Paging Example

For example, if you enter:

```
sar -p
```

where `-p` specifies paging activity, you might see the following:

```
00:00:01    atch/s  pgin/s   ppgin/s   pflt/s    vflt/s    slock/s
01:00:00    0.37    7.11     7.11      15.55     754.85    37.12
02:00:00    0.23    1.44     1.44      9.59      806.68    5.51
03:00:00    0.22    1.58     1.58      11.10     710.26    7.05
04:00:00    0.21    1.91     1.91      8.65      713.90    7.63
05:00:00    0.18    2.11     2.11      10.33     782.40    8.22
06:00:00    0.22    1.87     1.87      10.86     778.99    8.44
07:00:00    0.23    1.84     1.84      9.82      739.00    8.08
08:00:00    0.23    2.06     2.06      11.08     757.08    9.07
08:20:00    0.09    1.32     1.32      10.05     731.34    5.42
08:40:00    0.07    0.84     0.84      8.30      712.66    4.19
09:00:00    0.45    3.41     3.41      11.12     707.32    15.34
09:20:00    0.09    1.34     1.34      10.70     826.07    4.94
09:40:00    0.07    0.75     0.75      9.95      782.06    3.25
10:00:00    0.48    3.37     3.37      13.77     736.29    14.88
10:20:00    0.10    1.38     1.38      8.61      799.21    4.59
10:40:00    0.09    1.54     1.54      9.89      746.47    6.47
11:00:00    0.44    2.31     2.31      12.58     709.10    12.38
11:20:00    0.15    1.83     1.83      8.40      811.60    5.53
11:40:00    0.07    0.78     0.78      9.66      766.23    3.91
12:00:00    0.05    0.71     0.71      10.30     808.77    3.16
12:20:00    0.51    3.21     3.21      12.09     727.58    14.58
12:40:00    0.07    0.79     0.79      9.13      753.26    3.60
13:00:00    0.06    0.72     0.72      9.72      762.37    3.02
13:20:00    0.52    3.31     3.31      12.10     716.53    14.68
13:40:00    0.08    1.28     1.28      9.25      716.28    5.55
14:00:00    0.07    0.74     0.74      10.76     768.04    3.15
14:20:00    0.45    3.11     3.11      12.89     696.81    14.41

Average     0.22    2.15     2.15      10.70     753.67    9.68
```

where the meaning of each column is as follows:

Column	Description
atch/s	Page faults/second that are satisfied by reclaiming a page currently in memory (attaches/second)
pgin/s	Page-in requests/second
ppgin/s	Pages paged-in/second
pflt/s	Page faults from protection errors/second (illegal access to page) or copy-on-writes
vflt/s	Address translation page faults/second (valid page not in memory)
slock/s	Faults/second caused by software lock requests requiring physical I/O

xperfstate

The xperfstate utility displays hardware/software information for a system running UNIX with PDE. It provides multiple real-time views of system resources, including:

This resource...	Provides...
System	a summary of all nodes in the system (TPA and non-TPA).
Clique	a summary of all nodes in a specific clique (nodes that share access to the same disk arrays).
Cabinet	a summary of all nodes in a specific cabinet (generally there are two nodes in each 5100 cabinet).
Node	a summary of a specific node.
Component (CPU or disk)	data for a specific CPU or disk. (Other components are node components, as they are shared resources.)

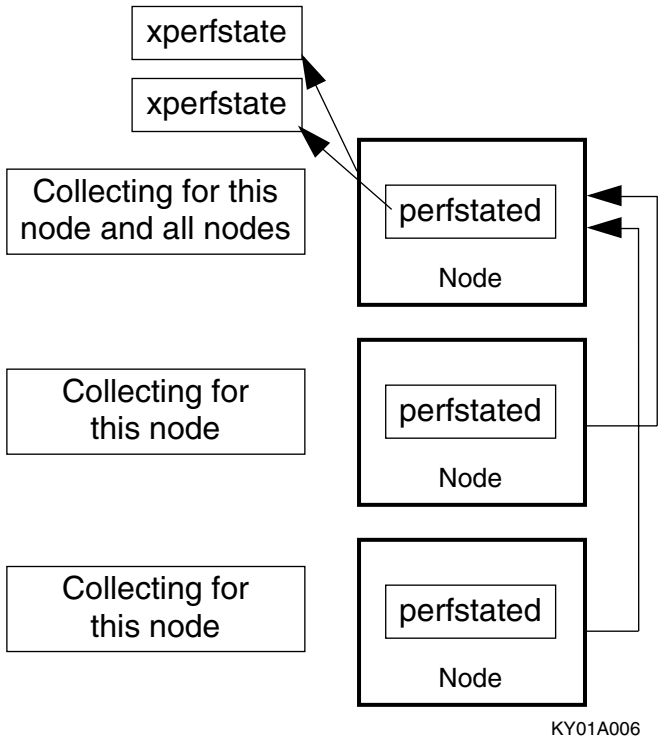
For more information on xperfstate, see *Teradata RDBMS Utilities*.

Impact on CPU Utilization

When you use the xperfstate utility:

Stage	Process
1	xperfstate starts a perfstated daemon on each node.
2	Each perfstated daemon collects data for its node.
3	All perfstated daemons send their data to the control node perfstated.
4	The perfstated daemon on the control node sends the data to all instances of xperfstate.

The following figure illustrates this process.



The xperstate utility consumes very little CPU time. The effect on the system is negligible.

The perfstated daemon consumption on the control node increases as the number of nodes increase, but very little CPU is used.

The number of open windows influences the amount of CPU consumed for xperfstate. Each open window on the controlling screen consumes a certain amount of CPU time just to update the data. Many xperfstate windows that are open and updating concurrently consume more CPU time than only one or two open windows. On systems with many nodes, the number of open windows is limited by the size of the screen.

The effect on system performance of many windows updating data is still negligible; however, it is a good practice to keep open only those windows that you need.

sar and xperfstate Compared

For CPU utilization, I/O activity, and memory issues such as aging, dropping, allocation, +and paging, you can use the sar and xperfstate utilities. (For PDE and UNIX MP-RAS usage data, see "Resource Sampling Subsystem Monitor" on page 2-42).

The output and information provided by each tool are compared in this section.

sar

The sar utility can produce snapshots of CPU activity that are useful for quick troubleshooting. Output is logged and can be displayed or printed.

You can use sar to obtain one or more of the following:

- Real-time snapshots at user-defined intervals
- Historical data stored as a binary flat file

Because you can set log times for intervals as short as 5 seconds, you can obtain snapshots in a stream of near-real-time information. The output options include:

- Display or print
- Summary reports
- Columnar output
- Predefined formats

xperfstate

Use the xperfstate utility if you prefer graphics to numbered lists. Xperfstate does not store data and is not suitable for historical reports. Xperfstate produces:

- Real-time data only
- Graphical display only

sar, xperfstate, and ResUsage Compared

Tabular Comparisons

The following table compares the types of information provided by sar, xperfstate and ResUsage.

Information	sar	xperfstate	ResUsage
CPUs	General CPU usage (system, user, idle, I/O wait)	General CPU usage (system, user, idle, I/O wait)	(SPMA/SCPU) CPU and node level aggregation, same as xperfstate. (SVPR) Breakdown of CPU usage for console utilities, session control, dispatcher, parser, AWT, and startup.
Buffers	Access, transfers, hits	N/A	(IPMA) Secondary cache access and misses.
Block device	LUN information	LUN information	(SLDV) I/O traffic, response time, and outstanding requests for SCSI devices related to vprocs.
TTY device	Available	N/A	N/A. Not applicable for Teradata systems.
System calls	Available	N/A	N/A. Not applicable for Teradata systems.
Swapping and switching	General swapping & switching activity	Process switches	(ICPU/IPMA) Interrupted and scheduled switching. (SVPR) Swapping.
Queue length	Run queue length and percent of queue used	Run queue length, and message queue length	(SPMA) Blocked and pending processes.
Non-Teradata RDBMS file access routines	Available	N/A	N/A. Not applicable for Teradata systems.
Process and i-node	Available	N/A	N/A. Not applicable for Teradata systems.

Information	sar	xperfstate	ResUsage
Message and semaphore	Available	N/A	N/A. Not applicable for Teradata systems.
Paging	All paging activity	Page Faults	(SVPR) context pages, paged in/out.
Memory	Kernel memory	File System Segments (FSG) cache size	(SPMA) Memory allocation in general, specific to vprocs, and backup node activity. Memory problems, including failures, aging, dropping, and paging. (SVPR) Memory allocation and memory resident with respect to vprocs.
BYNET	N/A	Various BYNET info	Various BYNET information in SPMA, IPMA, SVPR, IVPR.
Client	N/A	N/A	(SHST) Host and gateway traffic and management.
Teradata File System	N/A	N/A	(SPMA/SVPR/IVPR) General file system information.
Cylinder management	N/A	N/A	(SVPR) Cylinder events; migrates, allocations, mini-cylpacks, defrags. (IVPR) Overhead for the above events.
Database Locks	N/A	N/A	(SPMA/IPMA/SVPR) Database lock requests, blocks, and deadlocks. Note that All data collected in SVPR, IVPR, SLDV, and SHST is associated with a vproc providing detailed information not available with sar.

TOP

TOP, delivered in the GSCTOOL package reports the top processes consuming resources on a node.

Example

The following is the output from

```
top –n:
last pid: 0; load averages: 12.14, 8.66, 4.76 22:41:39
1210 processes:1209 sleeping, 1 on cpu
Memory: 213M swap, 879M free swap
PID USERNAME PRI NICE SIZE RES STATE TIME WCPU CPU COMMAND
254 root 49 0 612K 396K sleep 0:00 19.0% 7.42% actspace
8094 root 50 -20 790M 1152K sleep 0:04 8.0% 3.12% actspace
262 root 27 0 3360K 1116K cpu 0:00 7.0% 2.73% top
6094 root 48 -20 790M 1172K sleep 0:03 6.0% 2.34% actspace
7094 root 50 -20 790M 1156K sleep 0:04 5.0% 1.95% actspace
9094 root 50 -20 790M 1140K sleep 0:04 5.0% 1.95% actspace
```

BYNET Link Manager Status

You can use the BYNET Link Manager Status (blmstat) utility to troubleshoot BYNET problems.

Enter the following command to start the blmstat utility:

```
blmstat -qv | grep BrdActive
```

Interpreting the Results

If the output is approaching or exceeding 40%, the BYNET is saturated due to broadcast messages. Saturation due to broadcast messages causes slow point-to-point messages, which causes the entire system to slow down.

Notes

You can use the blmstat utility to find many more statistics on the BYNET. However, you must understand the internals of the BYNET to fully interpret blmstat output.

Example Output

Following is example output of the blmstat utility.

```
BrdActive%  10  BrdActive% 10
```

ctl and xctl

ctl (Windows 2000)

The ctl utility is a tool that allows you to display and modify fields of the PDE GDOs.

You can use ctl from one of the following:

- ctl window, Teradata Command Prompt, or Command Line
- Teradata MultiTool

xctl (NCR UNIX)

The xctl utility is an X window system-based tool that allows you to display and modify the fields of the PDE GDOs.

You can use xctl from one of the following:

- Non-windowing mode
- Windowing mode

For information on ctl and xctl, see *Teradata RDBMS Utilities*.

puma (NCR UNIX)

Introduction

The puma utility can be used to look at information about the system as it relates to performance.

Two useful puma commands are described:

- puma -c
- puma -m

puma -c

Returns the number of AWTs in use at the time of sampling as a list of worktypes for each vproc on the node, including PE vprocs.

PE vprocs are numbered starting at 16383. AMP vprocs start with the number 0.

In the following example, vproc 0 is displayed:

```
VPROC = 0
WorkType   Min Max Inuse  Peak
MSGWORKNEW 3   999  39     50 <= step sent by dispatcher
MSGWORKONE 3   999  23     23<= work spawned by an AMP on
                               behalf of another AMP
```

where:

Column	Definition
WorkType	The type of work that the AWTs are doing
Min	The minimum number of AWTs dedicated to this Worktype
Max	The maximum number of AWTs dedicated to this Worktype
Inuse	The number in use at the time of sampling
Peak	The highest number actually used since the last restart

In the first two rows, the Inuse column of MSGWORKNEW (also known as MSGWORKZERO) and MSGWORKONE cannot add up to more than 62. In the example above, VPROC 0 is at 62, which is the maximum.

puma –m

Returns the message queues, known as "message mail boxes," passed between Teradata processes on a node.

Message Mail Boxes

There are two types of message mail boxes:

- Well-known
- Transient

If the system becomes too flooded with messages in either type of mailbox, the task sending to the mailbox will be put into flow control by the messaging subsystem and no more messages will be allowed to be sent by the task until the MsgCount on the receiving message queue falls below 20 (default).

Note: Not all message senders are subject to flow control. For example, Dispatcher-sent messages are, but abort-sent messages are not.

On a very busy system, there are probably always some processes in flow control. This is not in itself necessarily a problem. What is a problem is having processes stuck in flow control and having no or few messages being processed on that mailbox.

Mailbox 2-11

Although there are many mailboxes, "well-known" mailboxes 2-11 (the primary work mailboxes for AWTs), are the mailboxes "of interest." That is, there may be other processes/tasks in flow control, but 2-11s are the ones that affect messages being sent by the dispatcher to AWTs. There should be one 2-11 mailbox for every AMP on the node.

Having other mailboxes with counts at, or greater than, the MsgCount flow control value may indicate that flow control is occurring in the messaging subsystem. For the non-AMP related mailboxes, this may mean that a specific utility, say, is in flow control, but not affecting new work.

Example

Here is an abbreviated example from a V2R4.1 system:

```
MbxId Part Owner MsgCount MbxName Flags MsgDQCount
11    11   7008  28       2-11    Work  61107
```

where:

Column	Definition
MbxId	Mail Box ID
Part	Internal Partition (as seen in Teradata Manager PMON Partition field when clicking on a session)
Owner	UNIX Process ID of the process that spawned the mailbox
MsgCount	The number of messages on the queue at the time of sampling
MbxName	The "well-known" mailbox ID. If blank, it is a "transient" mailbox.
Flags	The value Work is an indication that the mailbox is a special type of queue, with a group of many processes attached to it. The messages are in a priority order, and the process pulling the next message off the queue will run at the CPU priority ($M, and so on)) of the session for whom the message was sent.
MsgDQCount	The number of messages that have been pulled off the queue and processed since the last restart

The flow control value is normally automatically set to the number of nodes + 5 or the default of 20, whichever is greater, unless the value was tuned manually. In that case, it will not be automatically overwritten.

In the above example, the AWTs are in flow control at the time of this sample on this node.

The Flow Control Value

Use the following command to determine the flow control value. This example shows the default of 20:

```
# /usr/ntos/bin/tdntune -T | grep INITVAL
FCIN_P2PCHN_INITVAL 20
FCIN_P2PGRP_INITVAL 20
FCIN_GRP_INITVAL 20
FCIN_CHN_INITVAL 20
FCOUT_INITVAL 5
```

In the above example, all the FCIN* parameters are set to 20, indicating that the system on which tdntune was run has a flow control threshold of 20.

MsgCount

To see the MsgCount (that is, the number of messages on the queue at the time of sampling) of all 2-11 AWT mailboxes on all nodes, execute the following:

```
# rallsh -sv "/usr/ntos/bin/puma -m | grep ' 2-11 '" /* check for
MbxName 2-11 on all nodes */
```

Sample output:

```
Getting hosts for network: byn0
Using hosts: byn001-4 byn001-5
==== /usr/bin/rsh byn001-4 /usr/ntos/bin/puma -m | grep ' 2-11 '
====
12 11 6008 19 2-11 Work 382332
9 11 7006 19 2-11  Work 378904
9 11 10006 19 2-11 Work 388045
9 11 11006 18 2-11 Work 377668
9 11 14006 17 2-11 Work 400221
9 11 15006 20 2-11 Work 440050
9 11 18006 22 2-11 Work 384192
9 11 19006 19 2-11 Work 379054
9 11 22006 20 2-11 Work 377361
9 11 23006 19 2-11 Work 385318
==== /usr/bin/rsh byn001-5 /usr/ntos/bin/puma -m | grep ' 2-11 '
====
10 11 8007 19 2-11 Work 380607
9 11 9006 17 2-11  Work 400102
9 11 12006 20 2-11 Work 378202
9 11 13006 18 2-11 Work 440373
9 11 16006 19 2-11 Work 437675
9 11 17006 20 2-11 Work 376121
9 11 20006 19 2-11 Work 380059
9 11 21006 19 2-11 Work 380890
9 11 24006 18 2-11 Work 443176
9 11 25006 19 2-11 Work 433609
```

In the above example, five of the 20 AMPs are in flow control and all the others about to go into flow control. If after several iterations of running the rallsh command, the same mailboxes are still at or above 20, and the MgsDQCounts for each have not increased, then the system is badly congested.

The reason why the MsgCount for 2-11 mailboxes could be greater than the actual *INITVAL flow control threshold is that not all message senders, such as ABORT tasks, are subject to flow control.

MsgDQCounts

The following shell script adds up the MsgDQCounts for the puma –m output:

```
puma -m 2-11 | grep 2-11 | awk '{tot+=$NF} END{print tot}'
```

tdnstat

tdnstat provides another way to verify the extent of flow control. tdnstat is an internal utility that the TGSC system canary hang.sh script calls.

The tdnstat BLKQ (block queue) lists the number of messages that are blocked currently in flow control:

```
BNS BlockQueue Statistics :
BLKQ BLK_CNT 146
BLKQ UBLK_CNT 146
```

There is always a BLKQ count greater than zero, even on an idle system. If this number increases during a period when a check of the MsgDQCount shows that messages are not being processed, the system is in "serious flow control and stuck."

Usage Consideration

If there had been only one mailbox at or above 20 for several iterations, it indicates that:

- Only one AMP is in flow control, and
- The AMP is likely a hot AMP and could be caused by skewed data in the table or in spool.

Look for the actual hot AMP with puma –c.

In general, a heavily congested system that is also saturated (out of AWTs on some or all AMPs) will display many 2-11 mailboxes in flow control.

To see any mailbox on all nodes that do not have a MsgCount of zero, execute the following command:

```
#rallsh sv /usr/ntos/bin/puma m | grep ‘ 0 ‘
```

To see any mailbox on all nodes that have a MsgCount of 20, execute the following command:

```
#rallsh sv /usr/ntos/bin/pum m | grep ‘ 0 ‘ | grep ‘ 20 ‘
```

Resource Check Tools

Introduction

Resource Check Tools enable you to monitor system resources.

Basic Resource Check Tools

There are standard resource checks, and ones that are customizable. The basic components are:

- Dbscheck identifies a slowdown or hang of the Teradata DBS, like a canary query.
- Nodecheck provides system data to be used by syscheck for a node
- Syscheck, a system-wide tool, spawns an instance of nodecheck on all live nodes.

  Syscheck compares the results from nodecheck against threshold values from the user-editable file, syscheckrc. Syscheck echoes warning or alarm messages if certain thresholds are exceeded or fall below user-specified levels.

  Syscheckrc is resource file containing thresholds and alert levels.

See *Teradata RDBMS Utilities* for more detail on these tools.

Chapter 7:

Optimization through Workload and Session Management

This chapter discusses performance optimization through workload and session management.

Topics include:

- Priority Scheduler
- SET SESSION ACCOUNT and performance
- Accessing Priority Scheduler
  - Priority Scheduler Administrator
  - schmon
  - xschmon
- Priority Scheduler Simulator
- Job mix tuning
- Identify and handling "killer queries"
- Teradata Dynamic Query Manager (TDQM)
- Client-specific monitoring and session control tools
- Session processing support tools

Priority Scheduler

Introduction

You can control the system resources allocated to your sessions, processes, and tasks using the Priority Scheduler.

Priority Scheduler consists of the following three-tiered control structure to group user sessions for scheduling purposes:

- Resource partition
- Performance group
- Allocation group

Each of these components includes parameters that relate them to each other. You define and associate these components to design an environment that favors or inhibits the performance of individual sessions based on their logon account attributes.

Because Priority Scheduler can:

- Control resource sharing among different applications
- Automate changes in priority by time or day or by amount of CPU used
- Place a ceiling on system resources for specific applications,

it can manage resource distribution to improve performance of one application at the expense of another.

For a detailed description of Priority Scheduler, including on how to use it, see "Priority Scheduler" in *Teradata RDBMS Utilities*.

Components of Priority Scheduler

The following table describes the components of Priority Scheduler.

Component	Description
Resource Partition	A collection of prioritized performance groups. A resource partition can limit the total amount of CPU used by sessions assigned to its performance groups. Priority Scheduler provides resource partition zero (0) as the default partition. You can define four additional resource partitions. Each resource partition carries a weight that will be compared to other resource partition weights.
Performance Group	The name of the performance group, defined within each additional resource partition, matches an entry in the: • Column named AccountName in the USER record • accountid string in the logon statement. The name of each performance group must be unique.
Performance Period	A performance period links a performance group to an allocation group. A performance period allows you to change allocation group assignments based on time-of-day or resource usage. You must define at least one performance period, but you can have up to five performance periods per performance group.
Allocation Group	An allocation group determines the scheduling policy and set division type. An allocation group carries a weight that will be compared to other allocation group weights. Several performance groups within a single resource partition may share the same allocation group.

Priority Scheduler and the Default Resource Partition

If your system reaches the point that CPU saturation is a frequent problem, it is time to consider implementing Priority Scheduler to move the non-administrative users out of the default partition.

Even if you are not ready to implement a complex prioritization of the user workload, CPU resource bottlenecks can be handled more effectively just by moving everyone to a new partition with the same priority.

By default, all user accounts run at $M in the default partition. When the system becomes very busy, resource contention can occur in the default partition and is further exacerbated when user jobs are raised in priority in the default partition.

Never raise the priority of a user job in the default partition to $R. Use caution in raising these to $H. Never raise a large, complex job to $H in the default partition, as the impact could be unexpectedly severe. Moving all non-admin users out of the default partition and weighting the partitions in favor of default (RP0) will ensure that the system and background tasks that are trying to service the user jobs can execute as quickly as possible.

Tasks in the Default Resource Partition

The following table illustrates how tasks running in the default resource partition (RP0) might be defined in Priority Scheduler:

Task	Priority	Notes
Transaction and system recovery, including writing crashdumps	Rush$	Level of CPU used will vary depending on the workload and the recovery frequency
ABORT processing	Rush$	If DBScontrol ROLLBACKPRIORITY = true, then will run at user's priority
Dispatcher	Rush$	Most other PE work runs at user's priority
Session control	Rush$	All PE work up to and including logon, password validation. Once the user is identified and authenticated, session runs at user priority

Task	Priority	Notes
Dictionary cache manager in the PE	Rush$	This accounts for a lot of the activity in the default partition and will be greater the more tables, columns and access rights being utilized in the system. It is also one of the activities that can slow down user requests if it becomes the bottleneck. This accounts for a lot of the activity in the default partition and will be greater the more tables, columns and access rights being utilized in the system. It is also one of the activities that can slow down user requests if it becomes the bottleneck.
Deadlock detection	Rush$	Global deadlock detection and resolution
Recovery Manager utility	$L	Can be changed to $M or $H
Table Rebuild	$M	Can be changed to $L or $H
ResUsage Row Insert	Rush$	Collection done by PDE RSS daemon
Error Logging	Rush$	
PM/PC and all MONITOR RESOURCE/SESSION commands	Rush$	
showspace display of all space used	Low$ and med$	Part of this runs at Low$, and part of Med$
Filesystem utilities	$M	Can change to any other priority ($L, $H, $R)
Release of spool files	Rush$	
System startup	Med$ and rush$	
Checktable	Med$	
Mload permanent journal update	Rush$	

Task	Priority	Notes
HUT processes writing to data dictionary, event tracking	High$	
Purging transient journal	Varies	Runs in performance group 0-7, depending on level of update activity, and how often the purge program is called. It raises and lowers its own priority.

Usage Considerations

When defining the priority levels, weights, and policies for your workloads, consider the following:

- Tailor resource partitions with different weightings to suit different applications and users. For example, the more important the application, the more weighting (that is, the more CPU resources) you should assign to it.
- Within each resource partition, you can cause the weightings to vary at different times by changing the allocation group.
- If you set the scheduling set to Session, the system distributes resources equally among sessions instead of tasks; therefore, it gives all tasks in a session (including multi-thread jobs with parallel steps) *collectively* the same total resources as another session that might have only one single-thread task.
- Using the calculated Global Weights as a guide, modify the resource partition weights to insure that the DEFAULT resource partition (RP0) maintains a higher weight value, regardless of other active or inactive resource partitions or performance groups.

  This is especially important for the RUSH priority within RP0. Recommended weight ratio between RP0 and next highest-weighted partition is 2 to 1.
- Where possible, maintain user activity to a division other than HIGH or RUSH within RP0. Reserve these divisions exclusively for system maintenance and support activity.
- The Priority Scheduler will round fractions to whole numbers, so for each of the Performance Groups try to maintain a Global Weight spread in whole numbers.
- When you have a choice, tend toward fewer numbers of resource partitions and performance groups, especially if the divisions would have few assigned users, this will smooth out fluctuations in response times.
- Maintain development or test users in a separate allocation group and possibly a separate resource partition.

- Assign unique login ids to each user and group user types and/or job type by account string.
- Use the Account String Expansion (ASE) to identify the actual resource allocation for each of the resource partitions over time rather than using the "snapshot" view provided by the schmon utility.
- In addition, use ASE to differentiate the characteristics of the various work groups within the allocation groups. Adjust the Policy and Type as appropriate for the characteristics.
- Once the resource allocation is acceptable at the resource partition level, adjust the Global Weights of the allocation groups.
- Defer the use of "advanced features" such as MILESTONE limits initially.
- Be careful with the use of ABSOLUTE and RELATIVE policies. These policies have ceilings and will limit the users of such groups even if there is no one else on the system.
- Note that once created, a partition can be modified but not removed with schmon.

  The only way to completely remove a partition once it is created is to remove the system.gdo file and restart/reboot. If this is desired, call Support – do not try this without engaging TCSE.
- It is highly recommended that if the user changes any Priority Scheduler settings, he or she set up scripts that use the schmon commands to define them. All Priority Scheduler settings are contained in the /ntos/system.gdo and if that file is destroyed, it is recreated using only default settings. All user-defined settings are lost.

SET SESSION ACCOUNT and Performance

Introduction

On Teradata for UNIX, the ACCOUNT option of the SQL SET SESSION statement lets you dynamically change the performance group of your current session or request.

You can also embed one or more SET SESSION ACCOUNT statements in a BTEQ script or application program. With embedded SET SESSION ACCOUNT statements, you can manipulate the priority of the entire session or of one or more requests.

For example, if the session is running under a $H account, you can lower a request to $M, raise the next request to $H, lower the next to $L, and raise the next to either $M or $H.

Defining ACCOUNT Strings

You use SET SESSION ACCOUNT to modify your own session or request. The ACCOUNT clause of the CREATE USER and MODIFY USER statements lets you define the user logon environment. The specification can include an identifier (account_id) of the account to be charged for the resources consumed during a session.

Other facilities are available with which to control the session of other users (see "Changing the Priority of a Current Account" on page 7-8).

You can define multiple accounts. The account charged for a session depends on the account_id you specify when logging on. If you do not specify an account, the session runs under your user name.

Including Performance Group Names

You can include a performance group name in the account string prefix. The performance group weighting determines the percentage of computing resources Teradata allocates to sessions initiated under that user or account_id.

You must have already created the name and partition using the *schmon* utility of the Priority Scheduler Facility. A performance group name associates the account_id and its performance group with a resource partition.

Changing the Priority of a Current Account

A session always runs under a performance group. The current performance group of any session is one of the following:

- The performance group of the user name or account_id under which you logged on to this session.

  Note: If you have no account_id or did not specify one, the session runs under your user name. If a performance group is not defined for your user or account, the system automatically assigns the default (medium) performance group.
- The performance group you modify dynamically by issuing a SET SESSION ACCOUNT...FOR [SESSION/REQUEST] statement during the current session.

  At the session level, the performance group is changed for the remainder of your session or until another SET SESSION ACCOUNT is executed.

  At the request level, the performance group is changed only for the request executed after SET SESSION ACCOUNT; after that, the previous group resumes.
- The performance group you defined by embedding SET SESSION ACCOUNT statements in a program or BTEQ script.
- The performance group imposed dynamically by the database administrator on another user's:
  - Session, via Teradata Manager PMON session control
  - Request or session, via PM/API SET SESSION ACCOUNT

You can modify the current performance group assignment in the following ways:

You can change..	At this level...	With this tool...	Using...
your own account	• Session • Request	Teradata SQL	Static: • MODIFY USER statement
			Dynamic: • SET SESSION ACCOUNT='acct_id' FOR [SESSION/REQUEST] statement
another user's account (if you have the CREATE USER or SESSIONABORT privilege)	• Session	Static: • Teradata SQL	MODIFY USER statement
		Dynamic: • Teradata Manager, PMON • PM/API	• "Modify a user account/priority string" facility • SET SESSION ACCOUNT request
	• Request	Dynamic: • PM/API	SET SESSION ACCOUNT request

Accessing Priority Scheduler: Priority Scheduler Administrator, schmon, and xschmon

Introduction

The following utilities provide access to Priority Scheduler settings.

Priority Scheduler Administrator

Priority Scheduler Administrator (PSA), a Teradata Manager (TM) application, provides a graphical interface to Priority Scheduler.

Using PSA, you can define Priority Scheduler configurations and to observe scheduler performance to Teradata Manager users. Unlike schmon, PSA does not require root privileges.

See "Using Priority Scheduler Administrator to Balance the Workload" on page 11-17.

For information on Priority Scheduler Administrator, see *Teradata Manager User Guide*.

schmon Utility

The schmon utility provides a command line interface to Priority Scheduler. schmon allows you to display and alter Priority Scheduler parameters.

For a detailed description of the schmon utility, including how to use it, see "Priority Scheduler Facility" in *Teradata RDBMS Utilities*.

xschmon Utility

Like schmon, the xschmon utility allows you to display and alter Priority Scheduler parameters.

The xschmon utility is a graphical user interface X-window system that uses the OSF/Motif Toolbox to manage its windows resources.

For a detailed description of the xschmon utility, including how to us it, see "Priority Scheduler Facility" in *Teradata RDBMS Utilities*.

Priority Scheduler Simulator

Introduction

Priority Scheduler Simulator (PSS), a Microsoft Excel spreadsheet, allows you to generate “what-if” situations with different priority attributes.

PSS allows you to emulate Priority Scheduler weighting configurations to determine expected relative weights for various activity situations.

Example

To determine, for example, the actual allocation of CPU in real-world situations, you need to take into account any unseen loads across the system that might not be explicitly listed.

Unseen loads include gateway and streams activity, interrupt service, and non-Teradata applications.

Job Mix Tuning

Introduction

As a system reaches capacity limits, it may become necessary to apply resource management to the workload in order to be able to service all requests with reasonable expectations. This may be true even if the capacity issues only apply to short periods of peak usage during the prime hours.

CPU Resource Utilization

As CPU resource utilization is generally the binding factor on most Teradata systems with a configuration that is correctly balanced between CPU and disk I/O, it may become necessary to take steps to affect the overall CPU availability.

CPU busy can be determined from a historical perspective. However, it also a good practice to check the system for high CPU-consuming user jobs. This may be an opportunity for application tuning of some type, or provide evidence of a bad execution plan caused by stale statistics or an Optimizer bug. Use Teradata Manager's PMON to view the running SQL and EXPLAIN for an active session.

Steps in Tuning the Job Mix

If there are no apparent "killer queries" having an obvious impact on the system, but the system itself is showing signs of CPU saturation, then it is time to consider workload grouping and job mix tuning.

The basic steps to tuning job mix are as follows:

1. Establish work groupings. These groupings can be based on functionality, department, query type, or complexity.
2. Establish account identifiers with account string expansion. Tie the user accounts to specific workgroups.
3. Decide the priority by business criticality and response time. There may be high priority users from a business point of view who do work that does not require a specific response time, as well as "less business critical" users who run a series of short queries.

   It may be best to give the short-running queries the higher priority on the system to ensure that they finish quickly and consistently. This would not necessarily have an impact on the users with the higher business priority if they are running complex, long-running queries.
4. Assess the overall workload to determine if it might be necessary to throttle back work or to apply job-scheduling strategies.

The Priority Scheduler can be used to implement priorities and scheduling strategies to some extent.

TDQM can be used to throttle back workload during peak usage in the prime hours and to capture pent-up demand.

Identifying and Handling "Killer Queries"

Killer queries are high CPU-consuming queries that do a disproportionately low amount of I/Os in comparison to the number of CPU ticks. For example, a product join with high numbers of actual rows in both the left and right tables might become a killer query.

Killer queries can have a bad execution plan for a number of reasons, among them:

- Stale statistics
- No statistics on a skewed, highly non-unique index
- Poor choice of a primary index

Killer Queries and the "Penalty Box"

If killer queries are occasionally present and a “penalty box” is needed, it is a good idea to take the first step and move all users out of the default partition into a user partition, then create a penalty box using the ABSOLUTE policy in an allocation group assigned to a performance group in a third partition.

A penalty box is an allocation group in Priority Scheduler defined with very low CPU allocation. It functions like the "ice hocky penalty box" to "cool off" the consumption of the high-CPU (compute-bound) query by forcibly throttling the amount of CPU it can consume.

Putting a query is the penalty box is an alternative to aborting it. A killer query that has been moved into the penalty box is still be able to run.

Teradata Dynamic Query Manager

Introduction

You can use Teradata Dynamic Query Manager (TDQM) to manage access to the Teradata database resources. Managed accesses increases workload efficiency.

TDQM allows you to create rules for a session based on one or more of the following parameters:

- Date and time.
- None, one, or multiple account strings. Each may include a Priority Scheduler performance group, one or more ASE codes, and one account name.
- The user or group logon ID and the names of target databases and database objects, for example tables, views, stored procedures, and macros.
- The kind of resources involved in the query, such as:
  - Processing time
  - Number of rows returned
  - Joins or full-table scans

Request Scheduling

TDQM provides a set of tools that you can use to schedule single- or multi-statement query requests for execution at a later time.

The Scheduled Request (SR) function is made up of both client and server components. The SR client components submit and monitor scheduled requests, and the SR server piece checks, saves, and executes the requests.

For detailed information on TDQM, see *Teradata Dynamic Query Manager Administrator Guide* and *Teradata Dynamic Query Manager User Guide*.

Client-Specific Monitoring and Session Control Tools

Network Monitoring Tools

You can use the following "monitoring tools" to monitor and control sessions originating from network-attached clients:

<table>
<tr><th>Tool</th><th>Description</th><th>Reference</th></tr>
<tr><td>Gateway Global</td><td>Command-line editing tool you can use to access and modify the fields of the Gateway Control GDO.</td><td rowspan="2">*Teradata RDBMS Utilities*</td></tr>
<tr><td>Gateway Control</td><td>Command-line utility with commands that let you monitor network and session information, such as:
<table>
<tr><th>IF you want to see...</th><th>THEN use...</th></tr>
<tr><td>network configuration information</td><td>DISPLAY NETWORK</td></tr>
<tr><td>all sessions connected via the gateway</td><td>DISPLAY GTW</td></tr>
<tr><td>status information for a selected gateway session</td><td>DISPLAY SESSION</td></tr>
</table></td></tr>
<tr><td>tdnstat</td><td>Command-line utility that can get you a snapshot, or a snapshot differences summary, of statistics specific to Teradata Network Services.
You also can clear the current network statistics.</td><td>*Teradata RDBMS Utilities*</td></tr>
</table>

TDP Monitoring Tools

You can use the following "monitoring tools" to monitor session activity and performance on channel-connected (mainframe) clients:

Tool	Description	Reference
HSI timestamp	HSI (Host System Interface) timestamps tell you when TDP receives a request, when the request parcel is sent to or queued for Teradata, and when the response parcel is received from Teradata.	*Teradata TDP Reference*
TDPUTCE	TDPUTCE (TDP User Transaction Collection Exit) is a routine that collects statistics about all of the requests and responses controlled by TDP, including user, session/request parcels, timestamps, request type, and request/response parcels. Your site is responsible for developing applications that process and analyze the data collected by TDPUTCE.	
SMF	SMF (System Management Facility) is a mechanism that provides accounting and performance information on MVS, such as: • Statistical information about the processing activity of a PE recorded at shutdown. • Log-off session information, including the use of client and Teradata resources for a session. • Logon violations and security violations records. • Statistical information about the processing activity of the TDP, recorded at shutdown.	*Teradata TDP Reference*

Session Processing Support Tools

Tabular Summary

You can use the following "support tools" to control session processing and in this way optimize system performance:

Tool	Description	See
Logon control	Controls user access to Teradata based on client (host) identifiers and/or passwords.	*Teradata RDBMS Security Administration*
Teradata Manager	Tool that you can use to abort the transaction of a specified session or group of sessions and, optionally, to log off those sessions.	Teradata Manager online help
LOGOFF command (TDP)	Command that forces off one or more channel-connected sessions.	*Teradata TDP Reference*
LOGOFF POOL command (TDP)	Command that ends a session pool by logging off pooled sessions in use by application programs.	*Teradata TDP Reference*
KILL command of Gateway Control utility	With USER or SESSION, forces off one or more network-connected sessions.	*Teradata RDBMS Utilities*
PERM and SPOOL clauses of CREATE and MODIFY USER or DATABASE	Used to allocate permanent and temporary space. Use spool space to limit the results of erroneous queries, such as Cartesian products. If a user group shares tables, you can save space by: • Allocating 0 PERM space to users • Allocating all the table space to a single database • Granting access privileges on the database to all users in the group Be sure each user has enough spool space to accommodate the largest response set he or she is likely to need.	*Teradata RDBMS SQL Reference*
ACCOUNT clause of CREATE or MODIFY USER	Optionally, used to assign one or more account identifiers and/or a user priority, by user or by account.	Teradata RDBMS SQL Reference

Chapter 8:

Performance Tuning Using Query Analysis Resources and Tools

This chapter describes Teradata database query analysis resources and tools that help tune performance through application and physical database design.

Topics include:

- Query analysis resources and tools
- Query Capture Facility (QCF)
- Target Level Emulation (TLE)
- Teradata Visual EXPLAIN
- Teradata System Emulation Tool (TSET)
- Teradata Index Wizard
- Teradata Statistics Wizard

Query Analysis Resources and Tools

You can use the following resources and tools to take best advantage of the query analysis capabilities of the Teradata RDBMS.

Use This Tool...	To...
Query Capture Facility (QCF)	perform index analysis using an SQL interface to capture data demographics, collect statistics, and implement the results
Target Level Emulation (TLE)	replicate your production configuration in a safe test environment
Teradata Visual EXPLAIN	compare results from a query run at different times, on different releases, or with different syntax
Teradata System Emulation Tool (TSET)	capture the complete environment for a specific SQL or database name with everything but the data itself
Teradata Index Wizard	perform SI analysis and offer indexing recommendations, using data captured via QCF and/or DBQL
Teradata Statistics Wizard	collect statistics for a particular workload or select tables or columns or indexes on which statistics are to be collected and recollected

Each of the above is described in the sections that follow:

Query Capture Facility

Introduction

The Query Capture Facility (QCF) allows the steps of query execution plans to be captured. The query text and plans are stored in special relational tables that you create in a user-defined Query Capture Database (QCD). The captured data is used to gather data demographics and analyze indexes for the tables referenced in the queries.

Note: You must upgrade a QCD that was created on a system earlier than Teradata V2R5.0. If the version of a legacy QCD is lower than QCF03.00.00, you also must migrate the data to a new QCD. Once upgraded, QCD can be utilized by the Teradata Index Wizard.

QCF and SQL EXPLAINs

The source of the captured data is produced by the Optimizer, which outputs the text of SQL EXPLAINs detailing the final stage of optimization. Note that the current the data captured by QCF does not represent all the information reported by EXPLAIN.

The captured information is your workload becomes source input to:

- The Teradata Visual EXPLAIN
  See "Teradata Visual EXPLAIN" on page 8-5.
- The Teradata Index Wizard
  See "Teradata Index Wizard" on page 8-7.

For information on QCF, see *Teradata RDBMS SQL Reference*.

Target Level Emulation

Introduction

Target Level Emulation (TLE) allows the TGSC to emulate your production system for the purpose of query execution plan analysis.

Query plans are generated on the test system as if the queries were submitted on the production system. The TLE achieves this by emulating the cost parameters and random AMP samples of your production system on the test system.

Performance Benefits

You can use the tool to validate and verify new queries in a test environment ensuring that your production work is not disrupted by problematic queries.

Caution: TGSC should run TLE on a test system. Do not enable it on a production system.

For more information, see *Teradata RDBMS SQL Reference.*

Teradata Visual EXPLAIN

Introduction

Teradata Visual EXPLAIN client-based utility is an interface for application performance analysis and comparison.

You can use Teradata Visual EXPLAIN to:

- Generate a description of the query processing sequence
- Compare the same query run on different releases or operating systems
- Compare queries that are semantically the same but syntactically different

Performance Benefits

The results can help you understand changes to the Teradata database schema, physical design, and statistics.

For detailed information, see *Teradata Visual Explain User Guide*.

Teradata System Emulation Tool

Introduction

Teradata System Emulation Tool (TSET), a client-based tool, is integrated with Teradata Visual EXPLAIN and Compare (VECOMP) and designed for application developer to simulate production environments on very small or disparate test systems.

If you have a test system with some of the data, you can use TSET to import the production table detailed statistics, TLE data, all DDLs and Random AMP Sampling statistics from the production system to the test system.

Performance Benefits

TSET enable you to:

- Imitate the impact of environmental changes on SQL statement performance
- Provide an environment for determining the source of Optimizer-based production database query issues using environmental cost data and random AMP sample-based statistical data.

For more information TSET, see *Teradata RDBMS SQL Reference*. For information on how to use TSET, see *Teradata System Emulation Tool User Guide*

Teradata Index Wizard

Introduction

The Teradata Index Wizard analyzes SQL statements in a workload, using the contents of the tables in QCD, and recommends the best set of indexes to use.

In particular, Index Wizard helps re-engineer existing databases by recommending SI definitions that should improve the overall efficiency of the workload. Recommendations can include adding or deleting SIs to or from an existing design.

Index Wizard creates workload statements, analyzes them, and then creates a series of reports and index recommendations that show various costs and statistics. The reports help you decide if the index recommendation is appropriate or not.

Index Wizard then validates the index recommendations so you can compare performance with existing physical database design to the recommended physical database design enhancements, that is, recommended indexes. These recommendations can be used to evaluate potential performance improvements and modify the database accordingly.

Performance Impact

Teradata Index Wizard:

- Simulates candidate SIs without incurring the cost of creating them.
- Validates and implements SI recommendations.
- Provides automatic “what-if” analysis of user-specified index candidates.
- Interfaces with the Teradata System Emulation Tool to allow workload analysis on test systems as if the workload had been analyzed on the production system.
- Interfaces with the Teradata Visual EXPLAIN to compare query plans in the workloads.

For information on Teradata Index Wizard, see *Teradata RDBMS SQL Reference*. For information on how to use Teradata Index Wizard, see *Teradata Index Wizard User Guide*.

Teradata Statistics Wizard

Introduction

The Teradata Statistics Wizard assists in the process of collecting statistics for a particular workload or selecting arbitrary tables or columns or indexes on which statistics are collected or recollected.

In addition, the Statistics Wizard permits users to validate the proposed statistics on a production system. This feature enables the user to verify the performance of the proposed statistics before applying the recommendations.

As changes are made within a database, the Statistics Wizard identifies the changes and recommends which tables should have statistics collected, based on age of data and table growth, and what columns or indexes would benefit from having statistics defined and collected for a specific workload.

Performance Benefits

The Teradata Statistics Wizard recommends the collection of statistics on specified tables, columns, or indexes, the collection of which may improve system performance. See "Collecting Statistics" on page 12-53.

For information on Teradata Statistics Wizard, see *Teradata Statistics Wizard User Guide*.

Chapter 9:

Managing Space

This chapter discusses space management in the Teradata database.

Topics include:

- Running out of disk space
- Running out of free cylinders
- FreeSpacePercent
- PACKDISK and FreeSpacePercent
- Freeing cylinders
- Creating more space on cylinders
- Managing spool space

Running Out of Disk Space

Introduction

The Teradata database does not run out of disk space until it allocates and fully utilizes all cylinders.

Low Cylinder Utilization

Performance degradations can occur, however, as soon as the system exhausts the free cylinder pool. This happens because the system performs mini-cylpacks on cylinders with low utilization in order to reclaim the unused disk space. Therefore, it is important to avoid creating cylinders with low utilization.

Low utilization of cylinders can occur when:

- You FastLoad a table using a small FreeSpacePercent (FSP) and then insert additional data to the table that is greater than the FSP.
- You delete a significant percent of a table but have not yet run PACKDISK to reclaim the space.

Frequently Updated Tables

With frequently updated tables, the free space on the cylinder can become so fragmented that it cannot be used.

When this occurs, the system could allocate additional cylinders to the table. To avoid this problem, the system sometimes performs a cylinder defragmentation to make the free space on the cylinder usable again.

Performance Degradation

While mini-cylpacks and defragmentation can help the system reclaim free disk space for further use, they both incur a performance degradation. Properly size and tune the system to avoid this overhead.

Running Out of Free Cylinders

Introduction

Teradata requires free cylinders to ensure that there is:

- Permanent space
- Spool space

Ensuring Optimal Performance

To ensure optimal performance, Teradata uses both:

Item	Description
Contiguous sectors on a cylinder	Data blocks are stored on adjacent sectors in a cylinder. If a cylinder has 20 available sectors, but only 10 are contiguous, a 15-sector block must be stored on another cylinder.
Free cylinders	Teradata performs better if permanent data is distributed across multiple cylinders. However, permanent data and spool data cannot share the same cylinder. Therefore, a system must always have empty cylinders that can be used for spool space.

Managing Space

Teradata has two automatic processes to deal with space issues:

Process	Description
Mini-cylinder packs	Frees cylinders: • Spontaneously, when the free cylinder threshold is met • Synchronously, when there are no empty cylinders and one is required See "What Mini-CylPacks Does" on page 9-10 for more information.
Defragmentation	Does not free cylinders. It creates more space on the currently used cylinders, diminishing the need for empty cylinders. See "Defragmentation" on page 9-14 for more information.

Freeing Space on Cylinders

To free space on cylinders, you can also use:

- FreeSpacePercent (FSP) (see page 9-8)
- PACKDISK (see *Teradata RDBMS Utilities*)
- Cylinders to Save for Perm (see "Cylinders Saved for PERM" on page 5-4)
- Spool limits on user definitions
  - Set spool space limits at a smart user level.
  - Minimize the number for individual users to limit runaway query spool space.

  Both of the above will not necessarily stop space management routines such as mini-cylpack from running, but it could help manage the area.
- Archival and deletion of aged data

  Planned deletion of obsolete rows facilitates space availability. Depending on the nature of your data, you may want to archive before deleting. See *Teradata Archive/Recovery Reference* for more information.
- Additional disk space (see page 9-13)
- Data compression (see page 9-15)
- Data block size:
  - Maximum block size allocation
  - Minimum block size allocation
  - Journal data block size allocation

FreeSpacePercent

Introduction

FreeSpacePercent (FSP) is a system-wide parameter. Use the SQL CREATE TABLE statement to define the free space percent for a specific table. FSP does not override a value you specify via a CREATE or ALTER TABLE request.

In some situations, Teradata runs out of free cylinders even though over 20% of the permanent disk storage space is available. This is due to:

- A higher FSP setting than necessary, which causes the system to allocate unrequired space
- A lower FSP setting than necessary, which causes the system to excessively allocate new cylinders
- Low storage density (utilization) on large tables due to cylinder splits

Determining a Value for FSP

Use the data in the following table to determine a value for FSP:

IF the majority of tables are...	THEN...
Read-only	set the default system-wide FSP value to 0. You can override the FSP for the remaining modifiable tables on an individual table basis with the ALTER TABLE statement.
NOT read-only	the FSP value depends on the percentage of increase in table size due to the added rows. Thus, the FreeSpacePercent parameter should be set to a value that reflects the net growth rate of the data tables (inserts minus deletes). It is commonly set to 5 to 15%. A value of 0% would be appropriate for tables that are not expected to grow after initially being loaded. For example, if the system keeps a history for 13 weeks, and adds data daily before purging a trailing 14th week, use an FSP of at least 1/13 (8%). To accommodate minor data skews and any increase in weekly volume, you can add an extra 5% (for a total of 13%) FSP.

Because the system dynamically allocates free cylinder space for storage of inserted or updated data rows, leaving space for this during the initial load allows a table to expand with less need for cylinder splits and migrates. The system uses free space for inserted or updated rows. However, if you do not expect table expansion, that is, the majority of tables are read-only, use the lowest value (0%) for FSP.

When the system default FSP is zero, use the information in the following table to minimize problems.

IF you...	THEN...
Use read-only tables	change nothing. At load time, or PACKDISK time, the system stores tables at maximum density.
Add data via BTEQ or a CLI program	set the FREESPACE value on the CREATE TABLE statement to an appropriate value before loading the table. If the table is loaded, use the ALTER TABLE statement to change FREESPACE to an appropriate value before running PACKDISK.

If you set FSP to a value other than 0, tables are forced to occupy more cylinders than necessary. The extra space is not reclaimed until either you insert rows into the table, use the Ferret utility to initiate PACKDISK on a table, or until mini-cylinder packs are performed due to a lack of free cylinders.

When the system default FSP is greater than 0, use the information in the following table to minimize problems:

IF You...	THEN...
Use read-only tables	set FREESPACE on the CREATE TABLE statement to 0 before loading the table. If the table is loaded, use the ALTER TABLE statement to change FREESPACE to 0 before running PACKDISK.
Add data via BTEQ or a CLI program	change nothing. The system adds rows at maximum density.

Operations Honoring the FSP

When adding rows to a table, the file system can choose either to use 100% of the storage cylinders available or to honor the FSP. The following operations honor FSP:

- FastLoad
- MultiLoad into empty tables
- Restore
- Table Rebuild
- Reconfig
- SQL to add fallback
- SQL to create a secondary index

Operations that Disregard FSP

The following operations disregard FSP:

- SQL inserts and updates
- Tpump
- MultiLoad inserts or updates to populated tables

If your system is tightly packed and you want to apply or reapply FSP, you can:

- Specify the IMMEDIATE clause with the ALTER TABLE statement on your largest tables.
- DROP your largest tables and FastLoad them
- DUMP your largest tables and RESTORE them
- in Ferret, set the SCOPE to TABLE and PACKDISK FSP = xxxx

In each case, table re-creation uses utilities that honor the FSP value, and fills cylinders to the FSP in effect. These options are only viable if you have the time window in which to accomplish the processing. Consider the following guidelines:

- If READ ONLY data, pack tightly (0%).
- For INSERTs:
  - Estimate growth percentage to get FSP. Add 5% for skewing.
  - After initial growth, FSP has no impact.
  - Reapply FSP with DROP/FASTLOAD, DUMP/RESTORE or PACKDISK operations.
  - Experiment with different FSP values before adding nodes or drives.

PACKDISK and FreeSpacePercent

Introduction

When Teradata runs out of free cylinders, you must run PACKDISK, an expensive overhead operation, to compact data to free up more cylinders. To reduce the frequency of PACKDISK operations:

- When FastLoading tables to which rows will be subsequently added, set FSP to 5-20% to provide enough free space to add rows.
- For historical data where you are adding and deleting data, provide enough free space to add rows. For example, you add up to 31 days before deleting on a table with 6 months history.

  Add 1 month to 6 months: 1/7 = 14.3%
  Safety - plan on 1.5 months, 1.5 / 7.5 = 20%
  Set Free Space Percent to 20 %
- For historical data and fragmented cylinders:
  - For large tables, either set FSP to 20 - 35% , or set MaxBlockSize to smaller size (for example, 16 KB).
  - Translate free space to the number of data blocks; plan on at least 6-12 blocks worth of free space.
- Specify the IMMEDIATE clause with the ALTER TABLE statement.

The table header contains the FSP for each table. If you change the default FSP, the system uses the new default the next time you modify the table. FSP has not effect on block size.

Running Other Utilities with PACKDISK

If you are run PACKDISK frequently, you should use the following "tools," two of which are utilities, to determine the amount of free space:

- DBC.DiskSpace
- SHOWSPACE, a ferret utility, will show you the percent of free space per cylinder. If this figure is low, it will impact performance by performing "on the fly" cylpacks when the system needs contiguous space.
- SHOWFSP, a ferret utility like SHOWSPACE, is useful in discovering specific tables that need packing.

  Showfsp runs very quickly, and could be scoped to a vproc. It will show number of cylinders that can be freed up for individual tables by specifying a desired free space percent. It can be useful in discovering which tables would free the most cylinders if packdisk were run on them. There may be certain tables that would free up a huge percentage of cylinders.

Cylinder Splits

A FreeSpacePercent value of 0% indicates that no empty space is reserved on Teradata disk cylinders for future growth when a new table is loaded. That is, the current setting causes each data cylinder to be packed 100% full when a new table is loaded.

Unless data is deleted from the table prior to subsequent row inserts, this situation will guarantee that a cylinder split will be necessary the first time an additional row is to be inserted into the table (following the initial load). Cylinder splits consume system I/O overhead and result in poor utilization of data cylinders in most circumstances.

PACKDISK and Cylinder Splits

Running PACKDISK after setting the FreeSpacePercent will pack data to the percent specified (that is, 100 minus FreeSpacePercent).

Prior to a cylinder split, data occupies 100% of space available on a cylinder, but after a cylinder split, half of the data is moved to a new cylinder, resulting in twice the number of cylinders required to contain the same amount of data. In addition, the number of empty cylinders (needed for spool space) is depleted.

Running the PACKDISK utility reverses the effect of cylinder splits, and packs the cylinders full of data, leaving empty only the percentage of space indicated by the FreeSpacePercent parameter unless a different free space percent is specified.

Freeing Cylinders

Introduction

You can free cylinders by:

- Through Mini-cylpacks
- By adding disk space (see page 9-13)

Mini-Cylpacks

Although cylinder packing itself has a small impact on performance, it often coincides with other performance impacting conditions or events. When the Teradata file system performs a mini-cylpack, the operation frees exactly one cylinder.

The cylpack operation itself runs at the priority of the user whose job needed the free cylinder. This is the last possible step that the system can take to recover space to perform a write operation, and is a signal that the system is out of space.

The Teradata file system will start to mini-cylpack when the number of free cylinders drops to the value set by DBScontrol parameter MiniCylPackLowCylProd. The default is 10.

Needing to cylpack may be a temporary condition in that a query or group of queries with very high spool usage are eating all available free space. In any event, this is not a desirable condition, and if space is a problem, it is a good practice to run the packdisk utility proactively.

What Mini-CylPacks Does

Mini-cylpacks move data blocks in logical sequence from cylinder to cylinder, stopping when the required number of free cylinders is available. A single mini-cylpack may effect 2 to 20 cylinders on an Access Module Process (AMP).

The decision of how to migrate the data blocks proceeds in the following order:

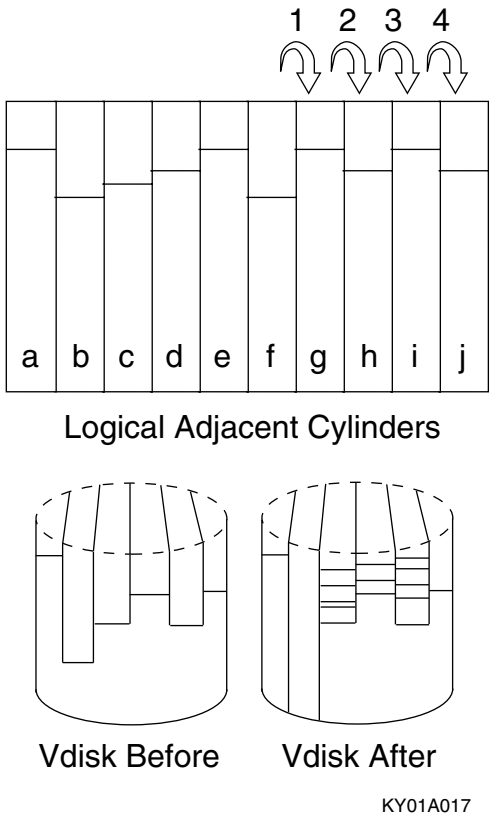
Stage	Process
1	If space can be made available either by migrating blocks forward to the next cylinder or backwards to the previous cylinder, choose the direction that would require moving the fewest blocks. If the number of blocks is the same, choose the direction of the cylinder with the most number of free sectors.
2	If Step 1 fails to free the desired sectors, try migrating blocks in the other direction.
3	If space can be made available only by allocating a new cylinder, allocate a new cylinder. The preference is to add a new cylinder: • Before the current cylinder for permanent tables • After the current cylinder for spool tables and while performing FastLoads.

When migrating either forward or backward, the number of blocks may vary because the system considers different blocks for migration.

Because of the restriction on key ranges within a cylinder, when migrating backwards, the system must move tables and rows with the lowest keys. When migrating forward, the system must move tables and rows with the largest keys.

The system follows special rules for migrating blocks between cylinders to cover special uses such as sort and restore. These are minor variations, such as migrating more data blocks than required in anticipation of addition needs, and looking for subtable breaks on a cylinder to decide how many data blocks to attempt to migrate.

The following figure illustrates a mini-cylpack.



In the above mini-cylpack:

Stage	Process
1	Data from cylinder 9 moves into cylinder 10 until cylinder 10 is full.
2	Data from cylinder 8 moves into cylinder 9 until cylinder 9 is full.
3	Data from cylinder 7 moves into cylinder 8 until cylinder 8 is full.
4	Data from cylinder 6 moves into cylinder 7 until cylinder 7 is full.

The process continues until one cylinder is completely emptied. The master index begins the next required mini-cylpack at the location that the last mini-cylpack completed.

The File Information Block (FIB) keeps a history of the last five cylinders allocated to avoid mini-cylpacks on them.

Note: Spool files are never cylpacked.

Use the dbscontrol utility (see "MiniCylPackLowCylProd" on page 5-20) to specify the free cylinder threshold that causes a mini-cylpack. If the system needs a free cylinder and none are available, a mini-cylpack occurs spontaneously.

Error Codes

Mini-cylpacks are a natural occurrence and serve as a warning that the system may be running short on space. Tightly packed data can encourage future cylinder allocation, which in turn trigger more mini-cylpacks.

The system logs mini-cylpacks in the Software_Event_Log with the following error codes:

Code	Description
340514100	Summary of mini-cylpacks done at threshold set in the DBS Control record.
340514200	A mini-cylpack occurred during processing and a task was waiting for it to complete.
340514300	The system could not free cylinders using mini-cylpack. The mini-cylpack failed. This means that the system is either getting too full or that the free cylinder threshold is set unreasonably high. Investigate this error code immediately.

Frequent 340514200 or 340514300 messages indicate that the configuration is under stress, often from large spool file requirements on all AMPs. Mini-cylpacks tend to occur across all AMPs until spool requirements subside. This impacts all running requests.

If table data is skewed, you might see mini-cylpacks even if Teradata has not used up most of the disk space.

Adding Disk Space

As your business grows, so does your database. Depending on the amount of historical data you wish to maintain online, your database may need to grow even if your business is not growing as quickly. With Teradata, you can add storage to existing nodes, or add storage as well as nodes.

Consider the following:

- Current performance of the existing nodes
- Existing bottlenecks
- Amount of space managed by an AMP
- Number of AMPs on the existing nodes

Creating More Space on Cylinders

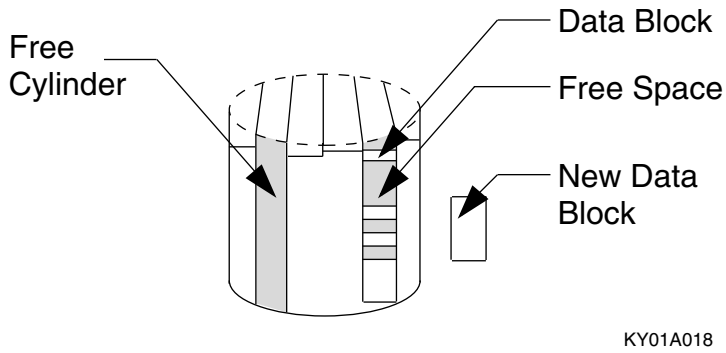
Introduction

This section discusses:

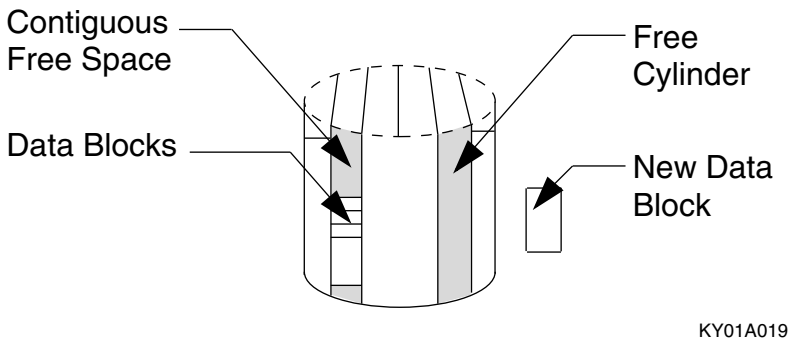
- Defragmentation
- Data compression
- Minimum and maximum data block size
- Journal block size.

Defragmentation

As random updates occur over time, empty gaps become scattered in data blocks on the cylinder. This is known as fragmentation. When a cylinder is fragmented, total free space may be sufficient for future updates, but the cylinder may not contain enough contiguous sectors to store a particular data block. This can cause cylinder migrates and even new cylinder allocations when new cylinders may be in short supply. To alleviate this problem, the file system defragments a fragmented cylinder, which collects all free space into contiguous sectors.



Use the dbscontrol utility (see "DefragLowCylProd" on page 5-9) to specify the free cylinder threshold that causes defragmentation. When the system reaches this free cylinder threshold, it defragments cylinders as a background task.



To defragment a cylinder, the file system allocates a new cylinder and copies data from the fragmented cylinder to the new one. The old cylinder eventually becomes free, resulting in a defragmented cylinder with no change in the number of available free cylinders.

since the copy is done in order, this results in the new cylinder having a single free sector entry that describes all the free sectors on the cylinder. New sector requests on this cylinder are completed successfully, whereas before they may have failed.

Data Compression

Implementing data compression on a grand scale will help most operations by making row sizes smaller, allowing more rows per block in a single Input/Output (I/O). This means less I/O and fewer blocks for the table.

Implement data compression through the CREATE TABLE statement. Compression may be a conflict in a truly Central Processing Unit (CPU)-intensive workload, but this is not normally a problem. Many tests have shown great improvements gained through the more-rows-per-block concept, thus reducing I/Os in the full table scan processes.

Maximum Data Block Size

You can set maximum block size two ways:

Operation	Comments
Set PermDBSize via the dbscontrol utility.	When you set maximum block size at the system level, a table utilizes a value only until a new value is set system-wide. This control helps organize the table for better performance for either Decision Support System (DSS) or Online Transaction Processing (OLTP) operations.
Use the CREATE or ALTER TABLE command.	When you set maximum block size at the table level, this value remains the same until you execute an ALTER TABLE command to change it.

Larger block sizes enhance full table scan operations by selecting more rows in a single I/O. The goal for DSS is to minimize the number of I/O operations, thus reducing the overall time spent on transactions.

Smaller block sizes are best used on transaction-oriented systems to minimize overhead by only retrieving what is needed.

In V2R2.x.x, the maximum data block size is 31.5 KB. In V2R3.0.x, the maximum data block size is 63.5 KB. For more information on data block size, see "PermDBSize" on page 5-24.

Rows cannot cross block boundaries. If an INSERT or UPDATE causes a block to expand beyond the defined maximum block size, the system splits the block into two or three blocks depending on the following:

IF...	AND...	THEN...
the new or changed row belongs in the beginning or end of a block	a block containing only that row is larger than the defined maximum size block for the table	the row is placed in a block by itself.
the new or changed row would fit in a block that is not larger than the defined maximum size block for the table		If possible, split the block into two parts such that each new block is not greater than the defined maximum.
the new row is greater than the maximum size block	the new or changed row belongs in the middle of the block, or an attempt to split the block in two failed	split the block into three parts with the new or changed row being in a block by itself.

Additional special rules exist that take precedence over the above. For example:

- Rows of different subtables never coexist in data blocks.
- Spool table blocks are usually maximum size, but sometimes can be smaller, because it is cheaper to write a new block than it is to update an existing block.

Minimum Data Block Allocation Unit

Set minimum block allocation via the PermDBAllocUnit field (see "PermDBAllocUnit" on page 5-22) in the dbscontrol utility.

Although this field may cause more unused space in blocks in the system, data can be maintained longer within a block without getting a larger block or doing a block split. This value also determines additional block growth (that is, a block must ultimately be a multiple of this value).

Journal Data Block Size

Journal data block sizes may also affect the I/O of a system. Set the journal data block size via the JournalDBSize parameter (see "JournalDBSize" on page 5-15) in the dbscontrol utility.

A larger journal size may result in less I/O or cause wasted block space on the database. Size your journal data blocks as accurately as possible.

Managing Spool Space

Introduction

Managing spool space allocation for users can be a method to control both space utilization and potentially bad (that is, unoptimized) queries.

Spool Space and Perm Space

Spool space is allocated to a user. If several users are active under the same logon and one query is executed that exhausts spool space, all active queries that require spool will likewise be denied additional spool and will be aborted.

If space is an issue, it is better to run out of spool space than to run out of perm space. A user requesting additional perm space will do so because he or she is executing queries that modify tables (inserts or updates, for example). Additional spool requests are almost always done to support a SELECT. Selects are not subject to rollback.

To configure this, see "Cylinders Saved for PERM" on page 5-4.

It is important to note that perm and spool allocations per user span the entire system. When the configuration is expanded, the allocation is spread across all AMPs. If the system size in AMPs has increased by 50%, it means that both perm and spool are now spread 50% thinner across all AMPs. This may require that some users' spool space, and possibly perm space, be raised if the data in their tables is badly skewed (that is, lumpy).

Increasing Spool Space

To increase spool space and increase performance:

- Create a spool reserve (see *Teradata RDBMS Administration*)
- Compress recurring values
- Eliminate unnecessary fallback
- Eliminate unnecessary indexes
- As a last resort, add hardware

Using Spool Space as a "Trip Wire"

Lowering spool space may be a way to catch "killer queries" that will in fact never finish or that will run the entire system out of free space if the user is allocated a very high spool space limit.

On handling "killer queries," see "Job Mix Tuning" on page 7-13.

In the interest of system performance, it is best not to allocate high spool space to all users and, in general, to be very conservative in the amount of space granted.

Chapter 10:

Using, Adjusting, and Monitoring Memory

This chapter discusses using, adjusting, and monitoring memory.

Topics include:

- Using memory effectively
- Shared memory
- Free memory
- FSG cache
- Managing I/O with Cylinder Read
- Monitoring memory

Using Memory Effectively

To determine if your system is using memory effectively:

Step	Action	See...
1	Start with a value for FSG Cache Percent.	"Reserving Free Memory" on page 10-4
2	Adjust value based on available free memory (40 MB required by UNIX as safety factor).	• "Reserving Free Memory" on page 10-4 • "Free Memory (UNIX)" on page 10-5

Also, you should consider adjusting the values in the following fields of the DBS Control Record:

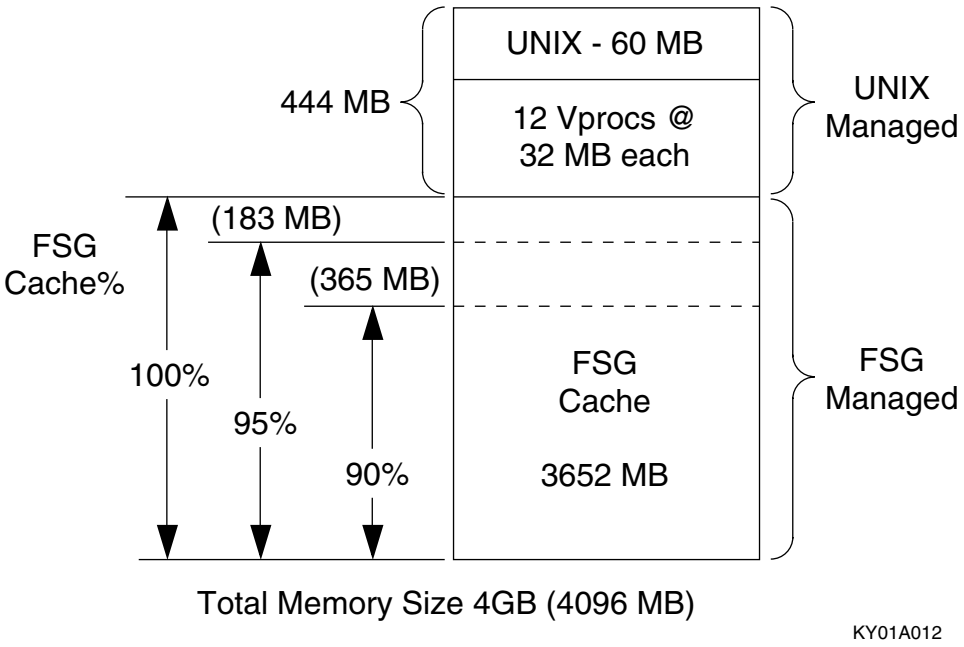
- "DBSCacheThr" on page 5-6
- "RedistBufSize" on page 5-31
- "SyncScanCacheThr" on page 5-42

The following sections discuss shared and free memory and explain how to reserve, increase, and monitor free memory.

Shared Memory (UNIX)

Diagram

The following diagram illustrates 4 GB shared memory on a Teradata RDBMS for UNIX system.



Shared memory on each node is divided into two main parts:

- Free memory, managed by UNIX (see page 10-5)
- FSG Cache, managed by Teradata File System (see page 10-9)

When UNIX boots, the system defines free memory. Parallel Database Extensions (PDE) reserves memory for UNIX overhead (a dynamic number usually about 20-60 MB) and 32 MB of memory for each Virtual Processor (vproc) to handle Teradata requirements.

The remaining memory is available for FSG cache. AMPs and PEs use this remaining memory.

For example, if you assume 2 GB (2048 MB) memory, memory for 8 AMPs, memory for 2 PEs, and memory for UNIX:

FSG Cache = 2 GB - (8 + 2) * 32 MB - 60 MB = 1668 MB

Early during UNIX startup, the system records the total amount of free memory as follows:

- When the database software starts up and the system knows the number of AMPs and PEs, the system makes an estimate of the memory needed to run all tasks with minimal paging. The system calculates the maximum number of pages available for caching Teradata file system blocks (FSG cache size) as the difference between initial free memory and this estimate.
- If you intend to run additional applications (with memory requirements unknown to the Teradata software) on the system, reduce the FSG cache percentage to leave memory available for these applications. The system applies this percentage to the above calculated maximum number of pages available, limiting the number of pages used by FSG cache.

Reserving Free Memory

You can use the xctl utility to reserve a percentage of shared memory for use by UNIX applications. For example, to reserve 20% of the FSG cache for UNIX applications over and above the 32 MB/vproc, go to the DBS screen and set FSG Cache Percent to 80. The system assigns 80% of the FSG cache to FSG and leaves the remaining 20% for other UNIX applications.

For more information on the xctl utility, see *Teradata RDBMS Utilities*. For Windows 2000, the equivalent of xctl is the ctl utility. For information, see *Teradata RDBMS Utilities*.

For more information on free memory, see page 10-5.

For more information on FSG cache, see page 10-9.

Memory Size

The appropriate amount of memory in each system running Teradata depends upon the applications running. You can use benchmarks to determine the appropriate memory size for a specific system.

Free Memory (UNIX)

Introduction

UNIX manages free memory. Free memory is used by:

- UNIX administrative programs, such as:
  - Program text and data
  - Message buffers
  - Kernel resources
  - Other applications such as FastLoad that require memory use
- Vprocs for non-file system activity, such as:

Activity	Description
AMP Worker Tasks (AWT)	Pieces of AMP logic used for specific AMP tasks
Parser tasks	Pieces of PE logic responsible for parsing Structured Query Language (SQL)
Dispatcher tasks	Pieces of PE logic responsible for dispatching work
Scratch segments	Temporary work space
Messages	Communication between vprocs
Dictionary cache	Dictionary steps for parsing
Steps cache	Temporary space used when executing steps

ResUsage and Available Free Memory

The ResNode macro displays limited information on free memory. The ResNode report includes the Free Mem % column, which is the percentage of unused memory.

Adjusting for Low Available Free Memory

When the amount of available free memory dips too far below 40 MB (10,000 pages), some sites have experienced UNIX panics. You can adjust the amount of available free memory by performing one or more of the following:

- Use the xctl utility to adjust the FSG Cache Percent to make more memory available to free memory. If the system takes too much memory from FSG cache, and UNIX does not use that memory, the free memory is wasted.

- If available free memory goes below 40MB during heavy periods of redistribution (as explained later in this section), you can lower the value of the RedistBufSize field in the DBS Control Record (see "RedistBufSize" on page 5-31).
- To protect against UNIX panics and prevent wasting free memory, adjust the UNIX parameters in the /etc/conf/cf.d/stune file as follows:

Change this parameter...	From default (pages) of...	To (pages)...
LOTSFREE	512	8192
DESFREE	256	4096
MINFREE	128	2048

This enables UNIX to start paging and free up memory sooner.

Potential Problems

A possible problem is when an application on a large configuration generates many messages over the BYNET with concurrent row redistributions involving all nodes (see the subsections below).

The following are NOT a problem:

- Row duplications
- Merging of answer sets

Row Redistribution Memory Requirement

Row redistribution uses separate single buffers per AMP for each node in the system. This means the amount of memory required for redistribution in a node grows as the system grows (see the table illustrating the growth of redistribution buffer size on page 5-31).

- Default redistribution buffer size = 32 KB per target node
- Total memory for 1 sending AMP = 32 KB * number of nodes in system
- For 8 AMPs per node, total memory required per node = 8 * 32KB * number of nodes in system

Redistribution Processing

The following example provides the calculations for a configuration of 8 nodes at 8 AMPs per node. (The system only reserves 32MB per AMP).

- Single node requirement (single user) = 32 KB * 8 = 256KB
- Multi-user (for example, 20 concurrent users) = 20 * 256KB = 5MB (not a special problem)

The following example provides the calculations for a configuration of 96 nodes at 8 AMPs per node:

- Single node requirement (single user) = 32KB * 96 = 3072KB (3MB)
- Multi-user (20 concurrent users) = 20 * 3072KB = 64MB (far exceeding 32MB perAMP)

Symptoms of high-volume redistribution processing include:

- Excessive memory paging/swapping
- Possible I/O bottleneck on BYNET I/O

Aggregate Processing Memory Requirement

1 MB virtual memory is available on each AMP for local aggregate processing.

When a high volume of row redistribution is combined with a high volume of concurrent sessions employing aggregate processing, memory could become a problem on large configurations.

Recommendations

For large configurations, consider one or more of the following options to resolve I/O bottlenecks or excessive memory swapping:

- Consider using aggregate join indexes to reduce aggregate calculations during query processing.
- SetR RedistBufSize an incremental value lower; for example, from 4 KP to 3 KB
- Set FSG Cache Percent to less than 100% (the percent specified depends on total memory size. (see Recommendation under "RedistBufSize" on page 5-31).
- Consider modifying the application to reduce row redistribution.
- Ask your NCR support representative to reduce the internal redistribution buffer size (for example, to 16KB).

  Note: This is an internal tuning parameter, *not* the user-tunable RedistBufSize.

Practical experience in the field indicates that most sites require more free memory for UNIX than the default calculated on page 10-3. On Teradata RDBMS for UNIX, it is recommended that you provide an additional 100 MB to free memory by setting the FSG Cache Percent to a value less than 100%. Use the following calculation:

FSG Cache Percent = (FSG Cache - 100 MB) / FSG Cache

This additional 100 MB for UNIX reduces the FSG Cache Percent as a function of total memory size as shown in the table below. You may need to perform further tuning depending on the load from UNIX applications and Teradata utility programs.

Memory Size (GB)	Memory for UNIX & 10 Vprocs	FSG Cache (MB)	Less 100 MB	FSG Cache Percent
1.0	380	644	544	84%
1.5	380	1,156	1,056	91%
2.0	380	1,668	1,568	94%
3.0	380	2,692	2,592	96%
4.0	380	3,716	3,616	97%

FSG Cache

Introduction

Teradata File System manages FSG cache, which is used by:

- AMPs on the node
- Backup activity for AMPs on other nodes

The Teradata File System uses FSG cache for file system segments such as:

- Permanent data blocks (includes fallback data and secondary indexes)
- Cylinder Indexes (CIs) for permanent data blocks
- Cylinder stats (for Cylinder Read)
- Spool data blocks and CIs for spool
- Transient Journals (TJs)
- Permanent Journals (PJs)
- Synchronized scan (sync scan) data blocks
- Buddy Backup data blocks

Space in FSG Cache

Space in the FSG cache is not necessarily evenly distributed among AMPs. It is more like a pool of memory; each AMP uses what it needs.

This cache contains as many of the most recently used database segments as will fit in it. When Teradata tries to read a database block, it checks the cache first. If the block is cached, Teradata avoids the overhead of rereading the block from disk.

The system performs optimally when FSG cache is as large as possible, but not so large that there is not enough memory for the database programs, scratch segments, and other UNIX programs that run on the node.

Managing I/O with Cylinder Read

Memory Slots in FSG Cache

An FSG segment is the basic unit of memory buffer provided by the PDE for the Teradata File System to manage and access data. When a task requires an FSG segment, the corresponding data is mapped into the FSG virtual address space.

With Cylinder Read, the FSG cache can be viewed as consisting of two regions, the Cylinder Pool and the Individual Segment. The Cylinder Pool occupies the high region and is cut into cylinder-sized memory slots. The size of each slot is 1936KB (equal to 484 pages of memory).

The Cylinder Read Process

When Cylinder Read is enabled (the initial default), it is invoked implicitly, based on memory conditions as well as the nature of the current statement. The processing sequence is as follows:

<table>
<tr><th>At this time ...</th><th>This entity ...</th><th>Performs the following ...</th></tr>
<tr><td>startup</td><td>each AMP</td><td>maps a view of its FSG cache into its virtual address space. (The percentage of available memory to be used for the cache is defined by the FSGCachePercent setting in DBSCONTROLGDO.)</td></tr>
<tr><td>startup</td><td>FSG</td><td>determines whether the amount of cache memory per AMP is sufficient to support Cylinder Read operation.
<table>
<tr><th>IF there is ...</th><th>THEN FSG ...</th></tr>
<tr><td>enough memory to support Cylinder Read</td><td>Allocates a number of cylinder memory slots per AMP. Depending on the settings of the DBSCONTROLGDO, this number is:
<table>
<tr><th>IF the Cylinder Read field is set to ...</th><th>THEN the number of slots FSG allocates per AMP is ...</th></tr>
<tr><td>DEFAULT</td><td>four</td></tr>
<tr><td>USER, and available memory is adequate</td><td>the number you selected in Number of Slots/AMP</td></tr>
<tr><td>USER, but available memory is not adequate for your setting</td><td>the number FSG calculates as being optimum.</td></tr>
</table></td></tr>
</table></td></tr>
</table>

<table>
<tr><th>At this time …</th><th>This entity …</th><th>Performs the following …</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>not enough memory to support Cylinder Read</td><td>turns Cylinder Read OFF. It is not enabled again until more memory is freed.</td></tr>
</table>
</td></tr>
<tr><td>receipt of a statement</td><td>DBS</td><td>determines if the statement is a candidate for Cylinder Read operation, such as a full-table scan, an update of a large table, or a join involving many data blocks.
<table>
<tr><th>IF the statement is …</th><th>THEN DBS …</th></tr>
<tr><td>not suitable for Cylinder Read</td><td>builds a subtable of data blocks from the target table and invokes a File System read function to read each data block.</td></tr>
<tr><td>suitable for Cylinder Read</td><td>prepares for processing as follows:
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Builds a subtable of data blocks from the table.</td></tr>
<tr><td>2</td><td>Sets the internal cylinder read (CR) flag.</td></tr>
<tr><td>3</td><td>Invokes a File System read function.</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>detection of the CR flag</td><td>the File System</td><td>loops through each cylinder that contains data blocks for the target subtable and checks the number of data blocks.
<table>
<tr><th>IF the number of data blocks on the current cylinder is …</th><th>THEN the File System …</th></tr>
<tr><td>less than six</td><td>reads the data blocks on the current cylinder one at a time.</td></tr>
<tr><td>six or more</td><td>
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Constructs a list of the data blocks on the current cylinder.</td></tr>
<tr><td>2</td><td>Sends a cylinder read request to the FSG.</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>At this time …</th><th>This entity …</th><th>Performs the following …</th></tr>
<tr><td>receipt of a statement prepared for Cylinder Read</td><td>FSG</td><td>uses Cylinder Read to scan the data when all of the following conditions are met:
<table>
<tr><th>IF all of the following are true …</th><th>THEN FSG …</th></tr>
<tr><td>a free cylinder slot exists within FSG cache</td><td rowspan="3">Loads into a cylinder slot the smallest chunk containing data blocks on the list.</td></tr>
<tr><td>data blocks already in cache from a previous statement do not reduce the number of data blocks in the current list to less than 6</td></tr>
<tr><td>the I/O time needed to read the blocks on the cylinder is less than the I/O time needed to load the blocks individually, based on:
• Chunk size
• Spacing between the data blocks in the chunk
• Drive seek-time
• Drive data-transfer rate</td></tr>
</table></td></tr>
<tr><td>cylinder read operation</td><td>scanning task</td><td>reads cylinders as follows:
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>As the File System prepares new subtable lists and FSG loads new cylinders, the scanning task continues to read until the statement is satisfied or terminated.</td></tr>
<tr><td>2</td><td>Each time the scanning task moves to the next cylinder, the previous cylinder is immediately freed and returned to the list of free slots.</td></tr>
<tr><td>3</td><td>If the scanning task encounters a disk read error, the statement is aborted and all data processed so far is rolled back.</td></tr>
</table></td></tr>
</table>

Changing the Cylinder Read Defaults

When Teradata RDBMS is installed, Cylinder Read is enabled by default and Cylinder Slots/AMP is set to 10. The feature is disabled automatically if FSG memory is calculated to be below 36 MB per AMP.

You can manually disable or re-enable Cylinder Read and/or change the number of slots per AMP using:

- Teradata MultiTool
- xctl utility (UNIX)
- ctl utility (Windows 2000)

The Cylinder Read setting and the Number of Slots/AMP value are interdependent, as follows:

IF the Cylinder Read field is set to …	THEN …
DEFAULT	the value for Cylinder Slots/AMP is calculated automatically. If you set the slider to a value, the setting is ignored.
USER	you can set the Cylinder Slots/AMP value yourself. However, based on FSG cache size, in rare cases FSG may have to change the number of slots per AMP. (NCR recommends that as a general rule, the default setting should provide the best performance.) For an explanation and instructions on how to check the current allocation after a reset, see "Viewing the Cylinder Slot Configuration" on page 10-13.

For detailed instructions on setting Cylinder Read parameters, see the chapters titled "ctl Utility" and "xctl Utility" in *Teradata RDBMS Utilities*.

Viewing the Cylinder Slot Configuration

During reset, FSG recalculates the size of FSG cache and determines whether there is enough memory to allocate the number of slots per AMP that you selected. If not, or if you did not select a number, it attempts to allocate the default. Otherwise, it allocates as many slots as it can. For example, only 2 slots can be configured when FSG cache is down to 36 MB per AMP.

Therefore, although it should not happen often, it is possible that after a reset the number of slots configured by FSG may be different from your selection. When you need to know, you can find the actual slot configuration using the Database Window and the procedure outlined below. (For complete details on all the operations you can run in the Database Window, see *Teradata RDBMS Database Window*.)

Step	Action
1	Access the Database Window.
2	Click on the Supvr icon to open the Supervisor window.
3	To bring up the File System Utility Routines (FSU) display, in the Enter a command: window, type: `start fsudriver`
4	When the FSU display appears, in the Enter a command: window, type the field number of the CylSlotInfo field: `93`

Step	Action

5	The field displays the current state of Cylinder Read and the number of slots configured per AMP. The display should be similar to the following:
	
	`******** Cylinder Read Status ********`
	
	`Cylinder Read is on.`
	`9 slots allocated.`
	
	`Hit return to continue`

Calculating FSG Cache Size Requirements

The FSG Cache Percent field controls the percentage of memory to be allocated to FSG cache. You can change the value in FSG Cache Percent using the xctl utility (on UNIX) or ctl utility (on Windows 2000). To determine size, see "Calculating the FSG Cache Size" in *Teradata RDBMS Utilities*

Tracking Cylinder Read ResUsage

The following fields have been added to the RSS_Svpr table. You can use these fields to track Cylinder Read behavior if you enable ResUsage logging.

For details on Resource Usage, see *Teradata RDBMS Resource Usage Macros and Tables*.

This Cylinder Read field ...	Reports the ...
Svpr_FileFcrRequests	total number of times a cylinder read was requested.
Svpr_FileFcrDeniedThresh	number of times a cylinder read request was rejected because FSG determined that either: • The number of data blocks to be loaded was below the threshold • It was more efficient to read the data blocks individually
Svpr_FileFcrDeniedCache	number of times that a cylinder read request was denied because a cylinder slot was not available at the time of the cylinder read request. (The sum of Svpr_FileFcrDeniedThresh and Svpr_FileFcrDeniedCache yields the total number of rejected cylinder read requests.)
Svpr_FileFcrBlocksRead	total number of data blocks that were loaded with cylinder reads.

This Cylinder Read field …	Reports the …
Svpr_FileFcrBlocksDeniedThresh	total number of data blocks that were not loaded with cylinder reads because the cylinder read requests did not meet the threshold criteria (linked to FileFcrDeniedThresh).
Svpr_FileFcrBlocksDeniedCache	total number of data blocks that were not loaded with cylinder reads because the cylinder read requests were submitted at times when cylinder slots were not available (linked to FileFcrDeniedCache).

Monitoring Memory

Use the ResUsage tables to obtain records of free memory usage and FSG cache. (UNIX knows nothing about FSG cache.)

Memory Type	Managed by	Monitor With	Comments
Free Memory	UNIX tools	sar or xperfstate or ResUsage	See Chapter 2 for more information on using ResUsage to monitor free memory.
	Windows 2000 tools	sar, ResUsage, Task Manager	
FSG Cache	Teradata File System	ResUsage (SPMA) ResUsage (SVPR)	See Chapter 2 for more information on ResUsage macros.

Chapter 11:

Teradata Manager and Performance

Topics include:

- What is Teradata Manager?
- Performance analysis and problem resolution
- Database management
- Operational control
- Using Teradata Manager applications
- Performance impact of Teradata Manager
- Monitoring real-time activity
- Investigating system behavior
- Investigating the audit log
- Analyzing workload trends
- Analyzing historical resource utilization
- Using Priority Scheduler to balance the workload
- Using alerts to monitor the system
- Using Performance Monitor Object
- Teradata Manager and ResUsage Compared
- Teradata Manager and DBC.AMPUsage view compared
- Teradata Manager compared with HUTCNS and DBW utilities
- Teradata Manager and SHOWSPACE compared
- Teradata Manager and TDP monitoring commands
- Teradata Manager and HSI, TDPUTCE, and SMF
- Teradata Manager and job control support tools

For complete information on Teradata Manager, see *Teradata Manager User Guide*.

What is Teradata Manager?

Command Central

Teradata Manager is the "command center" for the Teradata Database. Teradata Manager, a single Windows-based graphical systems management platform, contains a suite of specialized tools and applications to monitor and control the Teradata database.

Teradata Manager's client/server feature replicates performance data in the server for access by any number of clients. Because data is collected once, workload on the database remains constant while the number of client applications varies.

Teradata Manager's applications collect, analyze, and display performance and database utilization information in either report or graphic format from a desktop, laptop, or even from the palm of your hand using the Wireless Palm VII.

Performance Analysis and Problem Resolution

Introduction

Teradata Manager's suite of performance monitoring applications collects, queries, manipulates, and displays performance and usage data to allow you identify and resolve resource usage abnormalities detected in the Teradata Database. Dynamic and historical data are displayed in graphical and tabular formats.

Teradata Manager's applications and features include:

- Teradata Performance Monitor
- Teradata Priority Scheduler Administrator
- Centralized alerts/events management
- Trend analysis

Teradata Performance Monitor

Performance Monitor (PMON) provides the system status with functional areas for monitoring system activity. These include:

- Configuration Summary
- Performance Summary and Resource Usage
- Session and Lock information
- Session History
- Control functions
- Graphic displays of Resource data
- Graphic displays of Session data

PMON uses charting facilities to present the data to identify abnormalities. Color is used to indicate warning conditions. You can configure the Alert thresholds, color settings, and automatic data refresh rate values using the PMON Alert tab.

Detailed session and user information is useful for analyzing system activity and blocked sessions. Lock information helps you determine which sessions are blocking other sessions and why. Analysis of running queries lets you drill down from a blocked session to the query and the step level of the EXPLAIN for the query.

Teradata Priority Scheduler Administrator

Teradata Priority Scheduler Administrator (PSA) provides a graphical interface to Priority Scheduler.

See "Priority Scheduler" on page 7-2. See also "Using Priority Scheduler Administrator to Balance the Workload" on page 11-17. For complete information on Priority Scheduler, see "Priority Scheduler" in *Teradata RDBMS Utilities*.

Centralized Alerts/Event Management

This feature facilitates the monitoring of performance characteristics and faults on the Teradata database. The alerts/events management feature can automatically activate actions, such as sending a page, sending e-mail, or sending a message to a Simple Network Management Protocol (SNMP) system.

Other applications and utility programs can also make use of the Alert function by using a built-in request interface.

- The Alert Policy Editor is the interface that defines actions and specifies when they should be taken based on thresholds that can be set for Teradata performance parameters, database space utilization, and messages in the database Event Log.
- The Alert Viewer allows you to see system status for multiple systems.

Trend Analysis

Teradata Manager's trend analysis features allow the study of Teradata resource utilization trends from summarized reports displayed as charts in Teradata Manager.

You can detect resource usage abnormalities, determine the onset of a problem, and analyze the impact of the problem on the system. Teradata Manager charts can also be used to help schedule production jobs and balance Teradata load levels.

Database Management

Teradata Manager's database management applications permit you to monitor system tables and space usage and perform database administration via a graphical interface.

Teradata Manager's management applications and features include:

- Teradata Administrator
- Space Usage
- System Maintenance

Teradata Administrator

Teradata Administrator is a Windows-base tool that interfaces with the Teradata Data Dictionary in order to perform database administration tasks.

All Teradata Administrator functions are saved in SQL History files so you can use them later.

Space Usage

Space Usage monitors disk space utilization and re-allocates permanent space from one database to another.

System Maintenance

System Maintenance provides various macros for performing clean-up of system tables on Teradata.

Operational Control

Teradata Manager's applications monitor and maintain your critical system operations.

Operational control "tools" include:

- Session Information
- Remote Console
- Error Log Analyzer
- LogOnOff Usage
- BTEQ Window
- Access Management
- Customizable installation

Session Information

Session Information dynamically monitors the status of sessions on logical host processors attached to the Teradata database.

Status can be updated as frequently as every six seconds depending on the size of the database and the workload. Information is reported on the current status of a session including Idle, Active, Blocked, Responding, Parsing, Aborting, Details, and Prolonged Idles.

Using Session Information, abnormal conditions in the database can be identified and any unnecessary jobs aborted.

Remote Console

Remote Console allows you to run many of the Teradata console utilities from the Teradata Manager PC.

Error Log Analyzer

Error Log Analyzer provides an interface through which to view the error logs for an associated Teradata database.

LogOnOff Usage

LogOnOff Usage presents daily, weekly and monthly logon statistics.

BTEQ Window

BTEQWIN, the graphical user interface for BTEQ, incorporates Windows' features on the Teradata Manager PC.

Access Management

Security access to the Teradata database can be managed using Teradata Administrator and Profile.

Teradata Administrator establishes account and privilege assignments that control access to the Teradata database. The Profile feature creates user profiles that define who can access certain Teradata and Teradata Manager applications.

Customizable Installation

With Teradata Manager, one PC can manage multiple Teradata platforms, or conversely multiple PCs can manage a single Teradata platform from distributed locations. You can decide how each of your Teradata Manager PCs is configured and assigned.

Each PC can be configured using industry-standard wizards with the appropriate Teradata Manager applications for the job.

Using Teradata Manager Applications

This following table suggests ways in which Teradata Manage application can be used:

If you want to...	You Can Use...
Define the actions that should take place when specific events occur on your Teradata machines	Alert Policy Editor It allows the DBS to define the actions that should take place when performance of database space events occur on one or more Teradata machine.
View the overall system performance of your Teradata RDBMS systems from a single view point	Alert Viewer It provides the user, through a graphical user interface and colored icons, with a single "point of view" with respect to overall system performance.
Load and extract data from the Teradata RDBMS using a version of BTEQ with a Windows interface	BTEQWIN It allows the user to run multiple BTEQ sessions and to access different database simultaneously under Teradata Manager
Make sure your system is properly configured to run Teradata Manager	Configuration Check A Teradata Manager utility that collects configuration information related to an associated Teradata Database and stores it on the client PC.
Investigate the Teradata Database error log	Error Log Analyzer (ELA) It allows you to view the error log system tables on the Teradata Database
Determine whether system performance has been degraded by an inappropriate mix of SQL statements using a table of information extracted from the transaction logs	Locking Logger A menu-driven interface to the Locking Logger console utility.

If you want to...	You Can Use...
View daily, weekly, and monthly logon statistics based on information in the DBC LOGONOFF view on the Teradata RDBMS	LogOnOff Usage Presents daily, weekly, and monthly logon statistics based on information in the DBC.LOGONOFF view on the associated Teradata Database.
Control the allocation and consumption of computer resources available to the Teradata RDBMS	The Dashboard Resource-management tool that provides a graphical interface that lets you define Priority Definition sets and generate schmon scripts to implements these sets.
Set up the RDBMS to run Teradata Manager	RDBMS Setup Tool for setting up the Teradata Database with the databases, tables macros, and users necessary to run Teradata Manager.
Run many of the Teradata RDBMS console utilities from your Teradata Manager PC	Remote Console Allows you to run many of the Teradata Database console utilities from Teradata Manager.
Get information on session status, modify session priority, view blocking/blocked session, change session priority	Session Information Uses PMON to collect the session performance monitor data from the Teradata Database.
Monitor disk space utilization and move permanent space from one database to another	Space Usage Is used to monitor the use of disk space on the associated Teradata Database and to reallocate permanent disk space from one database to another.

If you want to...	You Can Use...
Create, drop and update statistics for the Teradata RDBMS	Statistics Collection Collects statistics on a column or key field to assist the Optimizer in choosing an execution plan that will minimize query time.
Analyze the performance of a Teradata machine and perform related control functions on that machine	PMON Is used to monitor the general health of a Teradata database, enabling you to analyze current performance and both current and historical session information and to abort sessions that are causing system problems.

Performance Impact of Teradata Manager

The following types of overhead are associated with Teradata Manager:

Monitoring Resources or Sessions

Monitoring resources or sessions includes activity the system performs when you set monitoring to a specific rate.

- Resource monitoring has little or no effect on performance, even with frequent collections.
- Session monitoring causes a slight throughput slowdown, depending on the workload size.

Querying Resources or Sessions

Querying resources or sessions includes the extra activity required to process and return an answer set when user makes a request to see the results.

- Resource querying causes minimal overhead.
- Session querying can cause very high overhead, depending on the workload size and querying frequency.

  Session querying overhead is minimal for AMP vprocs, but quite costly for PE vprocs.

  If your system is CPU-bound, seriously consider session querying rate and how it could affect performance.

Monitoring Real-Time Activity

Introduction

Teradata Manager provides several options for viewing real-time activity on your system.

Monitoring Overall Activity Using the Dashboard

The Dashboard provides you, on a single page, with a summary of the current and recent history state of the system. The page contains real-time data and trend charts showing resource consumption over the last 60 minutes.

Lights on the status bar at the top of the Dashboard indicate AMP, PE, Node and BYNET status: a green light indicates a status of “ok”. A red light indicates a status of “down”.

Viewing the Dashboard

While the top-level Dashboard shows current and recent historical information, you can drill down from the Dashboard, or specifically select the reports from the menu bar, to view details of system utilization and session information.

From the Dashboard, you can access displays that let you view:

- Current data
- Trend data over time
- Historical data
- vproc utilization
- Node utilization
- Session status

  You can also modify the priority of the session, abort the session, view what a selected session is blocking and/or is blocked by

  In addition, you can view key information about each session using PMON and view key information about each session using Session Information.

Investigating System Behavior

Teradata Manager provides several ways to investigate system behavior using various modules.

You can investigate:

- Error Log
- Disk space utilization
- Logon activity
- Lock contentions
- System performance parameters

Investigating the Audit Log

Each row in the Audit Log report indicates the results of a privilege check. Whether a privilege check is logged depends on the presence and the criteria of an access logging rule.

You can define your report criteria by setting Audit Log Filter parameters before running the Audit report.

For example, you can:

- Set a filter so you can narrow the results of your Audit reports
- Audit database and user privilege check results
- Audit table, view, and macro privilege check results
- Audit Grant and Revoke privileges check results
- Audit Index privilege check results
- Audit Checkpoint, Dump and Restore privilege check results
- Audit privilege check denials only
- Create a summary report of privilege check results
- Create a customized privilege check report

Analyzing Workload Trends

Workload Analysis provides a historical view of how the system is being utilized, based on data that has been collected by the Teradata Manager data collection service.

The data can be grouped in various ways, and trend tables and graphs can be filtered according to various criteria, depending on the type of report.

You can analyze:

- CPU utilization
- Disk I/O utilization
- Table growth
- Spool space usage
- Temporary space usage
- Heartbeat query response and retrieve time
- Number of concurrent/distinct users

Analyzing Historical Resource Utilization

Using Historical Resource Utilization reports, you can analyze the maximum and average usage for Logical Devices (LDVs), AMP vprocs, nodes, and PE vprocs on your system.

ByGroup reports and graphs differentiate the node processor generations (for example, 5100 vs. 5200) in a coexistence system, allowing for more meaningful data analysis for Teradata coexistence (mixed platform) systems.

You can generate a report that provides information on:

- Node CPU utilization
- AMP CPU utilization
- PE CPU utilization
- Disk utilization
- General system information averaged across nodes by node group
- Network (BYNET) utilization
- Host utilization
- Memory allocation, aging, paging, and swapping activity on the nodes

Using Priority Scheduler Administrator to Balance the Workload

Priority Scheduler Administrator (PSA) is a resource-management tool that provides a graphical interface that allows you to define Priority Definition Sets and generate schmon scripts to implement these sets.

For information on using Priority Scheduler Administrator, see *Teradata Manager User Guide*.

For information on Priority Scheduler, see "Priority Scheduler" on page 7-2. For a complete information on Priority Scheduler, see "Priority Scheduler," see *Teradata RDBMS Utilities*.

Using Alerts to Monitor the System

Teradata Manager includes an alerting facility that monitors the Teradata database and automatically invokes actions when critical (or otherwise interesting) events occur.

The alerting facility allows you to define simple actions such as page the Database Administrator (DBA), send e-mail to the Chief Information Officer (CIO), escalate incidents to the help desk. It also allows you to define more sophisticated actions that perform corrective measures, such as lowering a session's priority or running a user defined SQL script.

For information on the alerting facility, see *Teradata Manager User Guide*.

Alert Capabilities of Teradata Manager

The alert capabilities of Teradata Manager can be summarized as follows:

- The alert feature can generate alerts based on the following events:
  - System events
  - Node events
  - vproc events
  - Session events
  - SQL events
  - Manual events
- When an alert is triggered, one or more of the following actions can be performed:
  - Page an administrator
  - Send e-mail to an administrator
  - Display a banner message on the PC running ACM
  - Send an SNMP trap
  - Run a program
  - Run a BTEQ script
  - Write to the alert log

Suggested Alerts and Thresholds

The following table lists key events and the values that constitute either warning or alerts.

Type	Event	Warning	Critical
CPU saturation	Average system CPU > x%	(x = 95)	
I/O saturation	CPU+WIO > x% and WIO > y%	(x = 90, y = 20)	
Query blocked	Query or Session blocked on a resource for longer than x minutes, and by whom	(x = 60)	
Entire system blocked	Total number of blocked processes > x		(x = 10)
User exceeding normal usage	Number of sessions per user >x (with an exclusion list, and custom code to roll up sessions by user)	(x = 4)	
“Hot Node” or “Hot AMP” problem	Inter-node or inter-AMP parallelism is less than x% for more than 10 minutes	(x=10)	
Disk space	Disk Use% > x (vproc)	(x=90)	(x=95)
Product Join	Average BYNET > x% (system)	(x=50)	
System restart	Restart		SNMP
Node down	Node is down		SNMP
Canary query	Timeout		SNMP

Using Performance Monitor Object

The Performance Monitor Object is an ActiveX (COM) object that provides methods to allow retrieval of Performance Monitor & Production Control (PM & PC) data.

Each method returns a collection of objects, each of which encapsulates the performance data for that object.

It also provides for execution of the production control (action) functions of PM & PC.

For information on Performance Monitor Object, see *Teradata Manager User Guide*.

Teradata Manager and ResUsage Compared

Teradata Manager collects resource usage data in a manner similar to ResUsage data. And as such, it has advantages and disadvantages with respect to ResUsage.

For more information on ResUsage reports, refer to *Teradata RDBMS Resource Usage Macros and Tables*.

Benefits of Teradata Manager

Teradata Manager data has the following benefits:

- Teradata Manager saves the raw data in an in-memory buffer where a client application program can easily retrieve the data for real-time analysis.
- Teradata Manager acquires data in near real time, with less overhead and with minimal possibility of blocking on data. These capabilities allow you to perform frequent on-the-fly performance analysis.
- Teradata Manager provides data that ResUsage reports do not, such as session-level resource usage data, and data on application locks and which application is being blocked.

Limitations of Teradata Manager

Teradata Manager has the following limitations:

- Teradata Manager data is not as detailed as ResUsage data.
- Teradata Manager does not write data to tables. Therefore, you lose current resource usage data at the end of the sample interval.
- Teradata Manager cannot accumulate large amounts of data over a long period of time. Therefore, you cannot use this data for:
  - Examining trends and patterns
  - Planning system upgrades
  - Deciding when to add new applications to systems already heavily utilized
  - Building baseline resource usage profiles for operations

Teradata Manager and DBC.AMPUsage View Compared

Teradata Manager displays more centralized performance data than DBC.AMPUsage view, including:

- AMP CPU usage in seconds
- Disk Inputs/Outputs (I/Os) and other session usage information by user name and account name for the request active in the session.

See "ResUsage and DBC.AMPUsage View Compared" on page 2-17. For information on DBC.AMPUsage view, refer to *Teradata RDBMS Data Dictionary*.

Teradata Manager Compared with HUTCNS and DBW Utilities

The following three utilities have functions in common with Teradata Manager:

Utility	Description
Query Session	Provides information about active Teradata sessions.
ShowLocks	Provides information about host utility locks placed on databases and tables by the Archive and Recovery utility during database operations.
Query Configuration	Reports the current Teradata configuration from a system console running the Database Window or from a host terminal.

For more information on Query Session, Showlocks, and Query Configuration, see *Teradata RDBMS Utilities* and *Teradata RDBMS Database Window*.

Teradata Manager and Query Session

Following are the differences between Teradata Manager and the Query Session utility:

Teradata Manager Feature	Query Session Feature
Provides better information on Teradata SQL sessions.	Provides better information on utility sessions.
Displays the name of the user ID or session number causing the block, as well as the name of the database or table involved in the lock.	Does not identify the user ID or session number causing the block. It identifies only the table or database that is blocked. Without the user ID or session number, you cannot abort or log off the blocking session or user.
Allows you to select the information you want to display, as well as the display format. You can display information on: • All clients for all sessions • All clients for a single username • Single clients for all sessions • Single clients for all sessions for a given username • A single client for a single session and single username.	Does not allow you to select the information you want to display, nor the display format. It supports some wild card requests (requests for statistics on all logical host IDs and/or on all sessions).

Teradata Manager Feature	Query Session Feature
Is less likely to be blocked.	Is more likely to be blocked.
Only provides information on how long a session has been running.	Provides the total CPU usage of the current request.

Note: Because Query Session tells you how long the current transaction has been running, use it to identify problem sessions before you run ABORT SESSION from the DBW.

Teradata Manager and ShowLocks Utility

Following are the differences between Teradata Manager and the ShowLocks utility:

Teradata Manager	ShowLocks Utility
Displays all locks system-wide	Displays only client utility locks
Displays system usage data and other information on a session-by-session basis	Does not display system usage data and other information on a session-by-session basis

Teradata Manager and Query Configuration

The Query Configuration utility and Teradata Manager provide almost the same information about AMPs and PEs. Following are the differences:

Teradata Manager	Query Configuration Utility
	Displays information in many different ways. You can report AMPs and PEs separately, online or offline AMPs separately, and online or offline PEs separately. Or you can report either all online or offline AMPs and PEs.
	Provides information about the TJ of each AMP and also the recovery journal that is maintained so down AMPs can rebuild themselves.
Reports CPU type (386, 486, or Pentium) for up/online AMPs and PEs.	

Teradata Manager and SHOWSPACE Compared

Use the SHOWSPACE command (in the Ferret utility) to determine whether your disks require compression or your system requires expansion.

Both Teradata Manager and the SHOWSPACE command report on total disk utilization per AMP. Otherwise, the two programs are vastly different:

Teradata Manager Reports...	SHOWSPACE Command Reports...
the use of CPU, memory, BYNET, disk usage, and PEs.	only on disk usage.
only the percent of disk utilization per AMP and the number of disk reads and writes.	more disk usage statistics on a per-AMP basis.

For more information on the SHOWSPACE command, see Ferret utility in *Teradata RDBMS Utilities*.

Teradata Manager and TDP Monitoring Commands

You can use the following commands to investigate TDP activity:

This TDP Command...	Displays...
DISPLAY CELLS	cell availability and usage information the TDP internal memory management system generates.
DISPLAY IFP	status of PEs allocated to the TDP
DISPLAY POOL	information and statistics about one or more established session pools
DISPLAY SESSIONS	status of sessions communicating with Teradata through the TDP

These TDP commands display information about all sessions that are active on a specific TDP, while Teradata Manager displays information about all TDP sessions running on the PEs. For more information on the TDP commands, refer to *Teradata TDP Reference*.

Teradata Manager and DISPLAY IFP

Both DISPLAY IFP and Teradata Manager provide information on host reads and writes. However:

Teradata Manager provides...	DISPLAY IFP provides . .
PE data from a Teradata perspective.	PE data from a client perspective.
the following information that DISPLAY IFP does not: • Information on PEs, AMPs, and BYNET • System information, such as the percent busy of a PE and number of PEs on the system • Number of segments read into PE memory from the disk as a result of swapping • Number of memory allocations done per second per PE • Other PE statistics from a Teradata perspective	the following information that Teradata Manager does not: • Whether a PE is disabled • Whether a channel is down

Teradata Manager and DISPLAY POOL

There is no direct Teradata Manager equivalent to the DISPLAY POOL command.

Although Teradata Manager displays many statistics about sessions, it does not single out the sessions as pooled sessions. Unpooled sessions and pooled sessions look the same to Teradata Manager.

Teradata Manager and DISPLAY SESSIONS

Following are the differences between Teradata Manager and DISPLAY SESSIONS:

Teradata Manager Provides...	DISPLAY SESSIONS Provides...
more comprehensive information such as: • Information on blocked sessions and the source of locks. This information is detailed enough to allow you to respond appropriately. • Session run priority. • Logon time and date. • Partition associated with the session. • Number of requests and transactions that the session processes. • Spool space associated with the session. • Amount of CPU time used in all AMPs by this session.	• The last request the session executed. • When a session is switched from one PE to another.

Teradata Manager and HSI, TDPUTCE, and SMF

Teradata Manager has no equivalent functionality to the TDP tools HSI or TDPUTCE, and has no equivalent functionality to MVS SMF information for logon violations and security violations.

For more information about HSI, TDPUTCE and SMF, see *Teradata TDP Reference*.

Teradata Manager and Job Control Support Tools

Introduction

Teradata Manager is one of the Teradata job control support tools. Job control support allows you to control session processing to optimize system performance.

Teradata Manager and LOGOFF

Following are the differences between Teradata Manager and the LOGOFF command:

Teradata Manager Feature	LOGOFF Command Feature
Cannot disable logons.	Disables logons.
Can abort a transaction without logging off the session.	Cannot abort a transaction without logging off the session.
Can log off any session on any TDP. A user can be on one TDP and log off sessions on other TDPs.	Can only log off a session in which a user is on the same TDP as the session that is to be logged off.
Can list the sessions about to be aborted or logged off. The list includes the status of each session, so that if a session cannot be aborted or logged off, the user can tell why.	
Provides more methods of specifying an ABORT. You can abort all sessions on all clients. Further, you can log off a session based on user ID alone.	

Teradata Manager and LOGOFF POOL

Teradata Manager does not specifically identify pooled sessions, although it can log them off and abort them. Teradata Manager, unlike LOGOFF POOL, does not allow active sessions identified with a logoff to run to completion. However, Teradata Manager can abort a transaction without a logoff.

Teradata Manager and Gateway Control Utility

The following Gateway Control utility KILL commands work only on network-attached sessions and users:

Command	Description
KILL USER	Forces off all sessions owned by the specified username. You can list the permissible users by invoking a DISPLAY GTW or DISPLAY SESSION utility.
KILL SESSION	Terminates specific sessions identified by session number. To list the permissible values for the session number, invoke the DISPLAY GTW or DISPLAY NETWORK utility.

Teradata Manager works on both network-attached and channel-attached sessions. Teradata Manager can abort a transaction without logging off the session; the Gateway Control utility cannot.

For more information on the Gateway Control utility, see *Teradata RDBMS Utilities*.

Chapter 12:

SQL and System Performance

This chapter discusses Structured Query Language (SQL) operations and system performance.

Topics include:

- CREATE/ALTER TABLE and data retrieval
- Compressing columns
- Correlated subqueries
- Concatenation and correlated subqueries
- CASE expression
- Ordered analytical extensions
- Extending DATE with the CALENDAR system view
- Optimized INSERT SELECTs
- SQL techniques that reduce row redistribution
- Merge joins
- Hash joins
- Secondary indexes
- Join indexes
- Joins and aggregates on views
- Joins and aggregates on derived tables
- Derived table optimization
- GROUP BY and join optimization
- Outer joins
- Large-table/small-table joins
- Volatile and global temporary tables
- Partitioned Primary Index
- Identity Column
- Collecting statistics
- Referential integrity
- 2PC protocol
- Updatable cursors
- Sparse indexes
- EXPLAIN feature and the Optimizer

Unless otherwise noted, you can find more information on all of these topics in *Teradata RDBMS SQL Reference*.

CREATE/ALTER TABLE and Data Retrieval

Adjusting Table Size

The performance of data retrieval is in direct proportion to the size of the tables being accessed. As table size increases, the system requires additional I/O operations to retrieve or update the table.

If an application need not access every column in a very large table, consider splitting the table into two or more tables for better performance.

Keep in mind, of course, the requirements of other applications. They may need to join the table fragments to obtain needed data. You can use join indexes (see "Join Indexes" on page 12-31) to meet this need efficiently, but at the cost of some overhead and maintenance.

Reducing Number of Columns

As the number of columns increases, the table uses more data blocks and the number of I/O operations increases. Reducing the number of columns increases I/O performance.

If you define a table with indexes that you chose for maximum performance, and if users structure their statements to take advantage of those indexes, satisfactory response should be achieved even on very large tables of more than 100 columns.

Reducing Row Size

The size of a row is based on the total width of all the columns in the table, plus row overhead. The larger the row becomes, the more data blocks are needed, and the more I/O operations are required.

A row cannot span data blocks. If a single row is longer than the current maximum size of a multi-row data block, the system allocates a large data block (up to the system maximum block size) to accommodate this single large row.

See "Value List Compression" on page 12-6 on multiple value compression for fixed width columns.

If a single row exceeds the absolute maximum block size of 127 sectors (V2R4.1 default value), the system returns an error message to the session.

Altering Tables

You can use the ALTER TABLE statement to change the structure of an existing table to improve system performance and reduce storage space requirements.

Reduce the number of bytes in the table to reduce the number of I/O operations for that table.

The following summarizes the effect on table storage space of using ALTER TABLE to perform specific functions. (Resultant changes to the Data Dictionary have a trivial effect on performance.)

See *Teradata RDBMS Database Design* and the *Teradata RDBMS SQL Reference* for more information.

Function	Performance Impact	Space Requirement
Add column (COMPRESS, NULL)	All table rows are changed if a new presence byte is added.	Slight increase
Add column (NO NULL, DEFAULT, and WITH DEFAULT)	All table rows are changed.	Increase
Add column (NULL, fixed-length)	All table rows are changed.	Increase
Add column (NULL, variable length)	All table rows are changed.	Slight increase
Add FALLBACK option	Entire table is accessed to create the fallback copy. Long term performance effects.	Approximately doubled
Adding CHECK CONSTRAINT	Takes time to validate rows, which impacts performance.	Unchanged
Adding Referential Integrity	Takes time to check data. Impacts performance long term. Similar to adding indexes.	Possible great increase
Change format, title, default	No impact.	Unchanged
Changing cylinder FreeSpacePercent (FSP)	No impact.	Increase for BulkLoad operations such as default maximum, MultiLoad, restore
Changing maximum multi-row block size	No impact, unless the IMMEDIATE clause is used, which changes all table blocks.	Slight increase for smaller values; slight decrease for larger values

Function	Performance Impact	Space Requirement
Delete FALLBACK option	FALLBACK subtable is deleted. Long term performance effects.	Approximately half
Drop column	All table rows are changed.	Decrease

DATABLOCKSIZE

You can control the default size for multi-row data blocks on a table-by-table basis via the DATABLOCKSIZE option in the CREATE TABLE and ALTER TABLE statements as follows:

IF you specify...	THEN...
DATABLOCKSIZE in CREATE TABLE	all data blocks of the table are create using DATABLOCKS instead of PermDBSize (see "PermDBSize" on page 5-24).
DATABLOCKSIZE in ALTER TABLE	the datablocks can grow to the size specified. Whether they are adjusted to that new size gradually over a long period of time depends on the use of the IMMEDIATE clause.
the IMMEDIATE clause	the rows in all existing data blocks of the table are repacked into blocks using the newly specified size. For large tables, this can be a time-consuming operation, requiring spool to accommodate two copies of the table while it is being rebuilt. If you do not specify the IMMEDIATE clause, existing data blocks are not modified. As individual data blocks of the table are modified as a result of user transactions, the new value of DATABLOCKSIZE is used. Thus, the table changes over time to reflect the new block size.

To review block size consumption, you can run the SHOWBLOCKS command of the Ferret utility. See *Teradata RDBMS Utilities* for more information on running this command. To specify the global data block size, use PermDBSize (see "PermDBSize" on page 5-24).

Disk arrays are capable of scanning at higher rates if the I/Os are larger. But larger I/Os can be less efficient for row-at-a-time access that require the entire datablock to be read for the relatively few bytes contained in a row.

In general, the benefits of large datablocks with respect to scans outweigh, for the vast majority of workloads, the small penalty associated with row-at-a-time access up to 64K. Setting datablock size requires more consideration at 128K datablocks, where the penalty for row-at-a-time access becomes measurable.

FREESPACE

You can specify the default value for free space left on a cylinder during certain operations on a table-by-table basis via the FREESPACE option in the CREATE TABLE and ALTER TABLE statements.

This allows you to select a different value for tables that are constantly modified versus tables that are only read after they are loaded. To specify the global free space value, use FreeSpacePercent (see "FreeSpacePercent" on page 5-12).

Compressing Columns

Introduction

You can use the ALTER TABLE statement to compress columns and reduce the number of I/O operations.

Consider the following:

- Set the column default value to most frequent value.
- Compress to the default value.
  - This is especially useful for sparsely populated columns.
  - Overhead is not high.
- The I/O savings correlates to the percentage of data compressed out of a row

Value List Compression

Value List Compression (VLC) provides the Teradata database with the capacity to support multiple value compression for fixed width columns. When you specify a values or values, the system suppresses any data matching the compress value from a row. This saves disk space.

Smaller physical row size results in less data blocks and fewer I/Os and improved overall performance.

Multiple Compress Values

Because VLC allows you to specify a list of compress values for a column, the system suppresses data when one of the specified values exists in the column. Up to 255 distinct values (plus NULL) may be compressed per fixed-width column.

Note: If you specify compress values for a column in a partitioning expression and the column is not part of the primary index, you receive error message 3632.

Performance Impact

VLC improves performance as follows:

- Reduces the I/O required for scanning tables when the tables have compressible values in their columns
- Reduces disk space because rows are smaller
- Permits joins to look up the tables to be eliminated

Performance Considerations

VLC will enhance the system cost and performance of the high data volume applications like call record detail and click-stream data, and provide significant performance improvement for general ad-hoc workloads and full-table scan applications.

VLC improves performance depending upon amount of compression achieved by the revised DDL. There appears to be a small cost in terms of processing the compressed data. Select and delete operations show a proportionate improvement in all cases. Inserts and updates show mixed results. The Load operations (FastLoad, MultiLoad and TPump) benefit from the compressed values.

Tables with large number of rows and fields with limited number of unique values are very good candidates for compression. With very few exceptions, the CPU cost overhead for compression processing is minimal. The reduction in the table size depends upon the number of fields compressed and the frequency of the compressed values in the table column. The reduced table size directly translates into improved performance.

Correlated Subqueries

Introduction

A subquery is correlated when it references the columns of outer tables in the enclosing (outer) query. A correlated subquery allows a row in the inner query to refer to a row in the outer query. Columns of outer tables are called outer references.

Performance Value

A correlated subquery eliminates the need for intermediate or temporary tables, which cannot be optimized, and enables the Optimizer to choose the form that provides the best performance. A correlated subquery is fully integrated with global join planning to minimize costs.

Correlated subquery evaluation has two stages:

Stage	Process
1	For each row of the outer query, the system uses the values of the outer references in that row to evaluate the subquery result.
2	The system uses the subquery result to join with the associated row of the outer query, based on the subquery join constraint.

Processing of a query written with CS is significantly faster than the same query using temporary tables. In some cases, CS may cause a query to run twice as fast.

Example 1

You want to find the employee with the highest salary in each department.

```
SELECTlast_name
,salary_amount
FROMemployee ee
WHERESalary_amount =
(SELECT MAX (salary_amount)
FROM employee em
WHERE ee.department_number =
em.department_number);
```

This query executes as follows:

Stage	Process
1	Read the employee row.
2	Get the max department salary specified in the subquery.
3	Compare salary to max salary.
4	If equal, output this row.
5	Go to 1.

Example 2

You want to find job codes to which not employees are assigned. Enter:

```
SELECT job_code FROM job
WHERE NOT EXISTS
(SELECT job code FROM employee ee
WHERE ee.job_code = job.job_code);
```

The following rows are returned:

```
job_code
104202
104201
412103
322101
```

Alternatively, you can use NOT IN:

```
SELECTjob_code
FROM job
WHEREjob_code NOT IN
(SELECT job_code FROM employee);
```

The following rows are returned:

```
job_code
104202
104201
412103
322101
```

Concatenation and Correlated Subqueries

Introduction

Concatenation allows you to retrieve data correlated to the MIN/MAX function in a single pass.

This is a special application of concatenation that precludes the need for a correlated subquery.

Example

For example, you want to find the employee with the highest salary in each department. Your original query might be:

```
SELECT Dept_No
,Salary
,Last_Name
,Fname
FROM Employee_Table
WHERE Dept_No, Salary IN
(SELECT Dept_No, MAX(Salary)
FROM Employee_Table
GROUP BY Dept_No)
ORDER BY Dept_No;
```

You could rewrite the query as:

```
SELECT Dept_No,
MAX(Salary|| ' ' || Lname || ', '||Fname)
FROM Employee_Table
GROUP BY Dept_No
ORDER BY Dept_No ;
```

Note that if two or more employees have the same maximum salary, this query selects only one employee per department.

CASE Expression

Effects on Performance

The CASE expression can provide performance improvements for the following queries:

- For multiple aggregates filtering distinct ranges of values. For example, total sales for several time periods.
- To create two-dimensional reports directly from Teradata. For example, balances in individual accounts held by all bank customers.

CASE expressions help increase performance because they return multiple results in a single pass over the data rather than making multiple passes over the data and then using the client application to combine them into a single report.

You can see performance improvements using the CASE expression as the following increase:

- Number of queries against the same source table(s)
- Volume of data in the source table

Valued and Searched CASE Expression

Use one of the following CASE expression forms to return alternate values based on search conditions:

This form…	Tests…	Example

Valued	an expression against possible values.	Create a catalog entitled "Autumn Sale" that shows spring items marked 33% off and summer items marked 25% off.
		
		`SELECT item_number, item_description,`
		`item_price as "Current//Price"`
		`, CASE item_season`
		`WHEN 'spring' THEN item_price *(1-.33)`
		`WHEN 'summer' THEN item_price *(1-.25)`
		`ELSE NULL`
		`END AS "Sale//Price"`
		`FROM inventory_table;`

Searched	arbitrary expression(s).	Repeat the query above, and mark down by 50% summer items with inventories of less than 3.
		
		`SELECT item_number, item_description,`
		`item_price as "Current//Price"`
		`,CASE`
		`WHEN item_season = 'summer' and`
		`item_count < 3`
		`THEN item_price *(1-.50)`
		`WHEN item_season = 'summer' and`
		`item_count >= 3`
		`THEN item_price *(1-.25)`
		`WHEN item_season = 'spring'`
		`THEN item_price *(1-.33)`
		`ELSE NULL`
		`END AS "Sale//Price"`
		`FROM inventory_table`
		`WHERE item_season in ('spring' or`
		`'summer');`

The following examples illustrate simple code substitution, virtual denormalization, and single pass examples that use the CASE expression.

Example 1: Simple Code Substitution

For example, instead of joining to a description table, use the CASE expression:

```
SELECT CASE region_number
WHEN 1 THEN "North"
WHEN 2 THEN "South"
WHEN 3 THEN "East"
ELSE "West" END
,sum(sales)
FROM sales_table
GROUP BY 1;
```

Example 2: Virtual Denormalization

ABC Telephone Company has a History table with *n* columns, plus call minutes and call type:

- 1 - daytime call
- 2 - nighttime call
- 3 - weekend call

You want a summary of call minutes for each call type for each area code on a single line of output. The standard solution is:

Step	Action
1	Do a GROUP BY on call_type and area code in the History table.
2	Do a self-join to get call_types 1 and 2 into the same row.
3	Do another self-join to get call_type 3 into the same row that contains all 3 call types.

In the classic denormalization solution, you would physically denormalize the History table by putting all three call types in same row. However, a denormalized table requires more maintenance.

Instead, you can use the CASE expression to perform a virtual denormalization of the History table:

```
CREATE View DNV
as Select Col1, … , Col n
,CASE WHEN call_type = 1
THEN call_minutes END (NAMED Daytime_Minutes)
,CASE WHEN call_type = 2
THEN call_minutes END (NAMED Nighttime_Minutes)
,CASE WHEN call_type = 3
THEN call_minutes END (NAMED Weekend_Minutes)
FROM history;
```

Example 3: Single Pass

In this example, you want a report with five sales columns side by side:

- Current Year, Year to Date (Ytd)
- Current Year, Month to Date (Mtd)
- Last Year, Year to Date (LyYtd)
- Last Year, Month to Date (LyMtd)
- Last Year, Current Month (LyCm)

You currently execute five separate SQL statements and combine the result in an application program.

```
Select sum(sales) … where sales_date between 990101 and date;
[Ytd]
Select sum(sales) … where sales_date between 991001 and date;
[Mtd]
```

```
Select sum(sales) … where sales_date between 980101 and ADD_MONTHS
(date, -12); [LyYtd]
Select sum(sales) … where sales_date between 981001 and ADD_MONTHS
(date, -12); [LyMtd]
Select sum(sales) … where sales_date between 981001 and 981031;
[LyCm]
```

Instead, you can use the CASE expression to execute one SQL statement that only makes one pass on the Sales_History table.

```
Select …
sum(CASE WHEN sales_date between 990101 and       date THEN sales
ELSE 0 END),  [Ytd]
sum(CASE WHEN sales_date between 991001 and date THEN sales ELSE
0 END),  [Mtd]
sum(CASE WHEN sales_date between 980101 and ADD_MONTHS (date, -
12) THEN sales ELSE 0 END),[LyYtd]
sum(CASE WHEN sales_date between 981001 and ADD_MONTHS (date, -
12)THEN sales ELSE 0 END),[LyMtd]
sum(CASE WHEN sales_date between 981001 and 981031 THEN sales ELSE
0 END), [LyCm]
from ...
WHERE sales_date between 980101 and date ...
```

When creating views involving UNION ALL set queries, try to include all base table primary index columns in the definition to take advantage of this feature.

Note that hash joins must be disabled to utilize this feature.

Ordered Analytical Functions

Introduction

The Teradata's support for ordered analytical functions allows you to perform computations at the SQL level rather than through a higher-level calculation engine.

Teradata supports:

- Ordered analytical syntax
- Random stratified sampling
- Multiple aggregate distincts

For complete information on ordered analytic functions, see *Teradata RDBMS SQL Reference*.

Ordered Analytical Functions and Performance

Ordered analytical functions, which are extremely helpful in general decision support, speed up order-based analytical type queries.

Using ordered analytical functions, you can target the data analysis within the data warehouse itself. This provides several advantages, including:

- Improved processing performance
- Faster analysis than that performed by external tools and sort routines
- Full access to order analytical functions by external tools such as TeraMiner Stats.

  For example, the TeraMiner FREQ function uses CSUM, RANK, and QUALIFY in determining frequencies.
- Support of ANSI version of existing aggregate functions, enabling you to use RANK, SUM, AVG, and COUNT on multiple partitions within a statement select list.
- Simpler SQL programming, particularly because you can use:
  - Nested aggregates with the HAVING clause
  - SUM and COUNT Window functions
  - QUALIFY, RANK, and ORDER BY clauses

  For example, Teradata permits this query structure:

```
SELECT state, city, SUM(sale),
RANK() OVER
(ORDER BY SUM(sale) PARTITION BY state)
FROM Tbl1, Tbl2
WHERE Tbl1.cityid = Tbl2.cityid
GROUP BY state, city
HAVING MAX(sale) > 10
QUALIFY SUM() OVER
(ORDER BY MIN(sale) PARTITION BY state) > 10;
```

Example: Using Teradata RANK

RANK (sort_expression_list) returns the rank (1..n) of all rows in a group by values of sort_expression_list.

For example, assume you enter this query:

```
SELECT ProdId, Month, Sales, RANK(Sales)
FROM SalesHistory
GROUP BY ProdId
QUALIFY RANK(Sales) <=3;
```

The rows of the response table are ranked as follows:

ProdId	Month	Sales	RANK
1234	9907	500	1
1234	9909	300	2
1234	9908	250	3
…			
5678	9909	450	1
5678	9908	150	2
5678	9907	100	3

This opens up possibilities for applying RANK to non-analytical processing that may be cumbersome otherwise.

For example, RANK can:

- Process data sequentially.
- Generate unique sequential numbers on columns that uniquely define the row.
- Process consecutive rows in a predefined order, when you define a self-join on a ranked table.

  For example:

Step	Action
1	Create a copy of table with a new column containing the rank, based on some ordering criteria; for example: term_eff_date or load_event_id.
2	Define a self-join on the table similar to the following: • `WHERE A.rankvalue = B.rankvalue - 1` • `AND A.policy_id = B.policy_id`
3	Use the self-joined table to process all table rows in a single pass (proceeding from row number *n* to row number *n*+1). This offers significant performance improvement over making multiple passes to process just two rows at a time.

Random Stratified Sampling

Before Teradata V2R5.0, you could extract a random sample from a database table using the SAMPLE clause, and could specify one of the following:

- The number rows
- A fraction of the total number of rows
- A set of fractions as the sample

This sampling method assumed that rows were sampled without replacement, and they were not reconsidered when another sample of the population was taken. This method resulted in mutually exclusive samples when you requested multiple samples. In addition, the random sampling method assumed proportional allocation of rows across the AMPs in the system.

Teradata V2R5.0 enhances random sampling to incorporate, in addition to other sampling options, stratified sampling.

Random Stratified Sampling, also called proportional or quota random sampling, involves dividing the population into homogeneous subgroups and taking a random sample in each subgroup. Stratified sampling represents both the overall population and key subgroups of the population. The fraction specification for stratified sampling refers to the fraction of the total number of rows in the stratum.

The following apply to stratified sampling:

You can specify…	You cannot specify…
stratified sampling in derived tables, views, and macros.	stratified sampling with set operations or subqueries.
either a fraction or an integer as the sample size for every stratum.	fraction and integer combinations.
up to 16 mutually exclusive samples for each stratum.	

Multiple Aggregate Distincts

Before release V2R5.0, the Teradata RDBMS supported only one DISTINCT expression when performing an aggregation. But the Multiple Aggregate Distincts feature in Teradata V2R5.0 allows multiple DISTINCT expressions for aggregates, for example:

```
SEL g, SUM(DISTINCT a), SUM(DISTINCT b)
FROM T
GROUP BY g
HAVING COUNT(DISTINCT c) > 5;
```

The feature simplifies SQL generation.

Extending DATE with the CALENDAR System View

Introduction

Teradata provides a system view named CALENDAR with a date range from the year 1900 to the year 2100. You can extend the properties of the DATE data type by joining CALENDAR to one or more data tables. Also, you can define your own views on CALENDAR.

CALENDAR offers easy specification of arithmetic expressions and aggregation. This is particularly useful in online analytical processing environments, where it is common to request values aggregated by weeks, months, years, and so on.

Example

An example of using CALENDAR to solve this kind of query is shown below. This returns the dollar sales for the current week and the previous week, and for the same weeks last year, for all items in the sportswear class for women:

```
SELECT a2.week_of_calendar, SUM(a1.price)
FROM Sales a1, CALENDAR a2, Item a3, Class a4, TODAY a5,
WHERE a1.calendar_date=a2.calendar_date
AND ( a2.week_of_calendar=a5.week_of_calendar
OR a2.week_of_calendar=a5.week_of_calendar - 1
OR a2.week_of_calendar=a5.week_of_calendar - 52
OR a2.week_of_calendar=a5.week_of_calendar - 53
)
AND a1.itemID=a3.itemID
AND a3.classID=a4.classID
AND a4.classDesc="Sportswear_Women"
GROUP BY a2.week_of_calendar
ORDER BY a2.week_of_calendar
;
```

For complete details on the definition and use of CALENDAR, see the DATE date type in *Teradata RDBMS SQL Reference.*

Optimized INSERT SELECTs

Empty Table INSERT SELECTs and Performance

The INSERT SELECT optimizes performance when the target table is empty. If the target table has data, INSERT SELECT operates on an efficient block-by-block basis that bypasses journaling.

Normally, when the system inserts a row into a table, the system must make a corresponding entry into the TJ to roll back the inserted row if the transaction aborts. If a transaction aborts, the system deletes all inserts from the table one row at a time by scanning the TJ for RowIDs.

If the transaction aborts when the table into which rows are inserted is empty, the system can easily return the table to its original state by deleting all rows. Scanning the TJ is superfluous, and writing RowIDs to delete becomes unnecessary.

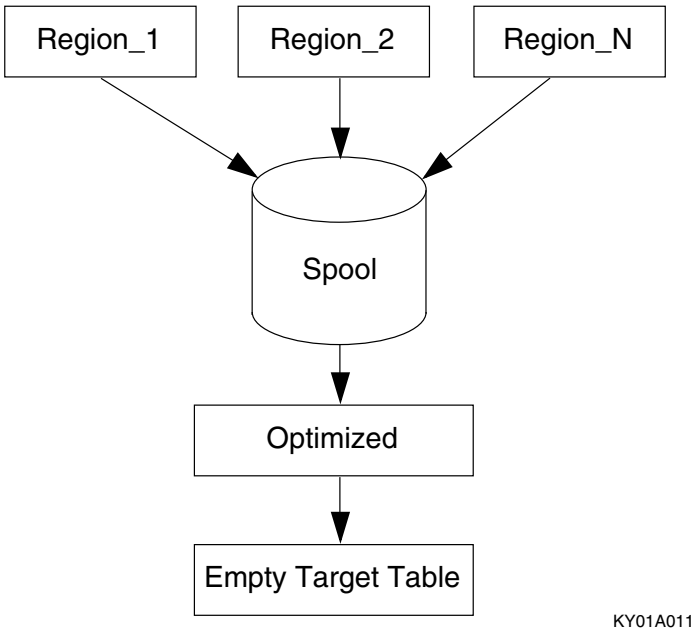
The advantages of using optimized INSERT SELECT are:

- Block-at-a-time processing
- Faster insert logic (that eliminates block merge complexity)
- Instantaneous rollback for aborted INSERT SELECT statements

Example

Using multiple Regional Sales History tables, build a single summary table by combining summaries from the different regions, and then insert these summaries into a single table via a multi-statement INSERT SELECT statement.

All multi-statement INSERT SELECT statements output to the same spool table. The output is sorted and inserted into an empty table.



Form a multi statement request by semicolon placement in BTEQ as shown below, or by placing statements in a single macro.

Note: If you execute each of the statements separately, only the first statement is inserted into an empty table.

```
INSERT into Summary_Table
SELECT store, region,sum(sales),count(sale_item)
FROM Region_1
GROUP BY  1,2

;INSERT into Summary_Table
SELECT region2, sum (sales), count(sale_item)
FROM Region_2
GROUP BY     1,2
. . .

;INSERT into   Summary_Table
SELECT region3, sum(sales), count(sale_item)
FROM Region_N
GROUP BY     1,2
;
```

INSERT SELECT Into SET Table

INSERT SELECT into a SET table from a source known not to have duplicate rows avoids duplicate checking of the target table during insertion. This occurs even during direct insertion from another SET table.

This should offer significant performance improvement in cases where there is a NUPI that is relatively non-unique or has few values that are very non-unique.

INSERT SELECT with FastLoad

Use the optimized INSERT SELECT to manipulate FastLoaded data:

Step	Action
1	FastLoad into a staging table.
2	INSERT SELECT into the final table, manipulating the data as required.

FastLoad and INSERT SELECT is faster than using an INMOD to manage data on the host, because the host is a single bottleneck as opposed to parallel AMPs that populate temporary tables for reports or intermediate results.

Multiple source tables may populate the same target table. If the target table is empty before a request begins, all INSERT SELECT statements in that request run in the optimized mode.

INSERT SELECT with Join Index

The fastest way of processing inserts into a table with a join index is as follows:

Step	Action
1	Use FastLoad to load the rows into an empty table
2	Do an INSERT SELECT from the freshly-loaded table into the target table with the join index

Processing for both steps is performed a block at a time and should provide the best throughput.

SQL Techniques that Reduce Row Redistribution

Introduction

This section discusses:

- Extracting combinations of join columns
- Using the BETWEEN date comparison

to achieve fewer row redistributions.

Extracting Combinations of Join Columns

Extracting all combinations of join columns, when these combinations are much less than the number of rows, will help achieve fewer row redistributions.

For example, the standard approach as follows:

Step	Action
1	Load daily sales data (5 million rows from 50 stores) into the Work table.
2	Join the Work table to the base reference tables to populate additional columns.
3	Eventually, insert the final Work table into the History table.

In this example, Step 2 joins the Work table to a reference Item table (120,000 rows):

Stage	Process
1	Join the Work table to the Item table by redistributing the 5 million row Work table on item_no to get Item data.
2	Redistribute 5 million rows back to insert into another temporary Work table.

A better method is the following:

Stage	Process
1	Extract the distinct item numbers from the temporary Work table (~100,000 rows).
2	Redistribute 100,000 rows instead of 5,000,000 rows to get the item data.
3	Redistribute 100,000 rows back to join to Work table.

Using the BETWEEN Clause

When considering a time interval, use the BETWEEN clause of MIN and MAX dates of an interval.

For example, a reference calendar contains 730 rows with:

- calendar_date
- fiscal_week
- fiscal_month
- fiscal_quarter

In this example, you want summary data from a History table for fiscal_quarter. A standard query would be:

```
SELECT
 H.item_code
 , SUM(H.items_sold)
, SUM(H.sales_revenue)
FROM History H , Calendar C

WHERE C.fiscal_quarter = "3Q99"
AND   C.calendar_date = H.sale_date

GROUP BY H.item_code
ORDER BY H.item_code ;
```

From a performance perspective, this query would:

Stage	Process
1	Build a spool table with dates from the reference calendar (90 days).
2	Duplicate the calendar spool. Either: • Product join the calendar spool with the History table (90 compares/history table row). • Sort both tables to do merge join. Alternatively, redistribute the entire History table. Product join the large table with the calendar spool (~1 row /AMP).

Another approach is to denormalize the History table. Add fiscal_week, fiscal_month, fiscal_quarter to the History table. Qualify fiscal_month directly in the denormalized table. The penalties for using this approach include:

- Denormalization maintenance costs are higher
- Extra bytes require more I/Os

Example: BETWEEN Clause

The solution is to rewrite the query:

```
SELECT H.item_code
, SUM(H.items_sold)
, SUM(H.sales_revenue)
FROM History H ,
(SELECT min(calendar_date)
, max(calendar_date)
FROM Calendar
WHERE fiscal_quarter = "3Q99") AS DT (min_date, max_date)
WHERE H.sale_date BETWEEN
DT.min_date and DT.max_date
GROUP BY H.item_code
ORDER BY H.item_code ;
```

From a performance perspective, the Optimizer could:

Stage	Process
1	Build a spool table with a single row containing the first and last dates of fiscal_quarter.
2	Duplicate one row spool. Product join one row spool with the History table (2 compares/History table row).

One customer reported that a query that typically took 3 hours ran in about 12 minutes.

The benefits of using the BETWEEN date comparison are:

- Reducing multiple comparisons from as many dates in the date interval down to 2/row, and saving sort or redistribution of a large table.
- Not having to denormalize

  Using the BETWEEN data comparison is faster than reading extra denormalized table bytes.

Note that in either case, the system must read all rows.The cost of reading extra denormalized table bytes is greater than the cost of building one row spool with MIN and MAX dates.

Merge Joins

Compared with Nested Join

In a large join operation, a merge join requires less I/O and CPU time than a nested join. A merge join usually reads each block of the inner table only once, unless a large number of hash collisions occur.

A nested join performs a block read on the inner table for each outer row being evaluated. If the outer table is large, this can cause each block of the inner table to be read multiple times.

Merge Join with Covering NUSI

When large outer tables are being joined, a merge join of the base table with a covering index can realize a significant performance improvement.

The Optimizer considers a merge join of a base table with a covering NUSI, which gives the Optimizer an additional join method and costing estimate to choose from.

Hash Joins

Introduction

The Optimizer may use a hash join instead of a merge join (of tables) for better performance:

- If at least one join key is not indexed.
- To provide a 10-40% performance improvement for the join step.

  Note: Since the join is only part of a query, you may not see a 40% improvement in the entire query.

The hash join eliminates the sort used prior to the merge join by using a hash table instead.

You can enable hash joins with the following DBS Control fields:

- HTMemAlloc (see page 5-13)
- SkewAllowance (see page 5-39)

Recommendations

Most sites should use the following values:

- HTMemAlloc = 1
- Skew Allowance = 75

Consider a different setting if:

- The system is always very lightly loaded. In this case, you may want to increase HTMemAlloc to a value between 2 and 5.
- Data is so badly skewed that the hash join degrades performance. In this case, you should turn the feature off or increase the Skew Allowance to 80 or 90.

Secondary Indexes

Secondary Indexes and Performance

Secondary indexes supply alternate access paths, which increases performance. For best results, base secondary indexes on frequently used set selections and on an equality search. The Optimizer may not use a secondary index if it is too weakly selective.

Statistics play an important part in optimizing access when NUSIs define conditions for the following:

- Joining tables
- Satisfying WHERE constraints that specify comparisons, string matching, or complex conditionals
- Satisfying a LIKE expression
- Processing aggregates

Because of the additional overhead for index maintenance, index values should not be subject to frequent change. When you change a secondary index, the system:

Stage	Process
1	Deletes any secondary index references to the current value (AMP-local for NUSIs, and across AMPs for USIs).
2	Generates new secondary index references to the new value (AMP-local for NUSIs, and across AMPs for USIs).

Using NUSIs

The guiding principle for using NUSIs is that there should be fewer rows that satisfy the NUSI qualification condition than there are data blocks in the table. Whether the Optimizer uses a NUSI depends on the percent of rows per NUSI value, as follows:

IF this many rows qualify...	THEN...
< 1 per block	NUSI access is faster than full table scan. For example, if there are 100 rows per block and 1 in 1000 rows qualify, the Optimizer reads 1 in every 10 blocks. NUSI access is faster.
>= 1 per block	full table scan is faster than NUSI access. For example, if there are 100 rows per block and 1% of the data qualifies, the Optimizer reads almost every block. Full table scan may be faster.

In some instances, values are distributed unevenly throughout a table. Some values represent a large percent of the table, other values have few instances. When values are distributed unevenly, the system:

- Performs a full table scan on values that represent a large percent of table
- Uses the NUSI for the rest of the values

To query index non-uniqueness, you might enter the following:

```
.set retlimit 20
sel <index column(s)>,count(*) from <tablename>
group by 1 order by 2 desc;
```

NUSIs and Blocksize

NUSIs need to have a larger ratio of rows that do not qualify versus those that do. Consider, for example, 100-byte rows; with a maximum block size of 31.5 KB, each multi-row data block contains approximately 315 rows.

For the NUSI to be effective, fewer rows must qualify than there are blocks in the table. This means fewer than 1 in 315 rows can qualify for a given NUSI value if the index is to be effective.

When the maximum block size is 63.5 KB, fewer than 1 in 635 rows can qualify for the NUSI to be effective. When the maximum block size is 127.5 KB, 1 in 1275 rows can qualify for the NUSI to be effective.

To reset the global absolute-maximum size for multi-row data blocks, see "PermDBSize" on page 5-24.

Index Access

To determine if the Optimizer will use an index to process a request, include a WHERE constraint based on an index value, and use EXPLAIN to determine whether the value affects path selection. See "Teradata Index Wizard" on page 8-7.

Even when you use an index constraint, equivalent queries formulated with different syntax can result in different access plans. The Optimizer may generate different access paths based on the forms given below; depending on the expression, one form may be better than the other.

```
Form 1.(A OR B) AND (C OR D)

Form 2.(A AND C) OR (A AND D) OR (B AND C) OR (B AND D)
```

In expressions involving both AND and OR operators, the Optimizer generates the access path based on the form specified in the query. It does not attempt to convert from one form to another to find the best path. Consider the following expression:

```
(NUSI = A OR NUSI = B) AND (X = 3 OR X = 4)
```

In this case, Form 1 is optimal, because the access path consists of two non-unique secondary index (NUSI) SELECTs with values of A and B. The Optimizer applies (`X=3 OR X=4`) as a residual condition. If the Optimizer uses Form 2, the access path consists of 4 NUSI SELECTs.

In the following expression:

```
(NUSIA = 1 OR NUSIA = 2) AND (NUSIB = 3 OR NUSIB = 4)
```

the collection of (NUSIA, NUSIB) comprises a NUSI. In this case, Form 2 is optimal because the access path consists of 4 NUSI SELECTs, whereas the Form 1 access path requires a full-table scan.

Assume an expression involves a single field comparison using IN, such as the following:

```
Field IN (Value1, Value2, ...)
```

The Optimizer converts that expression to:

```
Field = Value1 OR Field = Value2 OR ...
```

Therefore, the Optimizer generates the same access path for either form. However, if an expression involves a multiple field comparison using IN, such as in the following query,

```
a.(Field1 IN (Value1 OR Value2 OR ...)
AND Field2 IN (Value3 OR Value4 OR ...)
```

then the Optimizer converts the expression to:

```
b.(Field1 = Value1 OR Field1 = Value2 OR ...)
AND (Field2 = Value3 OR ...)
```

Note how the converted form differs from the following (which is in Form 2):

```
c.(Field1 = Value1 AND Field2 = Value3)
OR (Field1 = Value2 AND Field2 = Value4)
OR ...
```

Index Access Guidelines

Generally, Teradata follows these guidelines for index access:

Teradata uses...	To...
Unique Primary Index (UPI)	ensure fastest access to table data.
Unique Secondary Index (USI)	process requests that employ equality constraints.

Teradata uses…	To…
UPIs to match values in one table with index values in another	ensure optimal join performance.
PI	satisfy an equality on IN condition in a join.
Non-Unique Primary Index (NUPI)	• Perform a single-disk row selection or join process • Avoid sorting or redistributing rows.
information from a single AMP	estimate the cost of using an index when statistics are not available. This assumes an even distribution of index values (an uneven distribution affects performance).
index based on more than one column (a composite index) only	process requests that employ equality constraints for all fields that comprise the index. You can define an index on a column that is also part of a multi-column index.
bitmapping	process requests only when equality or range constraints involving multiple NUSIs are applied to very large tables.

For smaller tables, the Optimizer uses the index estimated to have the least amount of rows per index value.

Using appropriate secondary indexes for the table can increase the retrieval performance for the table, but the trade-off is that the update performance can decrease.

Join Indexes

Introduction

A join index is a data structure that stores and maintains the result of joining, with or without aggregation:

- The columns of two or more tables
- Two or more columns of a single table

The guidelines for creating a join index are the same as those for defining any regular join query that is frequently executed or whose performance is critical. The only difference is that for a join index the join result is persistently stored and automatically maintained.

Performance and Covering Indexes

Typically, query performance improves any time a join index can be used instead of the base tables. A join index is most useful when its columns can satisfy, or cover, most or all of the requirements in a join query. For example, the Optimizer will consider using a covering index for a merge join.

Covering indexes improve the speed of join queries. The extent of improvement can be dramatic, especially for queries involving complex, large-table, and multiple-table joins. The extent of such improvement depends on how often an index is appropriate to a query.

Multi-Table Non-Covering Join Index

In Teradata V2R5.0, queries are optimized to use a join index on a set of joined tables, even if the index does not cover completely the columns referenced in the table, when:

- The index includes either the RowID or the columns of a unique index of the table containing a non-covered column referenced by the query.
- The cost of such a plan is less than other plans.

A multi-table non-covering join index provides some of the query improvement benefits that join indexes offer without having to replicate all the columns required to cover the queries.

There is, however, the additional overhead of accessing the base table row when a non-covered column is required in the query.

Using Not-Case-Specific Columns in Covering Indexes

If you include the ALL option when creating a join index, the original case of the column is stored in the index. During processing it is extracted from the index rather than the base table.

Allowing a column declared as not case specific to be part of a covering index provides the Optimizer with one more index choice.

Covering Bind Terms

If the connecting condition of a subquery is IN and the field it is connecting to in the subquery is unique, you can define a join index on the connected fields. This provides one more type of index for the Optimizer to consider using in place of multiple base tables.

Using Single-Table Join Indexes

Single-table join indexes are valuable when your applications often join the same large tables. A single-table join index partitions all or a subset of a base table using a primary index based on the table's foreign key (preferably the primary index of the table to which it is to be joined).

Use of such an index greatly facilitates join processing of large tables, because the single-table index and the table with the matching primary index both hash to the same AMP.

The Optimizer evaluates whether a single-table join index can replace its base table even when the base table is referenced in a subquery (unless the index is compressed and the join is complex, such as an outer join or correlated subquery join).

Defining Join Indexes with Outer Joins

With very large tables, also consider defining a non-aggregate join index with an outer join. This approach offers the following benefits:

- For queries that reference only the outer tables, an outer-join index makes available the same performance benefits as a single-table join index.
- Unmatched rows are preserved.

Using Join Indexes with EXTRACT and Inequality Conditions

When defining join index conditions, the following are allowed:

- Inequality conditions

  To define inequality conditions between two columns of the same type, either from the same table or from two different tables, you must AND these with the rest of the join conditions.
- EXTRACT expression

These capabilities expand the usefulness of join indexes, because the Optimizer can more often choose to resolve a query with a join index rather than by accessing the data tables.

Using Aggregate Join Indexes

Aggregate join indexes offer an extremely efficient, cost-effective method of resolving queries that frequently specify the same aggregate operations on the same column or columns. When aggregate join indexes are available, the system does not have to repeat aggregate calculations for every query.

You can define an aggregate join index on two or more tables, or on a single table. A single-table aggregate join index includes:

- A columnar subset of a base table
- Additional columns for the aggregate summaries of the base table columns

You can create an aggregate join index using:

- SUM function
- COUNT function
- GROUP BY clause

The following restrictions apply to defining an aggregate join index:

- Only COUNT and SUM are valid, in any combination. (COUNT DISTINCT and SUM DISTINCT are invalid.)
- Always type the COUNT and SUM fields as FLOAT to avoid overflow.

  The system enforces this restriction as follows:

IF you …	THEN the system …
do not define an explicit data type for a COUNT or SUM field	assigns the FLOAT type to it automatically.
define a COUNT or SUM field as anything other than FLOAT	returns an error and does not create the aggregate join index.

Considering Multiple Join Indexes

For each base table in a query, the Optimizer performs the following:

In this phase…	The system...
Qualification	examines the two join indexes that replace the most tables and chooses the one that generates the best plan. Qualification for the best plan includes one or more of the following benefits: • Smallest size to process • Most appropriate distribution • Ability to take advantage of covered fields within the join index
Analysis (of results)	determines if this plan will result in unique results, analyzing only those tables in the query that are used in the join index.

Subsequent action depends on this analysis, as follows:

IF the results will be...	THEN the Optimizer...
Unique	skips the sort-delete steps used to remove duplicates.
Non-unique	determines whether eliminating all duplicates can still produce a valid plan, recognizing any case where: • No field_name parenthetical clause exists • All logical rows will be accessed

Protecting Your Join Index with Fallback

You can define fallback protection for a simple or an aggregate join index.

With fallback, a join index and the base table it references can be accessed if an AMP fails, and there is little impact on performance.

Without fallback, an AMP failure has significant impact on both availability and performance, as follows:

- You cannot update the base table referenced by a join index, even if the base table itself is defined with fallback
- A join index cannot be accessed by queries, and performance may be degraded significantly

The cost is a slight degradation when processing a DML statement that modifies a base table referenced by the join index, because the fallback copy of the join index must also be maintained.

Join Indexes and Collecting Statistics

To provide the Optimizer with the information needed to generate the best plans, you need to have collected statistics on the primary index columns of each join index.

Consider collecting statistics to improve performance during:

- Creation of a join index
- Update maintenance of a join index

Note: Column statistics for join indexes and their underlying base tables are *not* interchangeable.

You need to submit separate COLLECT STATISTICS statements for the columns in the join index and the source columns in the base tables, because index subtables and data tables are seen as separate entities to the Optimizer. (Also see "Collecting Statistics" on page 12-53.) This should not exact a very high cost, because Teradata can collect statistics while queries are in process against the base table.

Performance Benefits

Queries that use join indexes can run many times faster than queries that do not use join indexes. Covering indexes should perform at the higher end. Aggregate join indexes perform much better than join indexes in all areas.

In-place join indexes (where the columns of the covering index and the columns of the table to which it is to be joined both reside on the same AMP) outperform indexes which require row redistribution. An in-place, covering, aggregate join index that replaces two or more large tables in queries with complex joins, aggregations and redistributions can cause a query to run hundreds of times faster.

Cost Considerations

Join indexes, like secondary indexes, incur both space and maintenance costs. For example, insert, update, and delete operations must be performed twice, once for the base table and once for the join index

Space

The following formula provides a rough estimate of the space overhead required for a join index:

Join Index Size = U * (F + O + (R * A)

where:

Parameter	Description
F	Length of the fixed field <join-index-field1>
R	Length of a single repeating field <join-index-field2>
A	Average number of repeated fields for a given value in <join-index-field1>
U	Number of unique values in the specified <join-index-field1>
O	Row overhead (assume 14 bytes)

Note: Updates to the base tables can cause a physical join index row to split into multiple rows. The newly formed rows each have the same fixed field value but contain a different list of repeated field values.

It is important to know that the system does *not* automatically recombine split rows. To re-compact such rows, you must drop and recreate the join index.

Maintenance

The use of an aggregate join index entails:

- Initial time consumed to calculate and create the index

- Whenever a value in a join-index column of the base table is updated, time to recalculate the aggregate and update the index.

However, if join indexes are suited to your applications, the improvements in query performance can far outweigh the costs.

Join indexes are maintained by generating additional AMP steps in the execution plan. Those join indexes defined with outer joins will usually require additional steps to maintain any unmatched rows.

Overhead for an in-place aggregate join index can be perhaps three times more expensive than maintaining the same table without that index. For an aggregate join index that redistributes rows, the maintenance overhead can be several times as expensive.

Maintenance overhead for join indexes without aggregates can be as much as twenty times or more as expensive than maintaining the table without the index. The overhead is greater at higher hits per block, where "hits" is how many rows in a block are touched.

Since the Teradata writes a block only once regardless of the number of rows modified, as the number of hits per block increases:

- The CPU Path/Transaction decreases (faster for the case with no join index than for the case with a join index)
- Maintenance overhead for aggregate join indexes decreases significantly

If a DELETE or UPDATE statement specifies a search condition on the primary or secondary index of a Join Index, the Join Index may be directly searched for the qualifying rows and modified accordingly.

This direct-update approach is employed when the statement adheres to these requirements:

- A primary or secondary access path to the join index
- If a <join-index-field2> is defined, little or no modification to the <join-index-field1> columns
- No modifications to the join condition columns in the join index definition
- No modifications to the primary index columns of the join index.

Join Index Versus NUSI

A join index offers the same benefits as a standard secondary index in that it, like the standard secondary index, is:

- Optional
- Defined by you
- Maintained by the system
- Transparent to the user
- Immediately available to the Optimizer
- If a covering index, considered by the Optimizer for a merge join

- Reported by the HELP INDEX and SHOW TABLE statements

However, a join index offers the following performance benefits:

IF a join index is...	THEN performance improves by...
defined using joins on one or more columns from two or more base tables.	eliminating the need to perform the join step every time a joining query is processed.
used for direct access in place of some or all of its base tables, if the Optimizer determines that it covers most or all of the query. **Note:** A standard secondary index just points to the primary data in the base tables.	eliminating the I/Os and resource usage required to access the base tables.
value-ordered on a column of your choice, such as Date	allowing direct access to the join index rows within the specified value-order range.
a single-table join index with a foreign-key primary index	reducing I/Os and message traffic because row redistribution is not required, since the following are hashed to the same AMP: • A single-table join index having a primary index based on the base table's foreign key • The table with the column(s) making up the foreign key
defined with an outer join	• Giving the same performance benefits as a single-table join index, for queries that reference only outer tables. • Preserving unmatched rows.
created using aggregates	eliminating both the aggregate calculation(s) and the join step for every query requiring the join and aggregate.

See also "Secondary Indexes" on page 12-27.

For more information on the syntax, applications, restrictions, and benefits of join indexes, see:

- *Teradata RDBMS SQL Reference*
- *Teradata RDBMS Database Design*

Joins and Aggregates On Views

Views and Performance

Teradata can perform joins and aggregate queries on view containing aggregates, eliminating the need for temporary tables.

The overhead associated with temporary tables decreases because you can eliminate:

- Creating temporary tables
- Deleting temporary tables
- A set of I/O (read and write) from and to the temporary table

Operations Available with Joins and Aggregates on a View

When you perform joins and aggregate queries on a view, you can:

- Use aggregated values in arithmetic expressions
- Perform an aggregate on aggregations
- Perform an aggregate before a join to replace code values with names
- Control the join order of some of the tables
- Save building some temporary tables

A view might contain a date, a source and a destination, a count and some sums:

Business	View Columns
Airline	week, from_city, to_city, # flights, # passengers, # empty seats, revenue
Telco	month, from_city, to_city, # calls, # minutes, # dropped calls, revenue
Manufacturer	day, from_city, to_city, # shipments, # items shipped, # returns, revenue

Example 1

You want to create a report for a set of times and for each destination that includes an average and a maximum value of the count and sums. The purpose of the report is to determine potential loss of revenue by destination. To create this report, enter:

```
CREATE VIEW Loss_Summary_View
(week, from_code, to_code, count_a, sum_x, sum_y,  sum_z)

AS SELECT
C.week, H.from_code, H.to_code, COUNT(H.a),
SUM(H.x), SUM(H.y), SUM(H.z)
FROM History H ,    Calendar C
WHERE C.month = 199910
AND       C.day      = H.day
GROUP BY 1, 2, 3 ;

SELECT LSV.week, LD.to_location
, AVG(LSV.count_a), MAX(LSV. count_a)
, AVG(LSV. sum_x),  MAX(LSV. sum_x)
, AVG(LSV. sum_y),  MAX(LSV. sum_y)
, AVG(LSV. sum_z),  MAX(LSV. sum_z)

FROM  Loss_Summary_View LSV, Location_Description LD

WHERE LSV.to_code = LD.to_code

GROUP BY 1, 2;
```

Example 2

In the following example, join the CustFile table with the CustProdSales view (which contains a SUM operation) in order to determine which companies purchased more than $10,000 worth of item 123:

```
CREATE VIEW CustProdSales (custno, pcode, sales)
AS
SELECT custno, pcode, SUM(sales)
FROM SalesHist
GROUP BY custno, pcode;

SELECT company_name, sales
FROM CustProdSales a, CustFile b
WHERE a.custno = b.custno
AND
a.pcode = 123
AND
a.sales > 10000;
```

Joins and Aggregates on Derived Tables

What are Derived Tables?

A derived table is the resulting answer set of a SELECT statement in the FROM clause of another SELECT statement.

Derived Tables and Performance

Derived tables provide the same benefit as joins and aggregations on views, plus the flexibility to be free of predefined views. This is important if your query runs against a temporary table. You cannot create a view referencing a table that does not exist, but you can use derived tables as ad hoc queries once the tables have been created.

You can do away with creating temporary tables for specific queries by using a derived table in the FROM clause of the SELECT statement.

Example 1

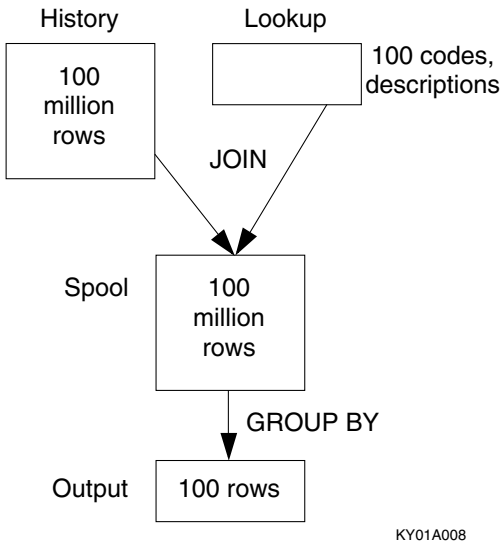
The derived table in the example query performs MAX and AVG functions on columns aggregated via COUNT and SUM functions.

```
SELECT LSV.week, LD.to_location
, AVG(LSV.count_a), MAX(LSV. count_a)
, AVG(LSV. sum_x),  MAX(LSV. sum_x)
, AVG(LSV. sum_y),  MAX(LSV. sum_y)
, AVG(LSV. sum_z),  MAX(LSV. sum_z)
FROM
(SELECT
C.week ,  H.from_code ,  H.to_code
, COUNT(H.a), SUM(H.x), SUM(H.y), SUM(H.z)
FROM History H ,    Calendar C
WHERE C.month = 199809
AND       C.day      = H.day
GROUP BY 1, 2, 3) AS LSV
(week, from_location, to_location
, count_a, sum_x, sum_y, sum_z)
, Location_Description LD
WHERE LSV.to_code = LD.to_code
GROUP BY 1, 2;
```

Example 2

You want to create a report that summarizes sales by code with a description of code. Following is an example of query syntax and processing:

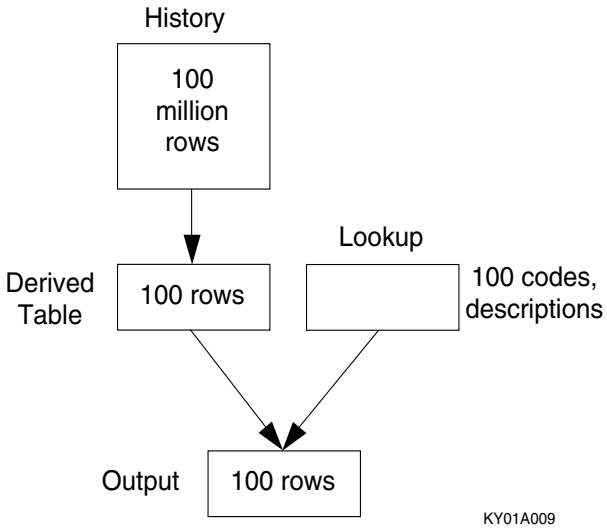
```
SELECT A.code, B.description,
SUM(A.sales)
FROM History A, CodeLookup B
WHERE A.code = B.code
GROUP BY 1, 2 ;
```



Following is an example of a query using derived tables:

```
SELECT DT.code
, B.description
 , DT.sales
FROM (SELECT A.code, SUM(A.sales)
FROM History A
GROUP BY A.code) AS
DT (code, sumsales)
, CodeLookup B
WHERE DT.code = B.code ;
```

This query process is illustrated below.



Derived Table Optimization

Introduction

The Optimize Handling of Derived Tables feature allows the Optimizer to apply certain optimizations to derived tables.

Since the introduction of derived tables in the Teradata RDBMS, the Optimizer materializes them before performing any other operation on the tables. That means that the Optimizer does not perform block integration or constraint optimizations.

Semantically, derived tables are similar to views, in that both are logical definitions of an SQL query. In the case of views, however, the Optimizer does attempt block integration and constraints optimization. For example, the Optimizer can integrate the view with other blocks of the query or perform constraints optimizations for aggregate views and views defined by a UNION.

The Optimize Handling of Derived Tables feature ensures that the Optimizer applies block integration and constraints optimizations to derived tables.

Performance

The Optimize Handling of Derived Tables feature gives a performance boost to derived table joins when the tables are joined on the primary key of the base table.

GROUP BY Operator and Join Optimization

Introduction

In previous Teradata database releases, the Optimizer performed all the joins first and then performed the aggregation. When tables were large, spool space might become a problem.

Partial GROUP BY

Teradata V2R5.0 contains enhancements that allow the Optimizer to consider automatically a partial GROUP BY in order to reduce the number of working rows as early as possible. As soon as the number of working rows is reduced, the space problem is solved.

Performance Impact

The GROUP BY Operator feature provides the following performance improvements:

- Query performance improves because unnecessary spool file redistributions are eliminated.
- After the Optimizer performs a partial GROUP BY, subsequent join steps run faster because they process fewer aggregated rows instead of all data rows.

Outer Joins

Introduction

An outer join, implemented either as a merge join or a product join, effectively performs an inner join and returns rows from the outer table that did not meet the join conditions, extending the results rows with NULLs for their non-matching fields.

Types of Outer Joins

The type of outer join you specify determines which table(s) contribute rows that did not meet the join conditions:

Outer Join Type		Returns All Rows From	
LEFT	TableX LEFT OUTER JOIN TableY	Left table	TableX
RIGHT	TableX RIGHT OUTER JOIN TableY	Right table	TableY
FULL	TableX FULL OUTER JOIN TableY	Both tables	TableX, TableY

For example, you want to join TableX and TableY. TableX contains:

Col1	Col2
1	Blue
1	Red
2	Red

TableY contains:

ColA	ColB
1	$15.97
2	$6.32
3	$1.49

You enter the following:

```
SELECT *
FROM TableX RIGHT OUTER JOIN TableY ON
TableX.Col1 = TableY.ColA;
```

This is the result:

```
Col1Col2ColAColB
1Blue1$15.97
2Red1$15.97
2Red2$6.32
??3$1.49
```

Note that question marks signify NULLs.

Large Table/Small Table Joins

Introduction

Large Table/Small Table (LT/ST) joins combine three or more small tables with one large table.

The Optimizer algorithm:

- Looks for the large relation
- Analyzes connections to each index
- Analyzes non-indexed case

The Optimizer must collect statistics on:

- Join and select indexes
- Small table PIs
- Selection columns, especially if the join is highly selective
- Join columns, especially if the join to the large table is weakly selective

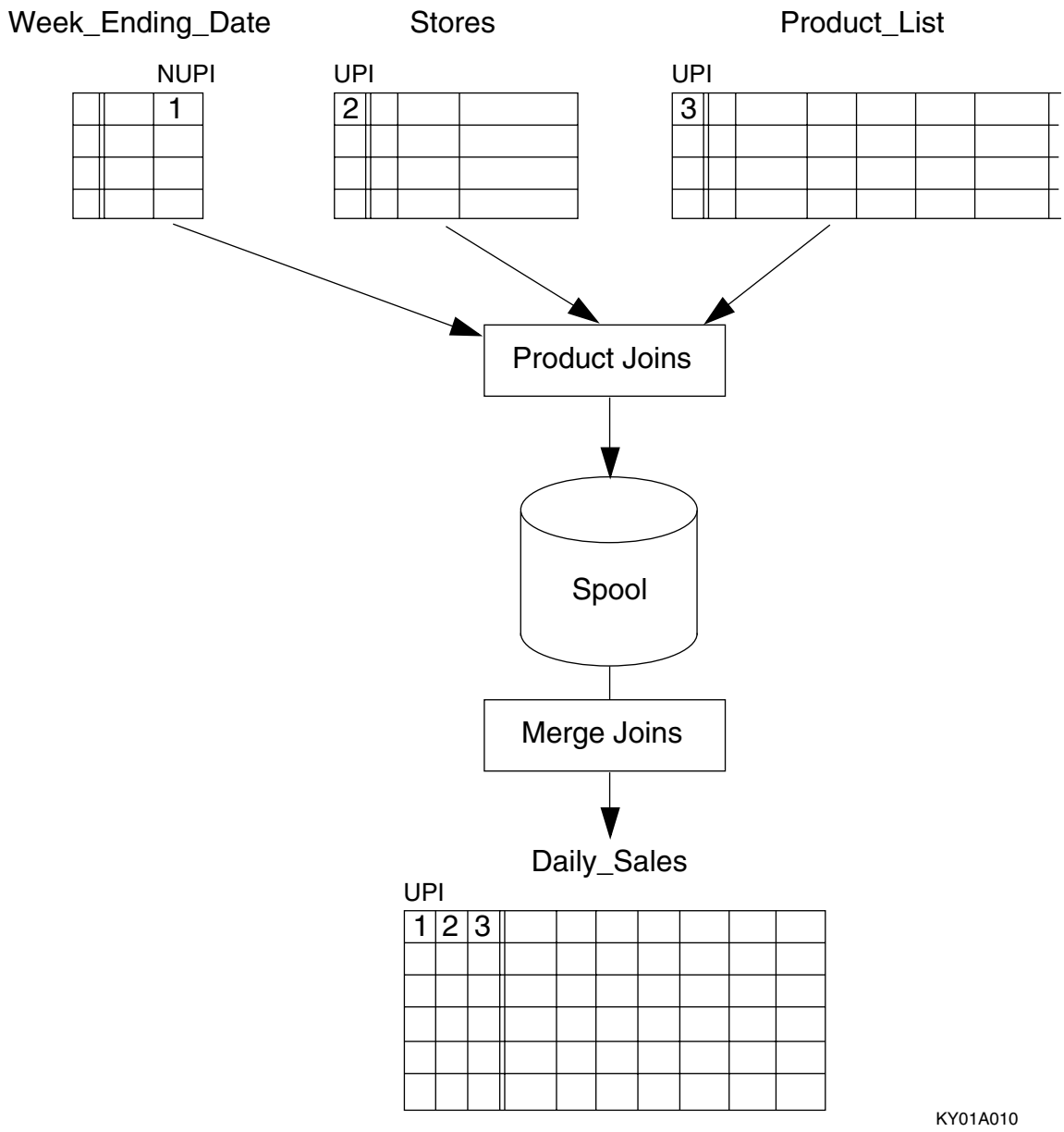
Consider the following points about LT/ST and indexes:

- Indexes are a much more important part of the join than in previous releases.
- Reconsider the decision not to index columns.
- Reconsider indexes on common-join column sets in large tables.

If the PI of a large table can be made up of elements from the small tables, the Optimizer uses a product join on the small tables. With the PI of the large table, the Optimizer can do a merge join and not to read the entire large table, which is much more efficient system resources use.

Example

For example, you want to examine the sales of five products at five stores for a one week time period. This requires joining the Store table (Table B), the Week_Ending table (Table A), and the Product_List table (Table C) with the Daily_Sales table (Table L). The following figure illustrates this join.



Selected portions of the Store table, Week_Ending table and Product_List table are product-joined. The result creates the PI for the Daily_Sales table. The joined small tables are now joined with the large table and an answer set is returned. The new algorithm uses significantly fewer system resources and requires less processing time.

Volatile and Global Temporary Tables

Introduction

Volatile and global temporary tables are similar:

- Each instance of is local to a session.
- The system automatically drops the tables at session end.
- Both have LOG and ON COMMIT PRESERVE/DELETE options.
- Materialized table contents are not sharable with other sessions.
- The table starts out empty at beginning of session.

For more information:

Operation	See
Volatile temporary tables	• *Teradata RDBMS SQL Reference* • *Teradata RDBMS Release Summary - V2R4.0* • *Introduction to Teradata RDMBS*
Global temporary tables	• *Teradata RDBMS SQL Reference* • *Teradata RDBMS Release Summary - V2R4.0* • *Teradata RDBMS Data Dictionary*

Volatile Temporary Tables

Volatile temporary tables are similar to derived tables:

- Materialized in spool.
- No Data Dictionary access or transaction locks.
- Table definition kept in cache.
- Designed for optimal performance.

Unlike derived tables, volatile temporary tables:

- Are local to the session, not the query
- Can be used with multiple queries in the session
- Can be dropped manually anytime or automatically at session end
- Require CREATE VOLATILE TABLE statement

Global Temporary Tables

Global temporary tables require the CREATE GLOBAL TEMPORARY statement.

Unlike volatile temporary tables, global temporary tables:

- Base definition is permanent and maintained in the data dictionary.
- Are materialized by the first SQL DML statement to access table.
- Space is charged against an allocation of temporary space.
- User can materialize up to 32 global tables per session.
- Tables can survive a system restart.
- Multiple concurrent users can reference the same global temporary table, but the contents are not shareable between users.

Partitioned Primary Index

Introduction

The Partitioned Primary Index (PPI) feature enables users to set up databases that provide performance benefits from data locality, while retaining the benefits of scalability inherent in the hash architecture of the Teradata database.

This is achieved by hashing rows to different virtual AMPs, as is done with a normal PI, but creating local partitions within each virtual AMP.

Normal PI access remains unchanged, but in the case of a range query, for example, each virtual AMP is able immediately to focus its search on specific partitions within its workspace.

Non-Partitioned Primary Index

A traditional non-partitioned PI allows the data rows of a table to be:

- Hash partitioned (that is, distributed) to the AMPs by the hash value of the primary index columns
- Ordered by the hash value of the primary index columns on each AMP

Partitioned Primary Index

A Partitioned Primary Index (PPI) allows the data rows of a table to be:

- Hash partitioned to the AMPs by the hash of the primary index columns
- Partitioned on some set of columns on each AMP
- Ordered by the hash of the primary index columns within that partition

PPI introduces syntax that you can use to create a table with a PPI and to support the index.

PPI Syntax

The syntax for creating a PPI is an extension to the syntax for specifying a PI. it supports altering a PPI along with changes, for example, to the output of various support statements. Two functions, RANGE_N and CASE_N, can be used to simplify the specification of a partitioning expression.

Note: If you specify compress values for a column in partitioning expression and the column is not part of the primary index, you receive error message 3632.

Hash Index for Primary Partitioned Index

To enhance performance further, Teradata V2R5.0 supports Hash Index for Partitioned Primary Index base tables. The syntax for creating a hash index remains the same.

You should specify, however, an ORDER BY HASH or ORDER BY VALUES when defining a hash index for a PPI base table.

Performance Impact

PPI improves performance as follows:

- Uses partition elimination to improve the efficiency of range searches when, for example, the searches are range partitioned
- Provides an access path to the rows in the base table while still providing efficient join strategies

Moreover, if the same partition is consistently targeted, the part of the table updated may be able to fit largely in cache, significantly boosting performance

Performance Considerations

Performance tests indicate that the use of PPI can cause dramatic performance improvements, both in queries and in table maintenance.

- While a table with a properly defined PPI will allow overall improvement in query performance, certain individual workloads involving the table may become slower, such as prime index selects where the partition column criteria is not provided in the WHERE clause.
- There are potential cost increases for certain operations such as empty table insert-selects.
- The partitioning environment must be carefully implemented in order to gain maximum benefit. Benefits that are the result of using PPI will vary based on:
  - The number of partitions defined
  - The number of partitions that can be eliminated given the query workloads, and
  - Whether or not the user follows an update strategy that takes advantage of partitioning.

Identity Column

Introduction

Identity Column is defined in the ANSI standards as a new column attribute option. When this attribute is associated with a column, it causes the system to generate a table-level unique number for the column for every inserted row. The insertion may either be a singleton insert, an INSERT SELECT, or a bulk insert through a utility such as TPump, BTEQ import.

Note: MultiLoad and FastLoad do not support Identity Column.

Advantages

The main advantage of an identity column is its ease of use in defining a unique row identity value. It can guarantee uniqueness of rows in a table when the column is defined as a GENERATED ALWAYS column with NO CYCLE allowed.

For some tables, it is hard to come up with a combination of columns that would make a row unique. If a composite index is undesirable, an identity column may be defined as the primary index.

An identity column is also suited for generating unique primary key values used as employee numbers, order numbers, item numbers, and the like. In this way, a user can get a uniqueness guarantee without the performance overhead of specifying a unique constraint.

Disadvantages

The main disadvantage of an identity column is that the generated values will have identity gaps whenever an insert into a table having an identity column is aborted or rows are deleted from the table.

Moreover, once a table with an identity column is populated, deleting all the rows in the table and reinserting new rows will not cause the numbering to restart from '1'. Numbering will continue from the table's last generated number. To restart numbering from '1', the table has to be dropped and re-created before reloading the rows. Identity columns should, therefore, not be used for applications that cannot tolerate gaps in the numbering. Identity gaps are more of an issue with applications using identity columns for auto-numbering employees, orders, and so on.

Performance Recommendation

Set Identity Column on a non-PI column to avoid performance degradation associated with setting identity column on the prime index column.

Collecting Statistics

What are Statistics?

Statistics are data demographics "hints" the Optimizes uses. Collecting statistics ensures that the Optimizer will have the most accurate information to create the best access and join plans.

Collecting Statistics and the Optimizer

Collecting statistics ensures that the Optimizer has the most accurate information to create the best access and join plans. Without collected statistics, the Optimizer assumes:

- Non-unique indexes are highly non-unique.
- Non-index columns are even more non-unique than non-unique indexes.

In table joins, statistics help Teradata determine the spool file size it should create to contain the result.

Statistics are especially informative if index values are distributed unevenly. For example, when a query uses conditionals based on non-unique index values, Teradata uses statistics to determine whether indexing or a full search of all table rows is more efficient. You can use the EXPLAIN modifier for information on the proposed processing methods before submitting a request.

If Teradata determines that indexing is the best method, it uses the statistics to determine whether spooling or building a bitmap would be the most efficient method of qualifying the data rows. As a general rule, Teradata uses bitmapping to apply complex conditional expressions based on NUSI values to very large tables.

The statistics are located in multiple system tables of the Data Dictionary.

Collecting Statistics: Recommendations

Statistics should be collected, at a minimum, for the following tables/columns:

- For small tables where the number of rows is less than 5 times the number of AMPs, collect statistics on the primary index.
- For small tables involved in a join, collect statistics on the primary index
- For large tables, collect statistics on all NUPIs.
- For all tables, collect statistics on columns in any WHERE clause.

For yielding best plans, consider collecting statistics on:

- All indexes
- High-access join columns

- Columns referenced in WHERE clauses if the columns contain very skewed, non-unique data.

Note: If you use join indexes, collect statistics separately on the columns of each join index subtable.

Once collected, statistics should be maintained to accurately reflect the data demographics of the underlying table. It is better to have no statistics than stale or otherwise incorrect statistics.

How the Optimizer Obtains Necessary Information

To create the AMP steps to carry out an SQL request in Teradata, the Optimizer needs to have the following information:

- Number of rows in the tables involved in the request
- One or both of the following:
  - Distribution of the specific data values for index columns
  - Non-index columns used in the request

The Optimizer can obtain this information from either of two sources, one inefficient and one much more efficient:

Source	Comments
Random AMP sample	If no statistics are available, the Optimizer uses random AMP samples for: • Table row counts • Distribution of index or column values If statistics are not available on secondary indexes, the Optimizer must make a rough estimate of the selectivity of indexed values, and the possible number of distinct values in any NUSIs. This may result in inefficient use of secondary indexes, especially in join processing. Random AMP samples are less detailed and less reliable (especially if the index is non-unique and the distribution of primary data rows is lumpy) than COLLECT STATISTICS. On tables with > 1000 rows per AMP, random AMP samples give reliable results. The Optimizer performs random AMP sampling on the fly during the parsing procedure.

Collected statistics	The Optimizer always uses available collected statistics because they are detailed and are more likely to be accurate than a random AMP sample.
	However, you must actively request collection by submitting a COLLECT STATISTICS statement. (This does not interfere with performance, since columns can be accessed for queries and statistics at the same time.)
	Note: You also can use COLLECT STATISTICS with the Teradata Manager. The syntax is:
	`Collect Statistics on TableName`
	`[Column column_name]`
	`[Index (column_name, column_name, ...)]`

Random AMP Sampling

The Optimizer will generally use Random AMP Sampling when no statistics are present. The exception is when estimating the number of rows that satisfy specific terms (conditions). In this case, the random AMP sample is of no use. The Optimizer looks for statistics on the column(s) involved. If they are present, the Optimizer uses them and is usually able to make a pretty accurate estimate. If statistics are not available, the Optimizer makes default estimates with varying levels of confidence.

Using Approximations

By default, the Optimizer, which makes decisions on how to access table data, uses approximations of the number of:

- Rows in each table (known as the cardinality of the table), and

- Unique values in indexes in making its decisions.

The Optimizer gets its approximation of the cardinality of a table by picking a random AMP and querying the AMP with respect to the number of rows in the table. The chosen AMP does not actually count all of the rows it has for the table, but generates an estimate based on the average row size and the number of sectors occupied by the table on that AMP. The Optimizer then multiplies that estimate by the number of AMPs in the system (making an allowance for uneven hash bucket distribution) to estimate the table cardinality.

The number of unique index values is similarly estimated. Given that most of the values involved in these estimates, other than the number of AMPs in the system, is an approximation, it is possible (although unusual) for the estimate to be significantly off. This can lead to poor choices of join plans, and associated increases in the response times of the queries involved.

Frequency Distribution of Data

Using the COLLECT STATISTICS option amasses statistics that include frequency distribution of user data.

Frequency distribution organizes the distinct values of an index or column into a group of intervals as follows (each interval represents approximately 1% of table rows):

Interval Level Statistics	Table Level Statistics
Number of distinct values in the interval	Minimum data value for the index or column
Number of rows in the interval	Maximum data value for the index or column
Maximum data value in the interval	Total number of distinct values for the index or column
Number of rows for specific high-bias values	Total number of rows in the table

COLLECT STATISTICS Guidelines

Follow the guidelines below to use COLLECT STATISTICS.

Task	Guideline
Collect UPI statistics.	You collect no other statistics on the table. The table is small (that is, 100 rows/AMP).
Collect NUPI statistics.	The NUPI is: • Fairly or highly unique, and • Used commonly in joins, or • Skewed

Task	Guideline
Collect NUSI statistics on all NUSIs.	The Optimizer can use the NUSI in range scans (BETWEEN... AND...). With statistics available, the system can decide to hash on the values in the range if demographics indicate it would be less costly than a full table scan.
Collect NUSI statistics on covering (ALL option) NUSIs	If a secondary index is defined with the intent of covering queries (the ALL option is specified), you should consider collecting statistics even if the indexed columns do not appear in WHERE conditions. Collecting statistics on a potentially covering NUSI provides the Optimizer with the total number of rows in the NUSI subtable and allows it to make better decisions regarding the cost savings from covering.
Collect NUSI statistics on NUSIs with ORDER BY	If a sort key is specified in a NUSI definition with the ORDER BY option, collected statistics on that column so the Optimizer can compare the cost of using a NUSI-based access path in conjunction with a range or equality condition on the sort key column.
Collect non-index column statistics.	Consider collecting statistics on non-indexed columns (single columns or multi-columns) that are fairly or highly unique and are used commonly in: • Equi-joins, especially with more than two tables. (If you have a multi-column group and can't afford to collect statistics on all columns, collect on the most unique column.) • Equi-compares. Collecting statistics on a group of columns allows the Optimizer to estimate the number of qualifying rows for queries that have search conditions on each of the columns or that have a join condition on each of the columns.
Refresh statistics after updates.	When: • The number of rows changed is greater than 10%. • The demographics of columns with collected statistics changes. • These statistics are not useful for the requests currently being entered (for example, queries based on unique PI values). • Users currently do not use the table. This allows the system to recover the disk space the table occupied in database DBC.

Statistics on Skewed Data

When you collect statistics for skewed data, the Optimizer can accommodate exceptional values.

Statistics reveal values that include the most non-unique value in the table and the most non-unique value per value range. The system divides the table into 100 groupings and maintains statistics on the most non-unique value/range.

Without collected statistics, the system derives row counts from a random AMP sample for:

- Small tables (less than 1000 rows per AMP),
- Unevenly distributed (skewed row distribution due to PI) tables

Note: Small tables often distribute unevenly. If rows in a table are not distributed evenly across all AMPs, random AMP samples may not represent the true total number of rows in the table.

Collecting Statistics for Join Index Columns

You should collect statistics separately for a base table column and its corresponding join index column. The statistics for base tables and join indexes are not interchangeable and the demographics for values in a base table may be very different from those for values in the join index. If statistics on a join index column are absent, the Optimizer does not try to derive them from the statistics of its underlying base tables.

In general, statistics for a join index should be collected on one or more of the following:

- *Always*, for *all* join indexes, the primary index column of the join index. This provides the Optimizer with baseline statistics, including the total number of rows in the join index.
- Columns used to define a secondary index upon the join index. These statistics help the Optimizer evaluate alternative access paths when scanning a join index.
- Search condition keys, which also assist the Optimizer in evaluating alternative access paths
- Columns used to join a join index with yet another table that is not part of the join index. These statistics assist the Optimizer in estimating cardinality.
- Also consider collecting statistics on other popular join index columns, such as one that frequently appears in WHERE conditions, especially if it is serving as the sort key for a value-ordered join index.

Statistics Collection on Sample Data

Without statistics, query performance can suffer because the Optimizer does not have the information it needs to choose access paths efficiently. However, collecting statistics can be very time consuming because the task performs a full

table scan and sorts the data to determine the number of occurrences of each distinct value.

Given this, you may choose to specify statistics collection on a sample of the data, instead of all the data. Collecting statistics on a sample significantly reduces the disk I/O required to read the data and the CPU time required to sort it.

You can specify optional USING SAMPLE keywords in either the:

- General COLLECT STATISTICS statement syntax
- Or, in the special syntax form modeled after CREATE INDEX

  When you use the keywords in the special syntax format, the syntax applies to each of the indexes listed in the COLLECT STATISTICS statement.

Sampled Statistics: Usage Considerations

Sampled statistics are generally appropriate for:

- Very large tables
- Uniformly distributed data
- Indexed or non-indexed column(s)

Sampled statistics, as specified by the USING SAMPLE option, should be considered when collecting statistics on very large tables and where resource consumption from the collection process is a performance concern.

Sampled statistics should not be used on small tables, nor should they be used as a wholesale replacement for existing collections. Rather, sampling should only be considered whenever the overhead from full scan statistics collection (most notably CPU costs) is of great concern to the customer.

Sampling may degrade the quality of the resulting statistics and the subsequent query plans chosen by the Optimizer. Thus, sampled statistics are generally more accurate for data that is uniformly distributed. For example, columns or indexes that are unique or nearly unique are uniformly distributed. Sampling should not be considered for data that is highly skewed because the Optimizer needs to be fully aware of such skew.

In addition to uniformly distributed data, sampling can be more accurate for indexes than non-indexed column(s). For indexes, the scanning techniques employed during sampling can take advantage of the hashed organization of the data to improve the accuracy of the resulting statistics.

When sampling, you need not specify the percentage of rows of the table to sample. By default, Teradata will begin with a 2% sample. If, after generating statistics for that sample, Teradata detects any skewing, it will increase the sample bit by bit to as high as 50% until it determines that the sample percent is in line with the observed skewing. Thus, sample statistics will generally require between 2 and 50% of the resources necessary to generate full statistics.

Stale Statistics

Statistics provide more detailed information, and include an exact row count as of the time that the statistics were gathered. However, if the statistics are "stale," that is, if the table's characteristics (distribution of data values for a column or index for which statistics have been collected, number of rows in the table, and so on) have changed significantly since the statistics were last gathered, the Optimizer can be misled into making poor join plans, with associated poor performance of queries which use the stale statistics.

Example

If for table A, statistics were gathered when table had 1,000 rows, but now the table has 1,000,000 rows (perhaps statistics were gathered during the prototyping phase), and if for table B, no statistics were gathered, but now the table has 75,000 rows, then if a product join between table A and table B is necessary for a given query, and one of the tables must be duplicated on all AMPs, then the Optimizer will choose table A to be duplicated, since 1,000 rows (from the stale statistics) is much less than 75,000 rows.

Since in reality Table A now has 1,000,000 rows, the Optimizer will be making a very poor decision (duplicating 1,000,000 rows instead of 75,000), and the query will run much longer than necessary.

When are Statistics Stale?

There are two general circumstances under which statistics can be considered to be stale:

1. Number of rows in the table has changed significantly.

   The number of unique values for each statistic on a table, as well as the date and time the statistics were last gathered, can be obtained by:

   ```
   HELP STATISTICS tablename;
   ```

   For statistics on unique indexes, this can be cross-checked by comparing the row count returned by:

   ```
   SELECT COUNT(*) FROM tablename;
   ```

   For statistics on non-unique columns, the HELP STATISTICS result can be cross-checked by comparing the count returned by:

   ```
   SELECT COUNT(DISTINCT columnname) FROM tablename;
   ```

2. The range of values for a index or column of a table for which statistics have been collected has changed significantly.

   Sometimes one can infer this from the date and time the statistics were last collected, or by the very nature of the column. For example, if the column in question holds a transaction date, and statistics on that column were last gathered a year ago, it is almost certain that the statistics for that column are stale.

Refreshing Stale Statistics: Recommendations

It is recommended that statistics be recollected if there is as little as a 10% change (rows added or deleted) in a table.

For high volumes of very non-unique values such as dates or timestamps, it may be advantageous to recollect at 7%.

How to Refresh Stale Statistics

If the statistics for a table may be stale, they can be easily recollected. The following statement:

```
COLLECT STATISTICS ON tablename;
```

will recollect statistics on all indexes and columns for which previous COLLECT STATISTICS statements were done (and for which DROP STATISTICS statements have not been done).

Because collecting statistics involves a full-table scan, it may take a significant amount of time. It should, therefore, be done off-hours for large tables.

It may be revealing to execute the Help Statistics statement before and after recollecting statistics, to see what, if any, difference the recollect makes.

Moreover, for frequently executed queries, requesting an EXPLAIN before and after recollecting statistics may show differences in join plans and/or spool row count/processing time estimates.

Referential Integrity

Introduction

Referential Integrity (RI) refers to relationships between tables based on the definition of a primary key and a foreign key.

For more information on RI, see:

- *Introduction to Teradata RDBMS*
- *Teradata RDBMS SQL Reference*
- *Teradata RDBMS Database Design*

Benefits of RI

The following table lists and describes the benefits of RI:

Benefit	Description
Maintains data consistency	The system enforces relationships between tables. For example, Teradata enforces the relationship between a Customer ID and an application based on the definition of a primary key and a foreign key.
Maintains data integrity	When performing INSERT, UPDATE, and DELETE operations, Teradata maintains data integrity among referencing and referenced tables.
Increases development productivity	It is not necessary to code SQL statements to enforce referential constraints. Teradata automatically enforces RI.
Requires fewer programs to be written	Teradata ensures that update activities do not violate referential constraints. Teradata enforces RI in all environments; you need no additional programs.
Improves query performance	Teradata chooses the most efficient method to enforce the referential constraints.

Overhead Cost of RI

Overhead cost includes the building of reference index subtables and inserting, updating and deleting rows in the referencing and referenced tables. Overhead for inserting, updating and deleting rows in the referencing table is similar to that of USI subtable row handling.

The system redistributes a row for each reference to the AMP containing the subtable entry (USI or reference index). Specific processing differs thereafter; most of the cost is in message handling.

When implementing tables with RI:

- Consider the performance impact to update operations first.
- INSERTs will slow performance if RI is in the tables. Performance will be even slower if application code is in the tables.
- Consider the cost of extra disk space for tables and extra cost for maintenance.
- Consider the cost of extra disk space for reference index subtables versus savings on program maintenance and increased database reliability.
- Compared with costs elsewhere (for example, secondary index), consider the cost of checking in the application especially via DML versus cost of not checking at all.

The following table describes the RI overhead for various operations:

Operation	Description

Building the reference index subtable	This is similar to executing the following statement:
	`SELECT I.Reference_Field, COUNT (*)`
	`FROM Referencing_table I,  Referenced_table E`
	`WHERE I.Reference_Field = E.Reference_Field`
	`GROUP BY I.Reference_Field;`

Inserting a row into a referencing table	Teradata makes an RI check against the reference index subtable. If the referenced field is in the reference index subtable, Teradata increments the count in the reference index subtable. If the referenced field is not in the reference index subtable, Teradata checks the referenced table to verify that the referenced field exists. If it does, Teradata adds an entry with a count of 1 to the reference index subtable.
Deleting a row from the referencing table	Teradata makes an RI check against the reference index subtable, and decrements the count in the reference index subtable for the referenced field. If the count becomes zero, Teradata deletes the subtable entry for the referenced field.
Updating a referencing field in the referencing table	Teradata makes an RI check against the reference index subtable, and executes both the inserting-a-row and deleting-a-row operations on the reference index subtable, decrementing the count of the old referenced field value and incrementing the count of the new reference field value. This is similar to changing the value of a USI column.
Deleting a row from the referenced table	Teradata checks the reference index subtable to verify that the corresponding referenced field does not exist. Assuming it does not exist, Teradata can delete the row from the referenced table. The reference index subtable check does not require the system to pass a message to another AMP, since the referenced field is the same value in the referenced table and the reference index subtable.

Join Elimination

This feature eliminates redundant joins that are based on information from RI.

The following conditions eliminate RI:

- RI exists between the two tables.
- Query conditions are conjunctive.
- The query does not contain reference columns from the primary key table, other than the primary key columns, including the SELECT, WHERE, GROUP BY, HAVING, ORDER BY, and so forth.
- Primary key columns in the WHERE clause appear only in primary key-foreign key joins

IF...	THEN...
the preceding conditions are met	the primary key and primary key join are removed from the query.
	all references to the primary key columns in the query are mapped to the corresponding foreign key columns.
foreign key columns are nullable	the “NOT NULL” condition is added.

Soft RI

To maximize the usefulness of join elimination, you can specify RI constraints that the Teradata RDBMS does not enforce. You must guarantee that this constraint is valid for tables. The Optimizer can use the constraints without incurring the penalty of database-enforced RI.

Current syntax for CREATE TABLE and ALTER TABLE statements allow you to ADD and DROP both column- and table-level constraints for enforcing RI. You can use the WITH NO CHECK OPTION clause to specify statements with soft RI.

When you use the WITH NO CHECK OPTION clause, the RDBMS does not enforce the RI constraint. This implies that a row having a non-null value for a referencing column can exist in a table even if an equal value does not exist in a referenced column. Error messages, that would otherwise be provided when RI constraints are violated, do not appear when you specify soft RI.

Note: Soft RI relies heavily upon your knowledge of the data. If the data does not actually satisfy the soft RI constraint the you provide and the Optimizer relies on the soft RI constraint, then queries can produce incorrect results.

Batch RI

When using traditional RI and soft RI you should take into consideration that:

- Traditional RI is expensive to maintain because checking for consistency is performed row-by-row. The RDBMS accepts the consistent rows and rejects the inconsistent ones.
- Soft RI relies heavily on your knowledge of the data. If the data is not consistent, then queries can produce incorrect results, or data can be corrupted.

To address these considerations, Teradata offers Batch RI that the RDBMS enforces through an "all" or "nothing" approach. That is, the RDBMS checks the validity of the RI when you create it and accepts the RI only if the whole data in the parent and child tables is consistent; otherwise, the RDBMS rejects the RI. The RDBMS applies the same checking to DELETEs, INSERTs, and UPDATEs.

You can use the WITH CHECK OPTION clause to specify batch RI in CREATE TABLE and ALTER TABLE statements.

Batch RI is:

- Easy to create and maintain because you can join the parent and child rows to check for consistency
- Appropriate to workloads when you are confident about the data consistency but still would like the RDBMS to double check

2PC Protocol

Introduction

Two-Phase Commit (2PC) is a protocol for committing update transactions processed by multiple systems that do not share the same locking and recovery mechanism.

For detailed information on 2PC, see *Teradata RDBMS Database Design*.

Performance Impact

Consider the following disadvantages of using the 2PC protocol:

- Performance may decrease because, at the point of synchronization, up to two additional messages are exchanged between the coordinator and participant, in addition to the normal messages that update the databases.

  If your original Teradata SQL request took longer to complete than your other requests, the performance impact due to the 2PC overhead will be less noticeable.
- If the Teradata server restarts, and a session using the 2PC protocol ends up in an IN-DOUBT state, Teradata holds data locks indefinitely until you resolve the IN-DOUBT session. During this time, other work could be blocked if it accesses the same data for which Teradata holds those locks. To resolve this situation, perform the following steps:

Step	Action
1	Use the COMMIT/ROLLBACK command to manually resolve the IN-DOUBT sessions.
2	Use the RELEASE LOCKS command.
3	Use the RESTART command to restart your system.

2PC causes no system overhead when it is disabled.

Updatable Cursors

Introduction

In ANSI mode, you can define a cursor for the query results and for every row in the query results in order to update or delete the data row via the cursor associated with the row.

This means that update and delete operations do not identify a search condition; instead, they identify a cursor (or a pointer) to a specific row to be updated or deleted.

Updatable cursors allows you to update each row of a select result independently as it is processed.

Recommendations

To reap the full benefit from the Updatable Cursor feature, you should minimize:

- The size of query result and number of updates/transaction
- The length of time you hold the cursor open

Using many updates per cursor may not be optimal because:

- It blocks other transactions.
- The system requires longer rollbacks.

In this case, it is better to use the MultiLoad utility to do updates.

Sparse Indexes

Introduction

Using sparse indexes, you can index a portion of the table using WHERE clause predicates to limit the rows indexed. This capability is implemented using join index technology.

Allowing constant expressions in the WHERE clause of the CREATE JOIN INDEX statement gives you the ability to limit the rows that are included in the join index to a subset of the rows in the table based on an SQL query result. This capability in effect allows you to create *sparse* indexes.

When base tables are large, you can use this feature to reduce the content of the join index to only the portion of the table that is frequently used if the typical query only references a portion of the rows.

Performance Impact

A sparse index can focus on the portions of the tables that are most frequently used. This capability:

- Reduces the storage requirements for a join index.
- Makes the costs for maintaining an index proportional to the percent of rows actually referenced in the index.
- May make query access faster because the join index is smaller.

EXPLAIN Feature and the Optimizer

Introduction

EXPLAIN is one of the most valuable tools in Teradata for understanding how the Optimizer works. When you type EXPLAIN in front of any SQL request, the Optimizer returns a description of how the request is broken into AMP steps for processing.

EXPLAIN quickly highlights missing statistics, unused indexes, and so on. By utilizing the information available from EXPLAIN, you may be able to prevent some performance problems from occurring.

Use EXPLAIN to…	Comments

identify secondary indexes by index number.	The EXPLAIN text identifies secondary indexes used for joins and accesses by internal Teradata database index number, as well as by column name. To obtain a list of secondary indexes by internal number for each of your tables, run the SELECT statement. For example, you might enter the following SELECT statement for a list of secondary indexes on the Message table in the RST database.
	`SelectColumnName`
	`,IndexNumber`
	`FromDBC.Indices`
	`WhereDatabaseName = 'RST'`
	`AndTableName = 'Message'`
	`Order ByIndexNumber;`

check if AMP plans for your joins are what you expected.	It is crucial that you develop the skill of following row counts through an EXPLAIN, to identify where the Optimizer has made an assumption different than the user.
evaluate the use of common and parallel steps.	EXPLAIN identifies serial, parallel, and common steps, including: • Any indexes that the Optimizer will use to select rows • The sequence of intermediate spool files that might be generated for joins The Optimizer creates common steps when different SQL statements need the same steps. EXPLAIN does not note common steps; it is up to you to recognize that a spool is being reused.

Use EXPLAIN to...	Comments
uncover hidden or nested views.	EXPLAIN displays the resolution of views down to the base tables so that you can identify obscure or nested views.
detect data movement.	See "Example: Using EXPLAIN to Detect Data Movement" on page 12-70.

The following may change and affect EXPLAIN output:

- Data volatility
- Distribution
- Software release level
- Secondary indexes, space requirements and maintenance
- Design (see "Revisiting Database Design" on page 1-14)

Keep a file of EXPLAIN output over time to identify processing changes and revisit index selection accordingly. Note that in Teradata V2R5.0, EXPLAIN now indicates conditional statements.

For more information on EXPLAIN, see:

- *Introduction to Teradata RDMBS*
- *Teradata RDBMS SQL Reference*
- *Teradata RDBMS Database Design*

Example: Using EXPLAIN to Detect Data Movement

Data movement includes duplication and redistribution in join plans.

In the following examples:

Table(s)	Comments
facttable	has columns c1, c2, c3, c4, c5, c6, c7, and c8 with PI c1, c2, c3, c7 and secondary index c1.
dimension1	has columns c1 and c4 with PI c1.
dimension2	has columns c2 and c5 with PI c2.
facttable, dimension1, and dimension2	belong to database jch_star.

Example 1

To take a look at data movement of a join with duplication, you enter an EXPLAIN statement similar to the following:

```
EXPLAIN
SELECT a.c1, a.c4, a.c7 FROM facttable a
```

```
INNER JOIN dimension1 b ON a.c1 = b.c1
INNER JOIN dimension2 c ON a.c2 = c.c2;
```

Example 1: Explain Output

The output would be similar to the following:

```
Explanation

-----------------------------------------------------------
```

1 First, we lock a distinct JCH_STAR."pseudo table" for read on a RowHash to prevent global deadlock for JCH_STAR.c.

2 Next, we lock a distinct JCH_STAR."pseudo table" for read on a RowHash to prevent global deadlock for JCH_STAR.b.

3 We lock a distinct JCH_STAR."pseudo table" for read on a RowHash to prevent global deadlock for JCH_STAR.a.

4 We lock JCH_STAR.c for read, we lock JCH_STAR.b for read, and we lock JCH_STAR.a for read.

5 We do an all-AMPs RETRIEVE step from JCH_STAR.b by way of an all-rows scan with no residual conditions into Spool 2, which is duplicated on all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated to be 768 rows. The estimated time for this step is 0.04 seconds.

6 We execute the following steps in parallel.

   1 We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to JCH_STAR.a by way of a traversal of index # 4 extracting row ids only. Spool 2 and JCH_STAR.a are joined using a nested join with a join condition of ("JCH_STAR.a.c1 = Spool_2.c1"). The input table JCH_STAR.a will not be cached in memory. The result goes into Spool 3, which is built locally on the AMPs. Then we do a SORT to order Spool 3 by field Id 1. The result spool file will not be cached in memory. The size of Spool 3 is estimated to be 2,574,931 rows. The estimated time for this step is 5 minutes and 34 seconds.

   2 We do an all-AMPs RETRIEVE step from JCH_STAR.c by way of an all- rows scan with no residual conditions into Spool 4, which is duplicated on all AMPs. The size of Spool 4 is estimated to be 768 rows. The estimated time for this step is 0.04 seconds.

7 We do an all-AMPs JOIN step from Spool 3 (Last Use) by way of an all- rows scan, which is joined to JCH_STAR.a. Spool 3 and JCH_STAR.a are joined using a row id join, with a join condition of ("JCH_STAR.a.c1 = Spool_3.c1"). The input table JCH_STAR.a will not be cached in memory. The result goes into

```
     Spool 5, which is built locally on the AMPs. The size of Spool
     5 is estimated to be 2,574,931 rows. The estimated time for
     this step is 7 minutes and 45 seconds.

8    We do an all-AMPs JOIN step from Spool 4 (Last Use) by way of
     an all-rows scan, which is joined to Spool 5 (Last Use). Spool
     4 and Spool 5 are joined using a single partition hash join,
     with a join condition of ("Spool_5.c2 = Spool_4.c2"). The
     result goes into Spool 1, which is built locally on the AMPs.
     The size of Spool 1 is estimated to be 2,574,931 rows. The
     estimated time for this step is 44.37 seconds.

9    Finally, we send out an END TRANSACTION step to all AMPs
     involved in processing the request.

10   The contents of Spool 1 are sent back to the user as the
     result of statement 1. The total estimated time is 0 hours and
     14 minutes and 4 seconds.
```

Example 2

To take a look at data movement of a join with redistribution, you might enter an EXPLAIN statement similar to the following:

```
EXPLAIN
SELECT a.c1, a.c4, a.c7 FROM facttable a
INNER JOIN dimension1 b ON a.c4 = b.c4
INNER JOIN dimension2 c ON a.c5 = c.c5;
```

Example 2: Explain Output

The output would be similar to the following:

```
Explanation

-----------------------------------------------------------

1    First, we lock a distinct JCH_STAR."pseudo table" for read on
     a RowHash to prevent global deadlock for JCH_STAR.c.

2    Next, we lock a distinct JCH_STAR."pseudo table" for read on a
     RowHash to prevent global deadlock for JCH_STAR.b.

3    We lock a distinct JCH_STAR."pseudo table" for read on a
     RowHash to prevent global deadlock for JCH_STAR.a.

4    We lock JCH_STAR.c for read, we lock JCH_STAR.b for read, and
     we lock JCH_STAR.a for read.

5    We execute the following steps in parallel.
```

1 We do an all-AMPs RETRIEVE step from JCH_STAR.c by way of an all-rows scan with no residual conditions into Spool 2, which is duplicated on all AMPs. The size of Spool 2 is estimated to be 768 rows. The estimated time for this step is 0.04 seconds.

2 We do an all-AMPs RETRIEVE step from JCH_STAR.a by way of an all-rows scan with no residual conditions into Spool 3, which is built locally on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The result spool file will not be cached in memory. The size of Spool 3 is estimated to be 7,966,192 rows. The estimated time for this step is 9 minutes and 44 seconds.

3 We do an all-AMPs RETRIEVE step from JCH_STAR.b by way of an all-rows scan with no residual conditions into Spool 4, which is redistributed by hash code to all AMPs. The size of Spool 4 is estimated to be 96 rows. The estimated time for this step is 0.04 seconds.

6 We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to Spool 3 (Last Use). Spool 2 and Spool 3 are joined using a single partition hash join, with a join condition of ("Spool_3.c5 = Spool_2.c5"). The result goes into Spool 5, which is redistributed by hash code to all AMPs. The size of Spool 5 is estimated to be 95,797 rows. The estimated time for this step is 5.34 seconds.

7 We do an all-AMPs JOIN step from Spool 4 (Last Use) by way of an all-rows scan, which is joined to Spool 5 (Last Use). Spool 4 and Spool 5 are joined using a single partition hash join, with a join condition of ("Spool_5.c4 = Spool_4.c4"). The result goes into Spool 1, which is built locally on the AMPs. The size of Spool 1 is estimated to be 10,505 rows. The estimated time for this step is 0.60 seconds.

8 Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

9 The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0 hours and 9 minutes and 50 seconds.

Chapter 13:

Data Management

This chapter discusses the impact of data management on performance.

Topics include:

- Data distribution issues
- Identifying uneven data distribution
- Parallel efficiency
- Primary Index and row distribution
- Recommended housekeeping
- Data protection options
- PDE-Dump and Disk Array Plus
- Determining table and database size growth rates

Data Distribution Issues

The following table lists data distribution issues, causes, and results.

Issue	Cause	Results
Same row hash value for an excessive number of rows • Rows with same row hash value cannot fit in a single data block. • Rows spill over into additional data blocks.	• Highly non-unique PIs. As an estimate, more than 1000 occurrences/NUPI value begin to cause performance degradation problems. This figure is based on all the rows for the same NUPI value spilling over into more than five data blocks. • Size of the data block.	• Increased Inputs/Outputs (I/Os) for updates. • Increased compares for inserts and FastLoad (more Central Processing Unit (CPU) and I/Os). • Performance degradation in the Restore and Table Rebuild utilities.
Some AMPs have many more rows of a table than do other AMPs.	One or a few NUPI values have many more rows than all the other NUPI values.	• Poor CPU parallel efficiency on the AMPs during full-table scans and bulk inserts. • Maintenance on the lumps involves increased I/Os for updates, and increased compares for inserts (more I/Os).

Identifying Uneven Data Distribution

Using SQL

You can use an SQL statement similar to the following to determine if data for a given table is evenly distributed across all AMP vprocs. The SQL statement displays the AMP with the most-used and the AMP with the least-used space, investigating data distribution in the Message table in database RST.

```
SELECT vproc,CurrentPerm
FROM DBC.TableSize
WHERE Databasename = 'RST'
AND Tablename = 'Message'
ORDER BY 2desc;
```

Using Space Usage Application

If Teradata Manager is installed, you can examine uneven data distribution via the Space Usage application. Space Usage presents detailed reports that show disk space utilization from several perspectives.

Although even data distribution has many advantages in a parallel system, at times you must sacrifice level data distribution to reap the other benefits of PIs, specifically of a PI in join operations.

Using Hash Functions

The following functions can be used to identify uneven data distribution:

Function	Definition
HASHAMP	AMP that owns the hash bucket
HASHBACKAMP	Fallback AMP that owns the hash bucket
HASHBUCKET	Grouping for the specific hash value
HASHROW	32 bits of row hash ID without the uniqueness field

HASHAMP Example

If you suspect distribution problems (skewing) among AMPS, the following is a sample of what you might enter for a three-column table:

```
SELECT HASHAMP (HASHBUCKET (HASHROW (col_x, col_y,
col_z))), count (*)
FROM hash15
GROUP BY 1
ORDER BY 2 desc;
```

HASHROW Example

If you suspect collisions in a row hash, the following is a sample of what you might enter for a three-column table:

```
SELECT HASHROW (col_x, col_y, col_z), count (*)
FROM hash15
GROUP BY 1
ORDER BY 1
HAVING count(*) > 10;
```

Impact of Uneven Data Distribution

Uneven data distribution results in:

- Poor CPU parallel efficiency on full table scans and bulk inserts
- Increased I/Os for updates and inserts of over-represented values

If you suspect uneven data distribution:

- Run the ResVproc macro to check the AMP parallel efficiency so that you can identify the AMP affecting that node.
- Check table distribution information at the AMP level by running an SQL query against the Data Dictionary/Directory view, DBC.TableSize (see page 13-3 for a sample query). This query identifies the number of bytes/AMP vproc for a given table.

Check Periodically for Skewed Data

The parallel architecture of the Teradata database will distribute rows across the AMPs using hash buckets. Each AMP has its own set of unique hash buckets. When data is said to be skewed or lumpy, it means that many rows are hashed to the same AMP because of their highly non-unique index(es), compared to the other AMPs on the system, and therefore the distribution is very uneven.

Although this doesn't always cause problems, nor can it always be avoided, it may result in hot AMPs during access, and on a very busy system, can use a lot of resources.

Sample Scripts

Below is a set of useful scripts written to check for skewed data:

Note: Running a script that checks for skewed data is a good performance practice when new applications are being loaded on the system or when there are lots of data changes.

```
/* */
/* LUMPY - identified those tables that are not evenly distributed
*/
/* Variance should ideally be less than 5%. Here we have */
/* it set to 1000% which will usually indicate that some */
/* or many vprocs do not have any data from the table at */
/* all. You can use "RETLIMIT" to limit the number of */
/* rows returned. */

SEL (MAX(CurrentPerm) - MIN(CurrentPerm)) * 100
/(NULLIF(MIN(currentperm),0))
(NAMED variance)
(FORMAT 'zzzzz9.99%')
,MAX(CurrentPerm)
(TITLE 'Max')
(FORMAT 'zzz,zzz,zzz,999')
,MIN(currentperm)
(TITLE 'Min')
(FORMAT 'zzz,zzz,zzz,999')
,TRIM(DatabaseName)
||'.'
||TableName
(NAMED Tables)
FROM DBC.tablesize
GROUP BY DatabaseName, TableName
HAVING SUM(CurrentPerm)
1000000
AND variance
1000
WHERE DatabaseName NOT IN('CrashDumps','DBC')
ORDER BY Tables;
/* */
/* Once you have identified a target table, you can display */
/* the detailed distribution with the following query. */
/* */
SELECT vproc, CurrentPerm FROM dbc.TableSize
WHERE DatabaseName = 'xxxx'
AND TableName = 'yyyy'
ORDER BY 1;
/* */
/* The following table will list the row distribution by amp */
/* for a given table. */
/* */
sel dt1.a (title'AMP')
,dt1.b (title'Rows')
```

```
,((dt1.b/dt2.x (float)) - 1.0) * 100
(format'+++9%',title'Deviation')
from
(sel hashamp(hashbucket(hashrow(<index>)))
,count(*)
from <databasename>.<tablename>
group by 1
)dt1 (a,b)
,(sel (count(*) / (hashamp()+1)(float))
FROM <databasename>.<tablename>
)dt2(x)
order by 2 desc,1;
/* */
/* The following query will provide the distribution by amp */
/* for a given index or column. */
/* */
sel hashamp(hashbucket(hashrow(index or column)))
,count(*)
from database.table
group by 1
order by 2 desc;
/* */
/* The following query will provide the number of collisions */
/* for row hash. */
/* */
sel hashrow(index or column), count(*)
from database.table
group by 1
order by 1
having count(*) > 10;
/* */
/* The following query will provide the number of amps and */
/* the number of rows impacted by a query. */
/* */
LOCKING TABLE <table> FOR ACCESS
SELECT COUNT(DT.ampNum) (TITLE '#AMPS')
,SUM(DT.numRows) (TITLE '#ROWS')
FROM
(SELECT HASHAMP(HASHBUCKET(HASHROW(<index>))),count(*)
FROM <table>
WHERE <selection criteria>
GROUP BY 1)DT (ampNum, numRows);
```

Parallel Efficiency

Introduction

If you have poor parallel efficiency and your data is not skewed, you should look for signs of skewing during processing.

Join Processing and Aggregation

Join processing and aggregation both may involve row redistribution. An easy way to find out if rows are redistributed during an operation is to check for high BYNET read and write activity.

In join processing, poor parallel efficiency occurs when the join field is highly skewed. Since rows are redistributed to AMPs based on the hash value of the join column, a disproportionate number of rows may end up on one AMP or on a few AMPs.

For example, you perform a join on city code with a large number of instances of New York and Los Angeles. A large number of rows would be redistributed to two AMPs for the join. Those AMPs would show much higher CPU utilization during the operation than the other AMPs.

Referential Integrity

Note that skewed processing can also occur with RI when the referencing column has skewed demographics, for example, when the referenced column is city code.

Performance Impact

If your parallel efficiency is poor and data distribution is not skewed, look for skewed processing. Both skewed data distribution and skewed processing can adversely affect node parallel efficiency as well as AMP parallel efficiency, because the CPU activity of a node is a direct reflection of the CPU activity of vprocs.

Primary Index and Row Distribution

Introduction

The hash value of the PI controls row distribution. In a normal environment, hash values are evenly distributed across the nodes and across the AMPs within a node.

The less unique the values for the index, the less evenly the rows of that table are distributed across the AMPs. If a table has a NUPI with thousands of instances of a single value, the table can become skewed.

Effects of Skewed Data

The effects of a skewed table appear in several types of operations. For example:

- In full table scans, the AMPs with fewer rows of the target table must wait for the AMPs with disproportionately high numbers of rows to finish. Node CPU utilization reflects these differences because a node is only as fast as the slowest AMP of that node.
- In the case of bulk inserts to a skewed table, consider the extra burden placed on an AMP with a high number of multiple rows for the same NUPI value.

  For example, assume you have a 5 million row table, with 5,000 rows having the same NUPI value. You are inserting 100,000 rows into that table, with 100 of those insert rows having the same NUPI value. The AMP holding the 5,000 rows with that NUPI value has to perform one half million duplicate row checks (5,000 * 100) for this NUPI. This operation results in poor parallel efficiency.

Performance Impact: Primary Index

Keep in mind the following:

- The more unique the PI, the more unique the row hash value.
- The more unique the row hash value, the more even the data distribution across all the AMP vprocs. Even data distribution enhances parallel efficiency on full-table scan operations.
- Unique Primary Indexes (UPIs) generate the most even distribution of table rows across all AMP vprocs.
- Non-Unique Primary Indexes (NUPIs) generate even row distribution to the same degree that values for a column or columns are unique. Rows with the same row hash always go to the same AMP, whether they are from the same table or from different tables.

If a PI causes uneven data distribution, you may decide to change it. To determine the best PI for a table, factor in the:

- Extent of update activity
- Number of full table scans
- Join activity against the PI definition
- Frequency of PI as a selectivity column and, therefore, a potential access path

Recommended Housekeeping

Introduction

The following section covers tables and logs that, if left unpurged, can grow considerably and impact both performance and space.

DBC.Accessrights

When a user/database is created there is a default number of rights added to dbc.accessrights. These should be reviewed and deleted as necessary. Whenever a logon is made to the database a full table scan is done against this table so the smaller it can be made, the better.

Sample Scripts

The following SQL scripts will check for inconsistencies in table data dictionary table, DBC.Accessrights.

These inconsistencies may have been introduced as a result of historical problems that caused invalid access rows to be inserted or failed to remove obsolete access rows. These problems have been corrected in the database code but the erroneous rows may still exist in the dictionary and should be removed if they exist.

```
/* ----------------------------------------------------------------
-----------------------------------*/
/* Check for "orphan" accessright entries, i.e. accessrights that
reference */
/* non-existent tables. Orphan access rights will not cause a
problem unless you */
/* get so many that they impact the access performance. */
/* ----------------------------------------------------------------
-----------------------------------*/
SELECT count(*) FROM DBC.AccessRights
WHERE TVMID <> '000001000000'xb
AND TVMID NOT IN (SEL TVMId FROM DBC.TVM);
/* ----------------------------------------------------------------
-----------------------------------*/
/* Check for accessrights that reference non-existent databases.
*/
/* ----------------------------------------------------------------
-----------------------------------*/
SELECT count(*) FROM DBC.AccessRights
WHERE DatabaseId NOT IN (SEL DatabaseId from DBC.Dbase);
/* ----------------------------------------------------------------
-----------------------------------*/
/* Check for access rights granted to non-existent users. */
/* ----------------------------------------------------------------
-----------------------------------*/
```

```
SELECT count(*) FROM DBC.AccessRights
WHERE UserID NOT IN (SEL DatabaseID FROM DBC.Dbase);
/* ------------------------------------------------------------
-----------------------------------*/
/* Check for accessrights that extend Reference or Index access
at the database */
/* level (this is only valid at the table level). Reference DR
43044 and */
/* incident REC9K7TQR for further details. If these exist it is a
security hole. */
/* All must be removed or any subordinate databases will inherit
the problem. */
/* ------------------------------------------------------------
-----------------------------------*/
SELECT count(*) FROM DBC.AccessRights
WHERE AccessRight IN ('IX','RF')
AND Databaseid <> '00'xb
AND TVMId = '000000000000'xb;
/* ------------------------------------------------------------
-----------------------------------*/
/* Check for accessrights entries that are not properly
manipulated by access */
/* logging (see DR 48757) */
/* ------------------------------------------------------------
-----------------------------------*/
SELECT count(*) FROM DBC.AccessRights WHERE AccessRight = 'RO'
and userid = '00000100'xb AND databaseid = '00000100'xb;
If there are any rows revealed by these queries, contact the GSC.
The rows can be
manually remove during a maintenance window.
```

Logging Tables to be Archived/Deleted

You can archive the following logging tables and purge information 60-90 days old. Retention depends on shop requirements.

Note: You should never DROP these tables, DELETE the entries:

- Table: DBC.AccLogTbl
  Purpose: Logged User-Object events
- Table: DBC.SW_Event_Log
  Purpose: Database Console Log
- Table: DBC.EventLog
  Purpose: Session logon/logoff history
- Table: DBC.ResUsage
  Purpose: Resource monitor log table

Also review all other ResUsage tables as well as how much, and how often, data is collected.

Accounting Tables

- DBC.ACCTG

  This table holds ResUsage data by Acct/User.

  To reset the values, execute the following command:

  ```
  UPDATE DBC.AMPUsage
  SET CPUTime = 0, DiskIO = 0 ALL;
  ```

- DBC.DatabaseSpace

  This table contains database and tablespace accounting information.

  To reset the values, execute the following command:

  ```
  EXEC DBC.ClearPeakDisk;
  ```

Log All Environmental Changes

Since many data warehouses start out as user-owned decision support systems and later grow into operationalized, active data warehouses with applications that have become mission critical to the business need to be managed rigorously.

We thus recommend that you keep a record of:

- New or changed applications
- Changes to software
- Changes to hardware
- Changes to system configuration files or components
- Network changes
- Major data increases
- Event logging table events (restarts, errors, and so on)
- Database changes (new/deleted tables, indexes, etc.) If this is too much of a burden due to size of the databases, it may be time to consider using Teradata Metadata.
- New users, additional sessions/workload

Data Protection Options

Teradata offers options that protect both the availability and the integrity of your data. Among these options are:

- Fallback
- Redundant disk arrays
- PJ

This section discusses the performance considerations of each.

Fallback Option

Fallback data is used to process a request if the primary data becomes unavailable. The fallback option can be defined to create a second copy of a:

- Primary data table
- Secondary index subtable
- Join index of any type, including aggregate join index

Fallback protects the accessibility and integrity of your data if:

- Base data becomes unavailable due to a software error
- An AMP is lost for any reason

Fallback data is not a mirror image of base data. Rather, fallback rows are hashed distributed to ensure they do not reside on the same disks as, or replicate the location of, their base rows. Fallback thus implies a trade-off, in that it somewhat reduces resources and performance for the sake of data availability.

Fallback with Primary Index Operations

The following table illustrates data availability with and without fallback protection for PI operations:

Operation	No Fallback		Fallback		
	No Data Corruption on Primary AMP	Data Corruption on Primary AMP	No Data Corruption on Primary AMP/No Data Corruption on Fallback AMP	Data Corruption on Primary AMP/No Data Corruption on Fallback AMP	No Data Corruption on Primary AMP/no Data Corruption on Fallback AMP
Data retrieval	Succeeds	Fails	Succeeds; the system uses primary copy.	Succeeds; the system uses the fallback copy.	Succeeds; the system uses the primary copy.
Data modification	Succeeds	Fails	Succeeds; the system immediately modifies primary and fallback copies.	Succeeds; The system uses the fallback copy and modifies the primary copy later.	Succeeds; the system uses primary copy and modifies the fallback copy.

Fallback with All-AMP Operations

The following table illustrates data availability with and without fallback protection for all-AMP operations:

Operation	No Fallback		Fallback	
	No Data Corruption on All AMPs	Data Corruption on Some AMPs (Not in Same Cluster)	No Data Corruption on All AMPs	No Data Corruption on Some AMPs (Not in Same Cluster)
Data retrieval	Succeeds	Fails	Succeeds; the system uses the primary copy.	Succeeds; the system uses the available portion of the primary copy and the appropriate portion of fallback copy.
Data modification or data definition	Succeeds	Fails	Succeeds; the system immediately modifies primary and fallback copies.	Succeeds; the system immediately modifies the available portions of primary and fallback copies. The system modifies unavailable portions later.

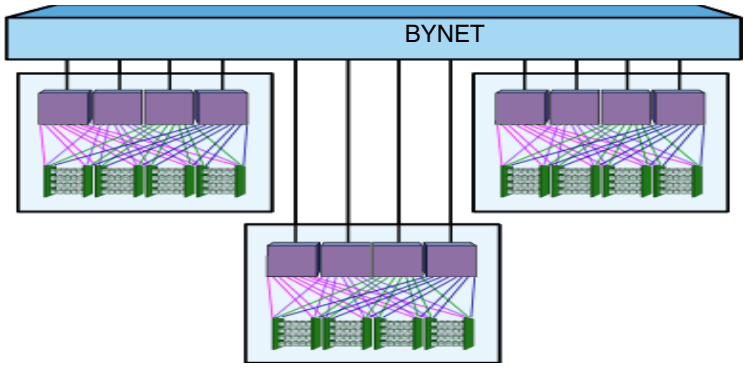
Redundant Disk Arrays

You can reduce the need for fallback by using Redundant Array of Independent Disks (RAID) technology. RAID 1 (mirroring), RAID 5 (parity), or RAID S (parity) protects data in the case of a disk failure.

Note: With Teradata for Windows 2000, the use of RAID 1 instead of RAID 5 is recommended if your applications do a high volume of updates.

BYNET Protection of Nodes/Disks in a Clique

The dual-redundant BYNET interconnect dynamically redistributes vprocs and node traffic within a clique. Banyon connectivity also ensures that all disks in a clique are accessible by all other nodes in the same clique. The following illustration shows the connectivity of three four-node cliques.



This enables an MPP system to continue to run if a node or disk fails in a clique, even if you do not specify the fallback option. (For 7x24 systems, however, the fallback option is recommended to minimize the risk of system downtime.)

Effects of Clustering and Fallback

When tables are defined with the fallback option, grouping AMPs into clusters helps protect data availability, even if data in two or more AMPs in different clusters is corrupted.

Clusters are user-defined and can span cliques. With clustering and fallback, the system copies the fallback copy of each data row in one AMP to an AMP in another clique. Therefore, if data in two AMPs is corrupted simultaneously in different clusters, all fallback rows are available from the remaining AMPs in each cluster.

Without clustering, the system acts as if there is only one cluster, so if data in two or more AMPs is corrupted at the same time anywhere in the system, Teradata fails.

For Teradata to fail with clustering, data must be corrupted in two or more AMPs at the same time.

Performance and Cluster Size

Typically, clusters consist of from four to eight AMPs. For most applications, a cluster size of four provides a good balance between data availability and system performance.

When data in one AMP is corrupted, the other AMPs in the cluster must handle all operations. This means the larger the cluster size, the less performance degradation you will see. For example, if data in one AMP in a two-AMP cluster is corrupted, processing takes twice as long; if data in one AMP in a four-AMP cluster is corrupted, processing takes only 33% longer.

Fallback has no negative effect on the processing of retrievals and most Data Definition Language (DDL) operations (for example, creating tables and views). In fact, without fallback, a retrieval does not succeed if an AMP on which table data is accessed is not operational.

As long as two or more AMPs in a cluster are not down at the same time, a SELECT operation succeeds every time with fallback-protected data.

Choosing AMP Clustering Assignments

AMP failures are usually the result of a hardware-related problem. To protect the availability of your data, define AMP clustering assignments as closely as possible to the following fault-tolerant strategy:

- No two or more AMPs of a cluster on the same node (for MPP systems).
- No two or more AMPs of a cluster in the same node cabinet.
- No two or more AMPs of a cluster serviced by the same disk array cabinet.
- No two or more AMPs of a cluster serviced by the same disk array controller.

PJ Option

If you decide that fallback protection is too costly in terms of storage space, PJ offers the following advantages:

- This data protection method consumes a minimum of space; space consumption is low because the system copies only the affected rows.
- Journal rows provide you with the means to roll the contents of a table forward or backward in time.
- You can checkpoint, save, and restore PJ tables. PJ contents remain until you specifically delete them.
- Parameters to the PJ option provide either single or dual copies of before-images, non-local (remote) single or dual copies of after-images, or local single copies of after-images.
- The PJ option is available at the database level and on a table-by-table basis. Only one PJ per database is allowed, but a table in one database can write to a PJ in another database.

Distribution of Journal Images

When determining the type of PJ to select, consider the distribution of journal rows. The following table describes the row distribution associated with each type of option.

If you do not specify the DUAL option, the system maintains a single copy of the journal for non-fallback tables.

Option	Description
Not local (remote) journal	If you specify a single, non-local journal for a table without fallback, the system writes a copy of each changed data row to a "backup" disk (a disk other than the primary disk but in the same cluster). Performance is slower for non-local journals than for local journals because the journal rows must travel to a different AMP and disk. However, a failure of the primary disk does not affect the journal rows. With a single copy of a non-local journal, you can recover the data of a non-fallback table by restoring the appropriate archive tape and performing a ROLLFORWARD or ROLLBACK using the after-image or before-image rows of the journal table.
Local journal	A local journal writes to the same disk that contains the primary data rows. Performance is faster compared to remote journaling because communication among the AMPs is not required. Also, recovery is faster with local journals than with non-local journals, as long as the primary disk is not damaged or corrupted. However, a failure of the primary disk affects the journal rows as well as the data rows. This means that *both the local journal data and the primary data could be lost if*: • Corruption occurs on the local disk • The data table is not defined with fallback If you use a remote journal, the system can recover the journal entries.
Dual journal	If you specify the DUAL option for a non-fallback table, the system maintains two copies of the original changed data rows. The system writes one copy to the primary disk, and one copy to a backup disk in the same cluster.

Note: Regardless of the type of PJ you select, for a table defined with the fallback option the system writes journal images of both the primary data row and the corresponding fallback row.

Distribution of PJ Images

The following table summarizes the distribution of journal images according to the type of journal option and the protection level of the data table:

Journal option	No Fallback	Fallback
BEFORE (single)	Primary rows on backup disk.	Primary and fallback rows on backup disk
DUAL	Primary rows on primary and backup disks	Primary and fallback rows on primary and backup disks.
NOT LOCAL, AFTER (single)	Primary rows on backup disk.	Primary and fallback rows on backup disk.
LOCAL, AFTER (single)	Primary rows on primary disk.	Primary rows on primary disk and fallback rows on fallback disk

Performance Considerations

When deciding whether to create a table with fallback protection, PJ protection, or both, consider the following:

- Fallback does not impact retrieval time, but does affect UPDATE performance because processing time more than doubles for Data Manipulation Language (DML) operations (for example, UPDATE, INSERT, DELETE).
- A duplicate copy of a table doubles the storage space occupied by the table.
- If you want extra protection for data-critical tables, consider using a PJ. Depending on your Teradata platform (SMP or MPP) and RAID configuration, PJ protection may be best instead of fallback, or in addition to fallback.
- Because fallback and journaling are options of the CREATE TABLE, CREATE INDEX, CREATE JOIN INDEX, and ALTER TABLE statements, you can make your choice on a table-by-table and index-by-index basis.

PDE-Dump and Disk Array Plus (NCR UNIX)

Introduction

Before Teradata V2R5.0 on NCR UNIX MP-RAS systems, a PDE dump was directed to a single, internal SCSI dump device using synchronized I/O. This configuration was not conducive to good dump performance.

Disk Array Plus

Teradata V2R5.0 adds support for Disk Array Plus (DAP) as a storage device for PDE dumps. DAP is a software product used to configure disks as Redundant Arrays of Independent Disks (RAID).

Performance

There are several types of RAID implementations. All are designed to improve performance and reliability. The DAP preserves the PDE dump, single-device view while taking advantage of the I/O throughput improvement provided by striping the DAP device across multiple SCSI devices. Data striping is RAID0.

Striping data across the disks in parallel provides increased performance, versus dumping data to a single disk. It also shortens PDE dump time during restarts increasing system availability.

Determining Table and Database Size Growth Rates

Introduction

By taking a snapshot once a month of the current size and row count of tables and databases, you can determine actual growth rates. Use this information to track growth trends and for capacity planning.

As tables become larger and access increases, it may be necessary to adjust the frequency of collect statistics functions or to add new indexes.

- Number of bytes

```
Select databasename, tablename, sum(currentperm)
from dbc.tablesize group by 1,2 order by 1,2;
```

- Number of rows

```
Select count(*) from tablexxx;
```

Chapter 14:

Database Locks and Performance

This chapter provides information on handling database locks in order to improve performance.

Topics include:

- Locking overview
- What is a deadlock?
- Deadlock handling
- Avoiding deadlocks
- Locking and requests
- Locking and transactions
- Locking rules
- LOCKING ROW / NOWAIT
- Locking and client (host) utilities
- Transaction rollback and performance

Locking Overview

Introduction

When multiple transactions need to perform work that requires a non-sharable lock on the same object, the Teradata Lock Manager controls concurrency by:

- Granting a lock to the transaction that requests access first
- Queuing subsequent transactions
- When the current transaction completes, releasing the lock and granting a new lock to the oldest transaction in the queue

The system includes two internal timers, but no time-out mechanism exists for transactions waiting in a queue.

Note: However, the MultiLoad client utility can time-out MLOAD transactions waiting for over 50 seconds (see *Teradata MultiLoad Reference*).

On Teradata, the following mechanisms exist:

- A mechanism that determines whether the transaction limit for a locking queue has been reached and if so, sends an error message to the session owning any transaction needing to be added to the same queue.
- A hung-transaction detection mechanism that detects and aborts transactions hung due to system errors.
- A deadlock detection mechanism that detects and aborts and rolls back the youngest transaction and sends an error message to the session owning that transaction.
- A user-tunable value that determines the time interval between deadlock detection cycles. (See dbscontrol in *Teradata RDBMS Utilities*.)

Most transactions on Teradata are processed without incurring a deadlock. For detailed information on lock compatibility and contentions, refer to *Teradata RDBMS Utilities*.

The rest of this section discusses the locking scheme and explains how to investigate transaction locks with the Lock Display utility.

Locking Levels

Locking levels determine the type of object that is locked, as follows:

This lock level…	Is used for…
Database	Data Definition Language (DDL) statements such as CREATE, DROP, or MODIFY DATABASE or USER.
Table	• Data Manipulation Language (DML) statements that access a table without using a primary index or a unique secondary index • Table-level DDL statements such as CREATE TABLE, VIEW, or MACRO and ALTER TABLE.
Row hash	DML statements that access by primary index or unique secondary index. Rowhash locks are the least restrictive. Other transactions may access other rows in the table while the rowhash lock is held. All rows with the same row hash are locked at the same time.
Row hash range	Rowhash locks within a range.

Locking Modes

Locking modes determine whether or not other users may access the target object. Locking modes include:

This lock mode…	Is placed…
Exclusive	only on a database or table when the object is undergoing structural changes or being restored by a host utility. Prohibits access to the object by any other user.
Write	in response to an INSERT, UPDATE, or DELETE request. Restricts access by other requests, except those that specify an access lock.
Read	in response to a SELECT request. Restricts access by requests that require exclusive or write locks.
Access	in response to a user-defined LOCKING FOR ACCESS clause. An access lock is shareable, permitting the user to read an object that may be already or concurrently locked for read or write.

Lock Display Utility

Use the Lock Display utility to display currently-held transaction locks. Transactions are identified by host ID and session ID.

Lock Display can return a variety of information, including but not limited to:

- Table-level locking on a specific table or all tables
- Rowhash-level or rowrange-level locking on a specific table or all tables
- Locking on specific or all databases
- Blocked transactions and those causing the block
- Internal locking information (for example, row control blocks, transaction control blocks, and so forth)

Each type of display can be requested for a sampling of AMPs or for ALL AMPs.

Caution: Be careful about an ALL AMPs or all databases display, especially on a system with many AMPs and a heavy workload, as the volume of information can be unmanageable. NCR recommends obtaining information from all AMPs only for a specific table or transaction.

For a complete description of and operating instructions for Lock Display, see *Teradata RDBMS Utilities*.

What Is a Deadlock?

A deadlock is the database equivalent of gridlock. Two transactions are deadlocked when each is waiting for the other to release a non-sharable lock on the same object.

The difference between a deadlock and a blocked request is as follows:

- A deadlock exists when at least two concurrent requests are each waiting for the other to release a lock on the same target object.
- A request is blocked when it is waiting in a queue for a long-running job to release a non-sharable lock (Write or Exclusive) on the target object.

A deadlocks can involve locks at the row hash level as well as the table level.

How Deadlocks Occur

A deadlock can occur in the following environment:

Stage	Process	
	IF...	**THEN...**
1	you use a Primary Index (PI) or Unique Secondary Index (USI) constraint in a SELECT statement	the lock manager applies a read lock on the row or set of rows that hash to the same value.
2	the same transaction constrains a subsequent DML statement	the lock manager upgrades the read lock to a write or exclusive lock.
3	concurrent transactions simultaneously require this type of upgrade on the same row hash	a deadlock can result.

Example

Assume that two concurrent users use the same PI value to perform a SELECT followed by an UPDATE, as follows:

UserA enters:

```
BEGIN TRANSACTION;
SELECT y FROM tableA WHERE pi =1;
UPDATE tableA SET y=0 WHERE pi =1;
```

UserB enters:

```
BEGIN TRANSACTION;
SELECT z FROM tableA WHERE pi=1;
UPDATE tableA SET z=0 WHERE pi=1;
```

Both users may simultaneously access the row for read during the SELECT process.

When the UserA UPDATE statement requires a write lock on the row, it must wait for the UserB read lock to be released.

The system cannot release the UserB read lock because the UserB UPDATE statement requires a write lock on the row. That request is queued waiting for the system to release the UserA read lock.

This sequence results in a deadlock.

LOCKING Clause with CREATE INDEX

CREATE INDEX processing begins with either a WRITE or a READ lock, which is upgraded to an EXCLUSIVE lock when the dictionary tables are updated.

CREATE INDEX requires a WRITE lock if you omit the LOCKING modifier or specify a LOCKING FOR WRITE clause. WRITE and EXCLUSIVE are both non-shareable, so there is no conflict when the WRITE lock is upgraded to an EXCLUSIVE lock.

CREATE INDEX uses a READ lock if you specify a LOCKING FOR READ/ACCESS/SHARE lock, which allows other transactions to concurrently read data from the same table.

When CREATE INDEX is ready to upgrade to an EXCLUSIVE lock, whether the upgrade can be granted or CREATE INDEX has to wait depends on whether any other transactions are running against the same table. If so, CREATE INDEX is blocked until those transactions complete. Once the EXCLUSIVE lock has been granted, CREATE INDEX blocks all subsequent transactions until it completes.

This procedure improves concurrency by reducing the time SELECT statements wait in a queue for the target table. However, it also allows a deadlock situation to arise.

If you are researching a deadlock situation, be aware that a CREATE INDEX statement running under a READ/ACCESS/SHARE lock might be the offending transaction.

Deadlock Handling

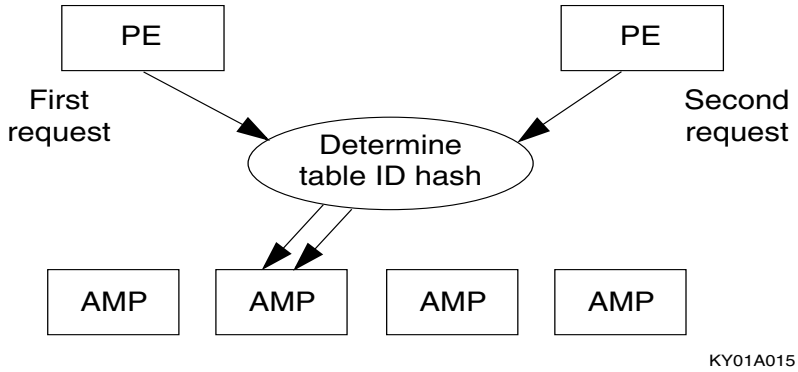
AMP-Level Pseudo Locks and Deadlock Detection

Pseudo table locks reduce deadlock situations for all-AMP requests that require write or exclusive locks.

Internally, each table has a table ID hash code and each table ID hash code is assigned to an AMP. With pseudo table locking:

- Each AMP becomes a gate keeper of the tables assigned to it.
- All-AMP requests for non-sharable (write or exclusive) locks go through the gatekeeper.
- If a non-shareable lock is being held for one all-AMPs request when another such request is received, each gatekeeper forms and manages its own locking queue.
- AMPs also look for deadlocks at the local level. AMP-local deadlock detection runs at fixed 30-second intervals.

The following illustrates pseudo table locking:



Example

Following is an example of the pseudo table locking process.

Stage	Process
1	UserA sends an all-AMP request for a non-shareable lock.
2	The PE sends a message to the gatekeeper AMP for the table.
3	The AMP places a rowhash lock on the internal tableID 0,3. The hash value is the tableID of the data table to be locked.
4	If no write or exclusive lock exists on the data table, UserA gets the non-shareable lock and proceeds with the all-AMP request.

Stage	Process
5	UserB sends another all-AMP request for the same table.
6	The PE sends a message to the gatekeeper AMP for the table.
7	Since system table 0,3 has a rowhash lock identifying the data table, the AMP knows that UserB must be queued.
8	UserB has to wait, but is next in line until the lock being held by UserA is released.

Global-Level Deadlock Detection

Deadlocks are rare because of the pseudo table locking mechanism at the AMP level, but they are still possible. At the global level, they are detected and handled as follows:

On Teradata...	On the client...
• Within the dispatcher partition of the parser engine, the global deadlock detection routine runs at intervals set by the DeadLockTimeout value in the DBS Control Record. • If a deadlock is detected, the routine determines which transaction to abort (usually the youngest), rolls it back, and generates a code 2631 error message. • You can reduce the interval between deadlock detection cycles by lowering the value in the DeadLockTimeout field (see page 5-8 and "DBS Control Utility" in *Teradata RDBMS Utilities*).	• The application must retry a transaction rolled back due to a deadlock (error code 2631). **Note:** DBC.SW_Event_Log tracks error 2631 (see *Teradata RDBMS Data Dictionary*). DBS sends this error when the number of transactions allowed to be in a locking queue is exceeded, requests are in conflict, or an internal error is encountered. If BTEQ receives an error 2631 and the BTEQ command ".SET RETRY ON" is active, RETRY automatically retries the rolled-back statement and any subsequent statements in the same transaction. Any statements in the transaction prior to the failed statement are not retried because they are lost, along with information that a transaction is in progress. (For details, see *Teradata BTEQ Reference*.) **Note:** Avoid using BTEQ to handle update transactions; it is not designed to be a transaction processor. • Other applications must be coded to – Check for error 2631 – Retry the transaction.

Caution: The global deadlock detector does not detect HUT locks placed by ARC. Use the ShowLocks utility to find out if HUT locks are being held on database entities. After the utility has finished processing, you can remove any active HUT locks with the RELEASE LOCK command. (See "Locking and Client (Host) Utilities" on page 14-14).

Avoiding Deadlocks

Guidelines

Follow these guidelines to prevent excessive deadlocking:

- Except with CREATE INDEX, use LOCKING FOR ACCESS whenever dirty reads are acceptable.
- With CREATE INDEX, omit LOCKING or use LOCKING FOR WRITE, if avoiding deadlocks is your objective. (To reduce the number of transactions waiting in locking queues because of index processing, use LOCKING FOR READ.)
- Beware of BTEQ handling of transaction processing. After transaction rollback, BTEQ continues the transaction from the point of failure, not at the beginning of the transaction!
- Set the DeadLockTimeout field via the dbscontrol utility to 30 seconds if you have a mix of DSS and PI updates on fallback tables.
- Be sure to use RELEASE LOCKS on Archive/Recovery jobs (the global deadlock detector does not recognize HUT locks).
- Use the Locking Logger to monitor and detect locking problems.
- Use the LOCKING ROW [FOR] WRITE/EXCLUSIVE phrase preceding a transaction. This phrase does not override any lock already being held on the target table. LOCKING ROW is appropriate only for single table selects that are based on a primary or unique secondary index constraint. For example:

  ```
  .
  .
  LOCKING ROW FOR WRITE
  SELECT y FROM tableA WHERE pi =1;
  UPDATE tableA SET y=0 WHERE pi =1;
  .
  .
  ```

- In macros, use multi-statement requests instead of Begin Transactions (BT)/End Transactions (ET) to minimize table-level deadlocking. For example:

  ```
  .
  .
  LOCKING ROW FOR WRITE
  SELECT y FROM tableA WHERE pi =1
  ; UPDATE tableA SET y=0 WHERE pi =1 ;
  .
  .
  ```

  This causes all the necessary locks to be applied at the start, which avoids the potential for a deadlock. Use the EXPLAIN modifier to check out the processing sequence.

Locking and Requests

Introduction

Request locks are acquired up front in ascending TableID order, which minimizes the chance of deadlocks if other users execute the same request, or if users execute other requests using the same tables.

The term request refers to any of the following:

This request type...	Is used with...
Multi-statement request	only DML requests.
Single statement request	DDL or DML requests.
Macro	multi-statement or single statement requests, following the above rules.

These request types above are also considered implicit transactions (or system-generated transactions). Therefore, the system holds locks for these request types until the requests complete.

The table-level write locks needed in these requests are:

- Acquired in TableID order
- Held until done

This minimizes deadlocks at the table level when many users execute requests on the same tables.

Sample Multi-Statement Request

Following is a multi-statement DML request. The end of the request is defined by a semi-colon at the end of a line.

```
UPDATE Dept
SET Salary_Change_Date = Date;
UPDATE Manager
SET Salary_Amt = Salary_Amt*1.08;
UPDATE Employee
SET Salary_Amt = Salary_Amt*1.04;
```

Locking and Transactions

Example

The following example illustrates a multi-request explicit transaction (also called a user-generated transaction). It contains two requests, and is an explicit transaction because it is framed by a BT and ET statement.

```
BT;
SELECT *
FROM Manager
WHERE Manager_Employee_Number = 1075;

UPDATE Employee
SET Salary_Amt = Salary_Amt*1.08
WHERE Employee_Number = 1075;
ET;
```

Note: This is not the same as the multi-statement request example on page 14-10. Locks are not acquired up front in TableID order because it is not a single request. The multi-request explicit transaction does not avoid deadlocks.

As the system executes an explicit transaction, each request within the explicit transaction places locks. The system parses each request separately. The system cannot arrange table locks in an ordered fashion to minimize deadlocks.

The system holds locks acquired during a transaction until the transaction completes.

Explicit transactions are generally coded within Pre-Processor or Call-Level Interface (CLI) programs (rather than through BTEQ scripts). An explicit transaction allows the program to process data between requests based on end-user interaction with data retrieved from earlier requests within the transaction.

Note that macros have an implicit BT; ET;. It is not necessary to include them within a macro. You may not have multiple BT; ET; statements in a single macro.

Locking Rules

Row-Level Locks

The system acquires row hash level locks for PI updates on fallback tables in two steps:

Stage	Process
1	Primary row hash is locked; update occurs.
2	Fallback row hash is locked; update occurs. For a single-statement transaction, the parser places a lock on a row only while the step that accesses the row is executing.

Table-Level Locks

Table-level locks occur during the following operations, even if they access just a few rows:

- Update through a non-unique secondary index (NUSI)
- INSERT SELECT operations
- JOIN/UPDATE operations
- Many DDL statements

  Note: DDL statements also cause locks to be placed on the applicable system tables

- Statements preceded by the LOCKING ... FOR ... clause

LOCKING ROW/NOWAIT

Introduction

When a PI or USI constraint is used for a SELECT query, the lock manager applies a Read lock on the row or set of rows that hash to the same value.

If a multi-statement transaction contains a PI or USI SELECT followed by an UPDATE, the lock manager first grants the Read lock and then queues the upgrade request for the subsequent Write or Exclusive lock.

If another transaction requires the same sequence of locks against the same entity at the same time, its Read lock is also granted and its upgrade request is also queued. This can result in a deadlock, because each upgrade request must wait for the Read lock of the other transaction to be released.

You can avoid this as the cause of deadlocks by using the LOCKING modifier with a ROW FOR WRITE/EXCLUSIVE phrase. With LOCKING ROW, a non-sharing Write or Exclusive lock is applied for the duration of the entire transaction, which must complete before a lock is granted to another transaction.

Note: LOCKING ROW is appropriate only for single table selects based on a primary index or unique secondary index constraint.

If you cannot take a chance on your transaction waiting in a queue, use the LOCKING modifier with the NOWAIT option.

LOCKING NOWAIT specifies that the entire transaction be aborted if, upon receipt of a statement, the lock manager cannot place the necessary lock on the target entity immediately.

This system treats this situation as a fatal error and informs you that the transaction aborted. Any processing performed up to the point at which NOWAIT took affect is rolled back.

Locking and Client (Host) Utilities

Introduction

Client, or host, utility (HUT) locks placed by the ARC utility operations differ from locks placed by other operations; for example:

- HUT locks are associated with the user who entered the command, rather than with the session or operation.
- Only the AMPs that participate in the operation are locked.
- Unlike transaction locks, which are released as soon as the transaction completes, HUT locks remain active until you release them.

Warning: HUT locks that are *not* released *are reinstated automatically after a Teradata reset.*

If performance is suffering because transactions are locked out following ARC operations, use the ShowLocks utility to find out if HUT locks have persisted. If they have, you can release them by submitting a separate RELEASE LOCK command; for example, if HUT locks exist on the PJ database, you submit:

```
RELEASE LOCK (PJ) ALL;
```

If you have to log on under a different username from the name associated with the operation that placed the locks, you should use RELEASE LOCK with the OVERRIDE option. See *Teradata RDBMS Archive/Recovery Reference*, "Archive/Recovery Control Language".

The easiest and safest method of controlling HUT locks is to include the RELEASE LOCK option in the command string; for example:

```
LOGON tdpid/dbaname,password;
CHECKPOINT (PJ) ALL;
ARCHIVE JOURNAL TABLES (PJ) ALL,
RELEASE LOCK,
FILE=ARCHIV3120;
LOGOFF;
```

The RELEASE LOCK option releases all HUT locks placed by the current user on a specified database, regardless of when or how the locks were placed.

RELEASE LOCK is available with the ARCHIVE, REVALIDATE REFERENCES FOR, ROLLBACK, ROLLFORWARD, RESTORE, and BUILD commands.

However, not all archive and recovery operations apply HUT locks. The following table shows what type, mode, and level of lock is applied according to the operation being performed.

Command	Lock Type	Locking Mode	Object Locked	For details and instructions, see....
ARCHIVE [DATABASE/ TABLE/ CLUSTER]	HUT	Read	Specified database(s) or table(s) or cluster(s)	*Teradata Archive/Recovery Reference* *ASF2 Administration and Operations Guide*
BUILD	HUT	Exclusive	Data table(s) being indexed	
CHECKPOINT	Transaction	• Write • Read	PJ being checkpointed Data tables writing to that journal	
CHECKPOINT WITH SAVE	Transaction	• Write • Access	PJ being checkpointed Data tables writing to that journal	
DELETE [DATABASE/ JOURNAL]	Transaction	Write	Database or PJ	
RESTORE JOURNAL	HUT	Write	PJ	
RESTORE [DATABASE/TABLE]	HUT	Exclusive	Specified database(s) or table(s)	
ROLLFORWARD/ ROLLBACKWARD	HUT	• Exclusive • Read	• Data table • PJ the data table writes to	

Transaction Rollback and Performance

Introduction

A rollback is a reversal of an incomplete database transaction. If a transaction fails to complete because the database restarts or the transaction's job is aborted, any partially completed database updates must be removed from the affected user tables to assure data integrity.

The Teradata database maintains transaction integrity via the TJ (dbc.transientjournal). The TJ contains data about incomplete transactions, recording a “before image” of each modified table row.

Effects on Performance

Because a rollback can conceivably involve millions or even billions of rows, a rollback can affect the performance and availability of resources in the Teradata database while the rollback is in progress.

The rollback will impact the performance of the system because it competes for CPU with other users. Moreover, a rollback can keep locks on affected tables for hours, or even for days, until the rollback is complete. During a rollback, there is a trade-off between overall system performance vs. table availability.

Rollback Priority

In the event of a rollback, these rows are re-applied to the table at the priority specified in the tunable dbscontrol parameters using the RollbackPriority flag (field 10 of General DBScontrol flags).

How RollbackPriority affects performance is not always straightforward, and is related to the Priority Scheduler configuration, job mix and other processing dynamics.

Setting Rollback Priority to FALSE

The default value is FALSE, which results in all rollbacks being executed in the RUSH$ performance group of the default resource partition (RPO). The intent of this is faster rollbacks at the expense of other online performance. Typically, the RUSH$ performance group and default resource partition are given the highest weights in the system, and with this recommended priority scheduler configuration, rollbacks will execute at the highest priority.

However, it will also put more work in the default partition, contending with other work executing there. FALSE is better for large rollbacks to critical tables accessed by many users because it is better to finish the rollback and make the table available.

Setting Rollback Priority to TRUE

If RollbackPriority is TRUE, rollbacks are executed within the aborted job's performance group and associated resource partition. The intent of this is to isolate the rollback processing to the job's performance group, while not affecting performance of the rest of the system. But "not affecting performance" in this case refers to CPU and I/O allocation.

If the rollback has locks on tables that other users are waiting for, this causes a greater performance impact for those users, especially if the rollback is in a lesser-weighted performance group. Also, if the rollback is executing under ABS or REL policy, there is the potential for the rollback to exceed the resource limits. If this is the case, the rollback will run at the priority of the allocation group, but will not be capped in its CPU usage, as other work in that allocation group will. TRUE is better for smaller rollbacks to non-critical, less extensively used tables.

Teradata has made the conscious decision to optimize normal workflow so exception cases such as a rollback will usually take 1.5 to 2 times longer than the runtime of the job. Problems can occur in those cases in which SIs are involved. USI updates are also logged into the TJ. If a USI is involved, the system will do two updates per row.

The real killers are NUSIs. NUSI changes are not logged in the TJ so a row hash comparison and possibly a byte-by-byte comparison of matching rows are executed to insure that duplicate rows are avoided. When NUSI's are involved, it is not unusual to see rollbacks that take greater than two times the original job run time. If multiple tables are involved with a complex join, rollback times in days rather than hours can occur.

Ways to Minimize or Avoid Rollbacks

Because rollback is an increasingly important, especially with sites running "ad-hoc", OLTP/OLCP, and CRM applications, a "rollback cancel" facility is now part of Teradata V2R5.0. Consider the use of TDQM, which will allow automatic SQL evaluation and defer or reject the request based on user specified criteria. SQL developers can also take steps to limit the impact of a rollback on their code.

If doing a lot of deletes on rows from a table, consider the use of Multiload instead of BTEQ/qryman to do the deletes. Multiload completely avoids use of the TJ, and is restartable.

To minimize the number of TJ rows being generated on an insert select, consider doing a multi-statement insert select into an empty table from both of the other tables. This only puts one entry into the transient journal to let the DBS know that the target table is an empty table and to drop all the rows in the target table if the need for a rollback is encountered. After the new table is created, drop the old table and rename the new table to the name of the old table.

More details on both the delete rows and insert select are available online in Support Link Knowledge Base.

How to Detect a Rollback in Progress

Sessions in rollback may appear to have logged off in both DBC.Logonoff and DBC.AccessLog, but this is not always the case. The logoff would depend on the manner in which the job was aborted. If you specify one of the following:

```
ABORT SESSION hostid.username LOGOFF
ABORT SESSION *.username LOGOFF
ABORT SESSION hostid.* LOGOFF
```

the "LOGOFF" option would terminate the session. Without it, the session should continue until the abort completes or the Supervisor issues a LOGOFF request. Unless an SQL job is explicitly coded to do otherwise, a session will also appear to have logged off if the system has undergone a restart.

The rollback or abort, is independent of the session. It is actually handled by a completely different mechanism with internally allocated AMP Worker Tasks.

There is a fairly simple but not completely foolproof way to identify the existence of a rollback.

Issue the command:

```
# rallsh -sv "/usr/ntos/bin/puma -c | grep -v ' 0 ' | grep
MSGWORKABORT "
```

If this command returns any lines like:

```
MSGWORKABORT 3 999 1 2
MSGWORKABORT 3 999 1 2
MSGWORKABORT 3 999 1 2
<... etc...>
```

on multiple successive samplings, then there is very likely someone in rollback. In a short, one-time only sample, the tasks in MSGWORKABORT could also have been finishing up an END TRANSACTION and are not actually aborting. Since we are looking for high-impact, long-running aborts, we look for some VPROC(s) with tasks in MSGWORKABORT for several minutes.

```
The actual output of the "puma -c" command is:
VPROC = 6
WorkType Min Max Inuse Peak
MSGWORKNEW 3 50 0 8
MSGWORKONE 3 999 0 7
MSGWORKTWO 3 999 0 2
MSGWORKTHREE 3 999 0 2
MSGWORKFOUR 0 999 0 0
MSGWORKFIVE 0 999 0 2
MSGWORKSIX 0 999 0 1
```

```
MSGWORKSEVEN 0 999 0 0
MSGWORKEIGHT 0 999 0 0
MSGWORKNINE 0 999 0 0
MSGWORKTEN 0 999 0 0
MSGWORKELEVEN 0 999 0 0
MSGWORKABORT 3 999 2 2 <=Inuse shows 2 AWTs doing ABORT
MSGWORKSPAWN 3 999 0 2
MSGWORKNORMAL 3 999 0 3
MSGWORKCONTROL 3 999 0 2
```

Look for a non-zero value in the "inuse" column for MSGWORKABORT. In the example above, two AMP Worker Tasks are being used in the abort process.

Following a restart, additional details on the rollback can be obtained from the RcvManager utility. See *Teradata RDBMS Utilities* for a description of this utility.

Chapter 15:

General Performance Enhancements

This chapter describes various V2R5.0-specific performance enhancements that require no user intervention.

Topics include:

- Reconfiguration and performance
- Effect of roles and profiles on performance
- Sessions, jobs, and performance
- "Optimized DROP MACRO" feature
- Reducing all AMP operations
- Constraints optimization
- Update performance features
- Teradata TPump and join index
- Optimizer enhancements
- Parallel dump

Reconfiguration and Performance

Introduction

Teradata V2R5.0 contains the enhancements to the following utilities that make reconfiguration quicker and easier:

- MOVE AMP and MOVE PE commands in the Configuration utility
- Parallel processing of tables by the Reconfiguration utility
- New utility in Parallel Upgrade Tool (PUT) 3.0 to support the MOVE AMP and MOVE PE commands.

Before the implementation of the new features, highly trained individuals moved AMPs manually. The process was highly constrained by the configuration. In Teradata V2R5.0, AMP vproc data can be moved at the logical level rather than at the physical level, thus removing many configuration constraints.

For information on the Configuration utility, the Reconfiguration utility, and pdeconfig, see *Teradata RDBMS Utilities*.

Configuration Utility

To support moving AMP vprocs to new storage units, Teradata V2R5.0 contains the MOVE AMP and MOVE PE commands for the Configuration utility.

- The MOVE AMP command specifies that all data rows from an AMP or a group of AMPs are to be moved to another AMP or group of AMPs.
- The MOVE PE command specifies that a PE or group of PEs is to be moved to another PE or group of PEs.

Reconfiguration Utility

Before Teradata V2R5.0, the Reconfiguration utility processed only one table at a time rather than multiple tables in parallel and displayed no indication of status.

Teradata V2R5.0 adds a new STATUS command and additional syntax elements to the existing RECONFIG command.

Parallel Upgrade Tool

To support the MOVE AMP and MOVE PE commands, Teradata V2R5.0 adds a new function to PUT 3.0 called Move AMPS and PEs.

The tool performs validation, error checking, and so on, on user input. Then it generates a new vconfig.out file containing the new AMP and PE definitions you specify.

See *Parallel Upgrade Tool (PUT) User Guide for UNIX MP-RAS Release 3.02* for more information about PUT 3.0 Move AMPs and PEs.

Performance Impact

The enhancements and new features associated with the Reconfiguration utility enables it to:

- Process up to ten reconfiguration sessions in parallel
- Process tables in parallel to better utilizes nodes, disks, and the BYNET
- Reduce the time required to reconfigure a system
- Increase system availability

Effect of Roles and Profiles on Performance

Introduction

Before the introduction of the Roles feature in Teradata V2R5.0, you could grant rights to a larger user base two ways:

- Use the ALL option in the GRANT statement to grant rights on the shared database objects to a database. The following example uses the department database:

  ```
  GRANT right ON database object TO ALL department database;
  ```

  Then, create users under the department database. The system automatically granted all rights held by the database to each user created under the database.
- Grant rights to users individually

Teradata V2R5.0 introduces the Roles feature that enhances rights management.

Benefits of Roles

A role is a collection of access rights. Rights are first granted to a role and the right to use the role is then granted to users. The Roles feature introduces a new method of rights administration and offers these benefits:

- Simplified access rights administration

  You can grant rights on database objects to a role and have the rights automatically applied to all users assigned to that role.
- Reduced size of the DBC.AccessRights system table

  The DBC.AccessRights system table is much smaller because rights are maintained on a role, rather than an individual level. Instead of inserting one row per user per right on a database object, the RDBMS inserts one row per role per right in the AccessRights table.
- Reduced disk space usage

  Less disk space is required for smaller DBC.AccessRights table
- Improved performance

  Dictionary contention for Data Definition Language (DDL) operations, especially CREATE USER, is reduced because operations are shorter.

Benefits of Profiles

A profile is a set of common user parameters that can be applied to a group of users. Profile parameters include:

- Account
- Default database

- Spool and temporary space allocations
- User-level password attributes

The Profiles feature simplifies user management as follows:

- Allows you to create a profile, which contains one or more of the command parameters that you assign to a group of users, for example every user created in a department database
- Change the profile instead of changing parameters for each user

Performance

The Roles feature provides the following performance benefits:

- Reduces disk space required DBC.AccessRights and usage when rights are managed on role level rather than individual level.
- Improves performance and reduces dictionary contention for DDL tables, especially CREATE USER.

Sessions, Jobs, and Performance

Introduction

A good "performance practice" is to check periodically the user logon rate and the number in dbc.logonoff to spot excessive sessions or logons.

Collecting user logons per day per workload lets you map query throughputs for predictions based on knowing increases in user logons. In addition, user logons per day per workload can be compared to canary responses.

Status Check Jobs

Moreover, establishing "status check" jobs that logon, check for table entries and logoff.

But if this is done in excess, the overhead to support the logon can have significant system impact.

Sample Query

The following query can be used to set up such a status check job:

```
/*================================================*/
/* Users logging on more than 100x/hour */
/*================================================*/
SEL a.LogonDate(title'Date')
,cast (a.LogonTime/10000 as integer)*100
(FORMAT'99:99', TITLE '// //Hour')
,A.username (Format 'x(12)') (Title 'User Name')
,COUNT(a.sessionno)
(title'Number of//sessions:',NAMED sesscount)
FROM DBC.LogOnOff a, DBC.LogOnOff b
WHERE a.Event='Logon'
AND b.Event<>'Logon'
AND a.LogonDate=b.LogonDate
AND a.LogonTime=b.LogonTime
AND a.SessionNo=b.SessionNo
AND a.IFPNo=b.IFPNo
/* ENDING DATE & TIME */
AND ((a.LogonDate= DATE and a.LogonTime <= TIME)
OR a.LogonDate < DATE)
/* BEGINNING DATE & TIME 0 = MIDNIGHT, 120000 = NOON */
AND ((b.LogonDate = DATE and b.LogonTime >= 0)
OR b.LogonDate > DATE)
GROUP BY 1,2,3
ORDER BY 1,2,4
HAVING sesscount> 100;
```

"Optimized DROP MACRO" Feature

Introduction

DDL operations such as DROP MACRO, DROP VIEW and DROP PROCEDURE benefit from a Teradata V2R5.0 feature called "Optimized DROP MACRO."

Any SQL applications that use CREATE and DROP objects such as a MACRO, a VIEW and a PROCEDURE benefit from Optimized DROP MACRO if the objects are newly created in Teradata V2R5.0 and no GRANT operations are executed on them.

Teradata TPump applications, for example, usually use MACROs on the fly and they benefit from this feature, too. The feature:

- Enhances DDL performance by increasing the concurrency of qualified DROPs and other DDL operations
- Potentially reduces response time of qualified DROPs
- Reduces CPU and disk I/O usage by qualified DROPs
- Improves performance generally for concurrent DDL and DML operations due to no lock conflicts on the AccessRights table.

Example

Full-file DELETE from the AccessRights dictionary table is a well-known performance problem because a table-level write lock is needed. That conflicts with DML operations that need to read AccessRights rows.

The Optimized DROP MACRO feature eliminates the costly full-file DELETE operation in cases where a rowhash-locked prime-index DELETE can be used instead, specifically whenever no GRANT operations have been performed on the object being dropped.

In such a case, all rows added to the AccessRights table by the CREATE have the same NUPI value (UserID and DatabaseID fields) and therefore these rows can be deleted with a prime-index DELETE instead of a full-file DELETE.

Performance Considerations

One possible negative performance impact from using rowhash locks rather than table-level locks is that deadlocks can occur when the DROP operation is running concurrently with another operation that requires a table-level lock on the AccessRights table.

Previously, when two requests were running concurrently using table-level locks on the same table, there would be no deadlocks due to the use of pseudo locks.

Reducing All AMP Operations

Introduction

The Operational Data Store (ODS) workload operates on a small number of rows and becomes inefficient when all AMPs are used. By recognizing ODS cases and executing them on only a few AMPs, ODS workload becomes much more efficient, increasing throughput and scalability.

The enhancements in Teradata V2R5.0 reduce or eliminate the number of all-AMP operations that are required to process a given workload. This optimizes the processing of all-AMP operations as follows. It:

- Avoids all-AMP work after the release of a transaction group
- Reduces messages for the return of a small number of rows
- Reduces all-AMP messages for aggregates
- Reduces all-AMP Merge and Merge-Delete step
- Supports group-AMP operations as follows:
  - Single table join indexes
  - Nested join (hash NUSI equivalents)
- Supports multi-lock requests

Avoiding All-AMPs Work After Release of the Transaction Group

Currently, during query processing, retrieving result spool data for non-PI operations requires an all-AMP BYNET merge, regardless of the number of AMPs involved with spool data. This is a very expensive process.

Enhancements in Teradata V2R5.0 eliminate the unnecessary all-AMP operation. The dispatcher enters only the AMPs involved with spool data in a dynamic group and passes the group to the BYNET Merge task. The enhancement enables the BYNET Merge task to handle the dynamic group instead of all AMPs.

When only one AMP has data to return, the BYNET Merge task is not used at all.

Reducing Messages for a Small Number of Rows

Certain ODS transactions return a small number of rows from many AMPs. To retrieve rows, the BYNET driver sends a message to a given AMP. Because the number of rows is small, the rows can fit in one response message.

Later, the BYNET driver sends another message to try to retrieve more rows, and the AMP responds with an indication that there are no more rows to send. In Teradata V2R5.0, AMPs can return "end of data" status with the data to indicate that the AMPs have no more rows to send. This capability saves one message per AMP.

The savings is especially beneficial for short queries.

Reducing All-AMPs Messages for Aggregates

For aggregate statements when a GROUP BY clause is not specified, enhancements in Teradata V2R5.0 reduce all-AMP messages as follows:

- Send SUM step to the AMP that is designated as the global aggregate executor.

  The global aggregate executor performs global aggregate processing for all groups. This reduces the number of all-AMP operations for the step by one half.
- Broadcast the step-to-step aggregation message to all AMP to build local aggregate results.

Reducing All-AMPs Merge and Merge-Delete Step

Currently, when executing a query plan, Merge and Merge-Delete operations almost always involve all AMPs. These all-AMP operations can add extra message traffic and task initiations and require extra disk accesses. This overhead can be costly for short queries. In some cases, the all-AMP initiation is not necessary.

To improve performance, Merge and Delete can, in most cases, be combined with the Retrieve and Join steps in the plan.

Supporting Group AMP Operations

Enhancements in Teradata V2R5.0 designed to reduce all-AMP operations include the group AMP capability.

Previously, when one of the sources for a Join step is a spool file containing rows from a previous Retrieve-Join step and the spool was redistributed, the Join step is an all-AMP step because the parser does not know in advance which AMPs have the redistributed spool rows.

Because the result of a join plan is always a spool relation, Teradata V2R5.0 enhances the parser to recognize whether the data in the spool was redistributed to one of the following:

- One AMP
- A group of AMPs
- All AMPs

The group AMP enhancement is particularly beneficial to the following Teradata database features:

- Single-table join indexes
- Nested join applications

Each feature is discussed in the following sections.

Supporting Single Table Join Indexes

The Optimizer can use a single table join index that does not necessarily cover all columns of a base table involved in a query. When more columns are required to process the query, the Optimizer uses the primary index or RowID of the base table rows to re-join with the base table for additional columns.

The Teradata database Single Table Join Index feature provides capabilities equivalent to those of a hashed NUSI, which the Teradata database does not support.

Access to a table by way of a non-hashed NUSI is an all-AMPs operation. If the NUSI were hashed, access to rows would be by means of a single-AMP step based on the hash of the access value. A single table join index can do something similar.

A single table join index contains any number of columns from the base table and has a primary index defined on columns that are different from that of the primary index of the base table. When a single table join index is defined with either RowID or primary index columns of the base table and hashed on a column other than the primary index column of the base table, the single table join index can be viewed as an auxiliary index to the base table, similar to a hashed NUSI.

Supporting Nested Join Applications

Replacing NUSIs with single table join indexes enhances processing for queries containing nested joins by introducing the group AMP capability. Steps that were sent to all AMPs are now sent to a group of AMPs that have the result data.

Supporting Multi-Lock Requests

Lock conflict can tie up AWTs when the lock is held up by a long running query. The multi-lock AMP step places a collection of database locks locally.

In Teradata V2R5.0, the lock manager treats this step as a group-lock request. After the group lock is submitted, the lock manager releases the AWT that initiated the lock request regardless of whether the locks have been granted.

If a lock request cannot be granted, the lock manager queues the lock. When the queued lock is granted, the lock manager proceeds to the next lock request. The last request granted in this group of locks triggers the lock manager to send a success response to the MLK (multi-lock) step initiator, the step built into the Parser as part of all other step messages required to process a request.

Constraints Optimization

Introduction

Teradata V2R5.0 constraints optimization:

- Extends transitive closure to include more cases involving equalities and inequalities
- Improves the performance of query plans that contain the UNION ALL set operator

Transitive Closure Extended

Currently, transitive closure exists for the following:

- Bind terms that are equality constraints between two columns
- Explicit terms that are equality constraints between a column and a constant

The Optimizer examines explicit terms and bind terms in order to generate new terms. For example, if A=B and B=5, the Optimizer generates a new term, A=5. This is helpful in increasing the selectivity on a table.

Teradata V2R5.0 allows the Optimizer to consider miscellaneous terms when the operator of the condition is an inequality. Transitive closure is extended to include inequalities. In the example A>B and B > 5, the Optimizer can derive A>5.

Improvements to Performance of UNION ALL Set Operator

Historically, when spool files resulted from a UNION ALL set operation, the Teradata database did not maintain information about how spools files were hashed, even though the Optimizer might be able to use the information later. This practice caused unnecessary spool file redistribution when the Optimizer performed a join on the hash columns of the spool file later in the query plan.

To avoid the cost of redistribution, the system, as of Teradata V2R5.0, saves the primary index of a local spool file containing multiple selects to base tables that have been combined using the UNION ALL operator.

Usage Notes

No new syntax is required for this feature. The Optimizer will simply attempt this optimization if all sets are base table selects involving the primary index and each set operation is UNION ALL. Each set must be hashed on the same columns (by position in the select list). Also, join terms involving these columns must exist to observe the optimization.

Update Performance Features

Introduction

Update performance features and enhancements improve update, insert, and delete operations. On customer systems, these operations take place throughout the day concurrently with query access to the data.

Specific Teradata V2R5.0 update performance enhancements include:

- Improving NUSI INSERT performance
- Modifying EXPLAIN to indicate conditional statements
- Lifting trigger and join index restrictions on Atomic Upsert

The following sections contain information about each of these enhancements.

NUSI INSERT Improvements

When inserting on an extremely NUSI value, the hash comprises a large number of rows. Most of the rows are near the maximum row size of 64 KB. The cost of scanning these rows on every NUSI insert significantly reduces the performance of insert operations on the table.

To avoid the cost of scanning, Teradata V2R5.0 contains a new "incremental scan" strategy that scans, at most, two NUSI rows per insert as follows:

- The last row in the hash for the index value, or
- One other row in the hash having a unique value that depends on the state of the last index row, including its own unique value and the number of RowIDs it contains.

If room remains for more RowIDs in the scan row, then the new RowID is added to the scan row rather than the last row. As long as the last row is not changed by NUSI deletes, successive inserts into the same NUSI value are added to the same scan row until it is filled.

The next insert goes into the last row, but that change increments the scan row selection for the following insert. As a result, the scan row slowly scans through the hash code so that partially filled NUSI rows for a given value are filled in between inserts to the last row for the value. The incremental scan of the hash detects gaps in the sequence of unique values and inserts new index rows to fill in the gaps.

EXPLAIN Statement

In Teradata V2R5.0, the EXPLAIN statement indicates which Merge and Upsert statements are conditional.

Trigger and Join Index Restrictions Lifted on Atomic Upsert

Teradata V2R5.0 enhances Atomic Upsert by lifting the semantic constraints on triggers and join and hash indexes on a target table. V2R5.0 does not, however, extend the syntax beyond what was implemented in V2R4.1, that is, a simple update with a primary index specified and a simple insert with constant or imported field values.

In order to support join and hash indexes and triggers for Atomic Upsert, Teradata V2R5.0 introduces a new conditional step-execution architecture in the Dispatcher. Conditional steps such as Update steps are now marked so that the Dispatcher knows whether to dispatch the Insert step based on a response from the processing of the Update step.

Performance Impact

Update performance features and enhancements provide the following performance benefits:

- With NUSI INSERT improvements, you can see performance improvements on row-at-a-time insert operations when applications make a large number of inserts into large tables with highly non-unique secondary index values.

  The following cases will benefit from the feature:
  - Inserts with a NUSI value qualifying as highly Non-Unique.
  - Updates that set a NUSI to a highly Non-Unique value

  The following limitations apply to the feature:
  - DELETE operations do not benefit, since the index rows must all be scanned until the deleted data RowID is found. The delete from the old NUSI value must use a complete scan of the index rows as with DELETE operations.
  - UPDATE operations only benefit for the insert into the new NUSI value.
  - An INSERT rollback is like a DELETE and, therefore, does not benefit.
  - A DELETE rollback, while like an INSERT, does not benefit because rollback has to check for a duplicate RowID.
  - Only simple NUSI indexes benefit: not Covered Indexes or Join Indexes.
  - INSERT-SELECT does not benefit since it no longer does NUSI maintenance with the inefficient row-at-a-time method, but instead uses a new Teradata V2R5.0 bulk index update service that improves performance significantly for all NUSI indexes regardless of their non-uniqueness.
- Improvements to Atomic Upsert result in an increase in Teradata TPump performance.
- When a large number of rows are deleted or inserted into a table with an NUSI or covered indexes, the index is updated a block at a time. The amount of performance gain depends on the number of rows per index block being updated.

When a large number of rows are inserted into a fallback table, the fallback message is no longer broken at the bucket boundary. It is broken when the buffer is full. This change can provide a performance gain for small- to mid-sized tables.

Teradata TPump and Join Index

Introduction

The Teradata TPump utility submits a large volume of update, insert, and delete statements concurrently. Each statement fully specifies the primary index of the affected tables.

Teradata TPump achieves concurrency by packing multiple statements into one request and submitting the requests across multiple sessions in parallel. Because each modification uses a row-hash lock and runs on a single AMP, Teradata TPump can take advantage of the parallelism inherent in the design of the Teradata database to achieve high throughput and scalability.

Teradata TPump and Join Index

When a join index is specified in a TPump statement, additional query processing is necessary to maintain the integrity of the join index. In some instances, join index maintenance places table-level read locks on other tables that are defined in the join index to materialize the join index rows, then places a table-level write lock on the join index table for update. Consequently, join index maintenance can affect Teradata TPump performance because maintenance can serialize the concurrent single-AMP update operations on base tables.

V2R5.0 Enhancements

Teradata V2R5.0 contains enhancements that minimize the impact of join index maintenance on the concurrency of Teradata TPump workload. The enhancements include:

- Reducing table-level locks incurred by join index maintenance to row-hash locks whenever possible
- Improving join index maintenance processing for some modification scenarios, "unmatching rows" maintenance, and aggregate join indexes
- Introducing new query processing steps for join index maintenance

The following sections contain information about each enhancement.

Enhancements to Join Index Maintenance

Before Teradata V2R5.0, for single-row base table modifications, when one row in the join index reflected one row in the base table, the system had to re-access all rows that made up the join index row, even if columns for the other tables were not being modified. These activities resulted in table-level locks and all-AMPs operations that affected the performance of Teradata TPump.

Teradata V2R5.0 provides the following enhancements to improve join index maintenance:

- Allows a column in the first field of a compressed join index to be modified directly as long as the update condition is only based on the columns in first field.
- Improves maintenance for a simple INSERT by generating single-AMP steps for the maintenance of join index with RowID.
- Enables the use of group-AMP steps by using a hash index to access other base tables when performing join index DELETE and UPDATE maintenance.
- Eliminates the NOT EXIST join from the maintenance plan when deleting from the inner table of an outer join index and the delete condition is specified on the outer join columns.
- Uses the Merge-Update step if the base table modification can be mapped to a direct update on the join index, but the update condition cannot be mapped to a cost effective access path to the join index. See "Merge-Update Step for Join Index Maintenance" on page 15-18.

Improvements to Aggregate Join Index Maintenance

Teradata V2R5.0 contains the following improvements for aggregate join index maintenance as shown in the following table:

WHEN...	THEN...
the delete condition is specified on the grouping columns of an aggregate join index	directly delete those aggregate rows that qualify the condition of the base table delete.
handling the aggregation step in the maintenance plan	eliminate the aggregation step in the INSERT and DELETE if eliminating the step is the best thing to do.

New Query Processing Steps for Join Index Maintenance

Teradata V2R5.0 contains improvements to join index maintenance that ensure that single-AMP and group-AMP steps and row-hash locks are used whenever possible. To facilitate this usage, release V2R5.0 introduces the following new query processing steps:

- Merge-Update
- Single-AMP Merge-Delete

The following sections contain information about each step.

Merge-Update Step for Join Index Maintenance

Before Teradata V2R5.0, join modifications were processed as a Delete followed by an Insert. V2R5.0 provides a new Merge-Update step that performs the equivalent of a Delete step and an Insert step using the materialized join index rows. Join modifications are done in one step replacing the Merge-Delete and Merge combination. This feature helps overcome the missing row problem in join updates.

To take advantage of the new step, the join modification must meet the following conditions:

IF the modification does not change...	THEN the join update...
the primary index of the join index	can take advantage of the Merge-Update step.
some of the columns of the join condition in the join index definition	
a column in the first field of a compressed join index	

Single-AMP Merge-Delete Step for Join Index Maintenance

Teradata V2R5.0 supports the use of a single-AMP Merge-Delete, Merge, Merge-Update step with row-hash locks when the primary index value of the modified rows is known.

Performance Impact

Using single-AMP and group-AMP query processing steps in conjunction with row-hash locks whenever possible can:

- Potentially increase the speed and throughput of Teradata TPump
- Improve scalability with growth

Optimizer Enhancements

Introduction

Teradata V2R5.0 includes the following enhancements to the Optimizer. It now:

- Recognizes cases that qualify for the new Merge-Update step
- Modifies cost coefficients

The following sections contain information about the enhancements.

Merge-Update Step

Teradata V2R5.0 contains enhancements that allow the Optimizer to recognize possible cases that qualify for the new Merge-Update step. Merge-Update is implemented in three matching modes:

- RowHash Match
- RowID Match
- WholeRow Match

The following table specifies some conditions that apply in the current implementation:

IF the modification does not update...	THEN the modification...
the primary index or unique secondary index of the target table	can take advantage of the Merge-Update step.
the columns of the target table that are part of the join index condition	
a target table that is an RI parent or child table	
a cursor as a Join Merge update	

For information about the Merge-Update step and join modification, see "Merge-Update Step for Join Index Maintenance" on page 15-18.

Optimizer Cost Coefficients

As the result of enhancements introduced in Teradata V2R5.0, the Optimizer can use CPU, I/O, and network cost constants that better reflect the fundamental relationships between processor and I/O costs.

Service time is the time needed to perform basic operations, for example accessing a row in memory expressed in instructions per row.

As I/O subsystems evolve, previous assumptions regarding I/O block sizes are no longer valid. Data block size has increased as has the number and variety of disk arrays. Teradata V2R5.0 extends the infrastructure to support additional I/O subsystem types and additional block sizes.

Performance Impact

Optimizer enhancements improve performance as follows:

- Including the Merge-Update step in the plan improves query performance involving tables without fallback. In this case, the rows are processed one-block-at-a-time for RowHash Match and RowID Match merge modes.
- Updating the Optimizer cost coefficients can result in more efficient Optimizer costing, enhanced query plans, and more accurate EXPLAIN estimates.

Parallel Dump (UNIX)

Introduction

Parallel Dump was implemented to allow faster dumping of data from processor memory by dumping to 2 disk drives in parallel, an improvement over the previous method of dumping to a single disk drive.

Note: Parallel Dump was implemented only on MP-RAS UNIX.

Performance Impact

By dumping to two disks (dump size less than 500 mb) in parallel, as opposed to one, there was an overall improvement in dump time of 45-50%.

When dump sizes are greater than 500mb and there is only 2GB of memory available on a node, I/O swapping occurs, since there is no room to write to memory. This additional overhead negates any performance improvement with the Parallel Dump feature.

Chapter 16:

Troubleshooting Database Performance

This chapter provides tips and techniques to help handle performance problems.

Topics include:

- Troubleshooting a hung or slow job
- Identifying peak periods and bottlenecks
- Job scheduling around peak utilization
- Expanding the system
- How to catch a killer query
- Running out of TDP memory
- Lock problems
- Lack of disk space
- Hardware faults
- Skewing

Troubleshooting a Hung or Slow Job

This section provides suggestions for troubleshooting a Teradata SQL job that either seems hung or is running more slowly than usual.

Although a hung and a slow job differ, for troubleshooting purposes, assume that they behave similarly. Perform troubleshooting in the following order:

Step	IF the Problem Is...	THEN...	See
1	Down hardware	Check the hardware status of your system to determine whether the BYNET or an AMP or PE is down, which could cause your job to appear hung or to run slowly. **Note:** This is an important troubleshooting step, whether you do it first or after examining for lock conflict.	next subsection
2	A blocked job	Use Teradata Manager to evaluate the data on application and utility locks that could be blocking a job.	page 16-3
3	A busy system	Determine: **a** the resource (AMP CPU, disk, BYNET, or PE) bottleneck. **b** who is using the bottlenecked resource and the amount of usage. **c** spool file size. **d** data distribution across the AMPs.	page 16-3

Troubleshooting Down Hardware

Use the following procedure to determine if your hardware is down:

Step	Action
1	**a** Use Teradata Manager or ResUsage data to determine whether the BYNET is up. **b** If not, fix the problem.
2	**a** Use Teradata Manager to determine whether all nodes are up. **b** If not, fix the problem. **Note:** Teradata Manager does not return information on down nodes. To determine if a node is down, compare the current node list against a previously known accurate node list.
3	**a** Use Teradata Manager to determine whether the AMPs and PEs are up. **b** If not, fix the problem.
4	If hardware is not the problem, go to "Troubleshooting a Blocked Job" on page 16-3.

Troubleshooting a Blocked Job

Use the following procedure to find and resolve a blocked job:

Step	Action
1	Use Teradata Manager to run the Lock Display utility and see whether a job is blocked and by whom.
2	If a job is blocked, you can run Query Session to determine how long the blocking transaction has been running.
3	To resolve the situation, you can abort the offending transaction, contact the user of the offending transaction, or wait until the block clears.
4	If a job is not blocked, go to the next subsection.

For example, DBA Joe Green is concerned about a job that is running slowly and is blocked by another job. Joe is uncertain about how important this job is in relation to the other applications running in the work environment. He does not know which of these jobs have greater priority.

Joe can perform any of the following:

- Use Teradata Manager on the job that is causing the block to get more information about that job; for example, disk I/O and CPU usage.
- Execute Query Session to determine how long the current transaction of the blocking job has been running, checking in particular the time of the last query.
- Telephone the person owning the blocking session and ask whether the job is critical; if not, ask the user to abort the job.
- For Multiple Virtual Storage (MVS) jobs, look at the MVS job log.

Troubleshooting a Busy System

Use the following procedure to determine if your system is busy:

Step	Action
1	Use Teradata Manager to determine how heavily the system components of AMP CPU, disks, or BYNET are being used. If your system is busy, go to step 3. If the system components of AMP CPU, disks, or BYNET are not busy, one of the following may have occurred: • Skewed resource utilization resulting from skewed data. A busy PE. Go to step 2.

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>2</td><td>Use Teradata Manager to determine PE usage for the hung or slow session.
<table>
<tr><th>IF the PE is...</th><th>THEN go to step...</th></tr>
<tr><td>busy</td><td>3</td></tr>
<tr><td>not busy</td><td>5</td></tr>
</table></td></tr>
<tr><td>3</td><td>Use Teradata Manager to determine which system resource is the bottleneck, who is using it, and the amount of usage.</td></tr>
<tr><td>4</td><td>After identifying the bottleneck, perform one of the following:
• Abort the heavy user.
• If you determine that no resource is causing the bottleneck and usage is valid (that is, no unauthorized heavy user), the system may be under-configured. Add nodes for more capacity.</td></tr>
<tr><td>5</td><td>Use Teradata Manager to determine if high and low disk and AMP CPU usage values are different.
<table>
<tr><th>IF high/low disk and AMP CPU values are...</th><th>THEN...</th></tr>
<tr><td>different</td><td>resource utilization is skewed. There may be a system problem in one of the AMPs or PEs or uneven data storage. Discuss with application users.</td></tr>
<tr><td>not different</td><td>there may be a client (that is, host) problem.</td></tr>
</table></td></tr>
</table>

Other Hints

You can use the following commands to help you determine the cause of a system hang:

Hint	See
Log onto BTEQ. If you can, try to select time or date.	*Teradata BTEQ Reference*
Run ShowLocks. If any HUT locks are active, release them with the RELEASE LOCKS command.	*Teradata RDBMS Utilities*
Run Lock Display and/or Query Session to see if queries are blocked by a long-running job holding a non-shareable lock.	*Teradata RDBMS Utilities*

Hint	See
Run Locking Logger and check for a locking contention. (For a record of error 2631 conditions, look in the view DBC.SW_Event_Log.)	*Teradata RDBMS Utilities*
Run SHOWSPACE. Verify that the system is not running out of disk space.	*Teradata RDBMS Utilities*
On UNIX: • Run schmon. • Run xperfstate. • Run xcpustate.	 • *Teradata RDBMS Utilities* (“Priority Scheduler Facility”) • *Teradata RDBMS Utilities* • *Teradata RDBMS Utilities*
Run vprocmanager to determine the state of the system. Is it in debugger mode?	*Teradata RDBMS Utilities*
Check the Teradata system logs (on UNIX, look in */var/adm/streams)* for mini-cylpack, deadlock, and memory paging messages, and so on.	• *Teradata RDBMS Messages* • *Teradata RDBMS Database Window*
Check ResUsage reports.	• Chapter 2 • *Teradata RDBMS Resource Usage Macros and Tables*
On UNIX, you can run puma -p to determine the number of tasks, mailboxes and monitors in use.	*Teradata RDBMS Utilities*

Identifying Peak Periods and Bottlenecks

Identifying the Peak Periods

Using Teradata Manager, ResUsage data can help you identify peak utilization periods. The Teradata Manager, with the Dynamic Utilization Charting (DUC) application and its near real-time scale, makes this particularly easy to do.

However, even without the Teradata Manager, you can run ResUsage figures for a complete day, several days, or an entire week to obtain a utilization profile over time. That will identify when the system is most heavily used.

Looking for the Bottleneck

Once you have identified the peak period, you can look for the bottleneck. Examining ResUsage data is one technique helpful in finding bottlenecks. After you locate the bottleneck, you can make informed decisions on how best to alleviate the problem.

Some possible bottlenecks are:

- CPU saturation or parallel efficiency
- Disk saturation
- Free and/or FSG cache memory
- BYNET
- Vprocs (system saturation, hot AMPs, load balancing, or parallel efficiency)
- Channel (number of sessions or channel speed)
- LAN/gateway (number of sessions or network connections)
- Workloads (multiple CPU- or disk-intensive applications that run simultaneously)

Run baseline profiles of major jobs to determine workload utilization of resources.

You can input collected information to plan better job scheduling, system upgrades, and expansions.

Job Scheduling Around Peak Utilization

Rescheduling Jobs

Once you determine your peak system utilization times, you can recommend that some jobs be moved to other time slots.

For example, if peak periods are 9 A.M. to 5 P.M., except for the noon to 2 P.M. lunch break, it might be wise to schedule batch and load jobs overnight to reserve peak daytime hours for DSS queries or HATP (High Availability Transaction Processing).

Bound Jobs

Since some jobs tend to be CPU-bound while others are I/O bound, it is a good idea to determine which jobs fit into which category.

Doing this makes you are able to recommended counter-scheduling a CPU-bound job with an I/O bound job so that the resource under-utilized by one job can be used more fully by the other.

Expanding the System

Introduction

To improve performance, you may need to consider expanding your system by adding disk arrays, memory, vprocs, or nodes.

Adding Disk

When you add disks and/or disk arrays, you increase system capacity. This is helpful in either a DSS and a transaction processing environment with a growing database or increasing concurrent disk I/O. To determine if the system needs more storage capacity, look at the ResUsageSvpr table for unusual disk activity such as frequent:

- Mini-cylpacks
- Defrags
- PACKDISKs

You may need to add more storage capacity to existing nodes when:

- Excessive disk activity is impacting performance
- Storage needs were underestimated
- Application changes require additional spool space
- Database growth requires additional storage

In the event that you need more storage capacity but not more CPU power, add storage without adding nodes.

When you add disk arrays, you must configure them and assign the storage to existing AMPs (join procedure) or add new AMPs (system reconfiguration). Contact your NCR support representative for details and outage estimates.

Note: Be aware of your memory requirements when adding AMPs to existing nodes.

Adding Memory

When you add memory, you increase the cache to maximize the capability of the CPUs. This is helpful when a system is I/O-bound on the disk (the CPUs are processing faster than the disk contents can be read into memory).

Add more memory for the following conditions:

Condition	Description
Paging/ swapping	When there is a lot of paging and swapping, more memory means that more code and data can be cached, causing decreased I/O for paging and swapping. It is recommend that you reduce free space.
Tables in memory	Increased memory may reduce I/O by allowing: • Tables too large to remain resident in memory to remain in memory • More small tables that can remain in memory and can reside concurrently in memory Unfortunately, there are no statistics at this time to help you detect this condition.
Add vprocs to nodes	When you add vprocs to nodes, you also may want to increase the memory. Each vproc consumes 32 MB of memory. Additional vprocs can substantially reduce free memory, which can cause more I/Os because the system can cache fewer data blocks.

To add memory to a node, bring the node down, install the memory, and then bring it back up. See your NCR support representative for details and outage estimates.

Adding Vprocs

Add vprocs for the following reasons:

Purpose	Description
Increase single-user response time	Optimization of single-user throughput or response time may be especially significant when there are fewer AMP vprocs/node than CPUs/node. In a concurrent workload, you may be able to add more AMPs /node and achieve the desired throughput.

Purpose	Description
Modify the configuration	When you add storage, you normally add AMPs. For example, if you add two disk cabinets to each of 20 ranks, you would add 10-20 more AMPs to the system configuration. The number of AMPs you add depends on the number of ranks assigned to the existing AMPs. If your nodes will not be able to handle the additional load caused by the vprocs, you must consider additional nodes.
Increase session capacity	When the system reaches concurrent session limits, add more PEs to handle additional sessions. Verify that there is enough CPU and memory capacity to handle more PEs. Each PE can handle 120 sessions with a maximum of 1200 sessions per node for gateway sessions. More than 10 PEs per node will not solve a session limit problem. The channel driver does not have a session limit, but one channel PE is normally configured for each channel connection. **Note:** When you add vprocs, more memory is required by UNIX to run a vproc, which reduces the amount of memory available for FSG cache (memory for caching data blocks) by 32 MB/added vproc.

Adding vprocs to a system requires a system reconfiguration (pdeconfig, config, reconfig). Adding AMPs requires a data redistribution. See your NCR support representative for details and outage estimates.

Adding Nodes

If you cannot satisfy increased capacity or throughput needs by adding disk arrays, memory, and/or vprocs, you can add nodes. Usually you add nodes because you need more disk storage capacity and more CPU power.

Consider the first options carefully, because adding new nodes costs more than adding disks and memory to an existing node. However, you must add nodes if the system is CPU bound.

For example, if the configuration is adequate from an I/O perspective and the CPU is at maximum capacity, adding nodes tends to alleviate the problem.

If you need more nodes, but not disk storage, determine if you should:

- Add nodes and disk storage to maintain the current ratio and find other uses for the excess storage such as fallback, join indexes, multi-temperate data such as longer historical retention.
- Add nodes and AMPs and redistribute the storage to reduce the amount of storage managed by each node.
- Add non-TPA nodes and move applications from your TPA nodes.

Consider the following:

IF you add…	THEN you must perform…
TPA nodes	a full system reconfiguration, from defining nodes to reconfiguring (redistributing) your data.
Non-TPA nodes	some system reconfiguration, but data will not be redistributed.

Contact your NCR support representative for details and outage estimates.

How to Catch a "Killer Query"

Introduction

Occasionally someone may launch a query that is so badly optimized that it generates a very bad execution plan.

For information on handling "killer queries," see "Job Mix Tuning" on page 7-13.

Interactive Methods for Catching Killer Queries

Using Teradata Manager PMON is the recommended tool for investigating bad queries. Here are the steps:

1. Receive notification of slowdown
2. View PMON session screen, sort by descending order, delta AMP CPU.
3. Click on the highest CPU user.
4. Scroll through the sessions, looking for
   - High total CPU, low I/O accesses,
   - High CPU skew
5. Click on the SQL for this user
6. Click on the EXPLAIN for this user
7. In the EXPLAIN, look at:
   - Look at how many steps and what type of operation (in particular, product join) is being performed.
   - Look to see if we are stuck on this step. If yes, this is likely the culprit. To be sure, clip out the SQL and paste it into a Queryman or BTEQ screen, and run a full EXPLAIN. If the step shows a product join with many, many rows, this will validate.
8. If you have a penalty box set up in Priority Scheduler, move this session to the penalty box using the modify screen from the session details screen, and change the account string, or consider aborting the session if it is only doing a SELECT.

If You Don't Have Teradata Manager . . .

If you are aware via user feedback or a canary (or heartbeat) trigger that the system is slowing down, there may still be time to run a few key queries to find the rogue query, before the system hangs:

- High CPU

Finds the users with high CPU relative to I/O accesses. In general, the profile for a bad query can be identified by looking at the ratio between the two. If the ratio of CPU divided by I/O is > 100, this is a sure candidate. In a product join, the row redistribution is done in memory. This gives a particularly bad product join (e.g., left table with millions of rows times the right table with millions of rows) the profile of high CPU user, because it:

  – is almost always ready to run
  – does no physical I/O,
  – generally uses up its entire time quantum when it is switched in.

- Skewed/hot Spool

  Finds the jobs/users with lumpy spool. If a plan is bad, it may often result in lumpy spool. While these type of queries will have a tendency to run out of spool space on one AMP before they can finish (or run forever, as in some cases), if users have very high individual spool space limits, this can cause all free spool on a single AMP to be exhausted for everyone else as well. In a case where CPU busy is very high or at maximum capacity, these hot spool situations will also cause overall degradation because the AMP is likely hot also.

- Active AMP Users

  A catchall script that for the active users, gets CPU Usage & Logical Disk I/Os & Skew by User with more than 10,000 CPU seconds, CPU or disk I/O skew greater than 100X the average. Uses the DBC.AMPUsage view.

Any of the above can be scripted to run at regular intervals and hooked to some sort of alert.

Sample Script: High CPU Use

```
bteq<<- [end]
/*================================================== */
/* HICPU */
/* The following SQL will give you a list of likely candidates to
investigate */
/* for over-consumption of CPU relative to disk I/O. In general,
we */
/* are only concerned with muliltple amp requests and requests of
long */
/* duration (cpu time > 10). We are using the ratio of: */
/* disk IO / CPU time < 100. Altermatively, you can use the ratio:
*/
/* (cpu * 1000 ms / io) > 10. The cut-off or "red flag" point will
be system */
/* dependent. For the 4700/5150, the ratio will be higher than for
the */
/* 4800/5200 which in turn will be higher than the 4850/5250. */
/*================================================== */
.logon systemfe,service
```

```
.export file=hicpu.out
sel ST.UserName (FORMAT 'X(10)', TITLE 'UserName')
,ST.AccountName (FORMAT 'X(10)', TITLE 'AccountName')
,ST.SessionNo (FORMAT '9(10)', TITLE 'Session')
,SUM(AC.CPU) (FORMAT 'ZZ,ZZZ,ZZZ,ZZ9.99', TITLE 'CPU//Seconds')
as cput
,SUM(AC.IO) (FORMAT 'ZZZ,ZZZ,ZZZ,ZZ9', TITLE 'Disk IO//Accesses')
as dio
,dio/(nullifzero(cput))
(FORMAT 'ZZZ.99999',TITLE 'Disk to//CPU ratio', NAMED d2c)
from DBC.SessionTbl ST
,DBC.Acctg AC
where ST.UserName = AC.UserName
and ST.AccountName = AC.AccountName
group by 1,2,3
having d2c < 100
and cput > 10
order by 6 asc;
.export reset
.quit;[end]
```

Sample Script: Hot / Skewed Spool in Use

```
/* HOT SPOOL - this query will sometimes identify hot AMP */ /*
problems. It is dependent on the running job having spool */ /*
(which may not always be true), and the user having select */ /*
rights on DBC *tables*... (you could convert this to DBC */


/* views). It's use is largely based on checking the count */ /*
of vprocs that are in use and comparisons of the avg, max, */ /*
and sum values of spool on the vprocs. Note the calculation*/ /*
of Standard Deviation (P) and the Number of Deviations. */ /* The
count of vprocs is simple, if it is less than the */ /* number of
vprocs on the system, then the query is not */ /* distributing to
all VPROCS. If the MAX is twice the AVG, */ /* then you have an
out of balance condition, since a single */ /* VPROC has twice the
spool of the AVG vproc... */
bteq <<- [end]
locking DBC.DataBaseSpace for access locking DBC.DBase for access
select DBase.databasename as UserDB, sqrt(((count(css) *
sum(css*css))- (sum(css)*sum(css)))/ (count(css)*count(css)))
(format 'zzz,zzz,zzz,zz9.99') as STDevP ,(maxspool - avgspool ) /
nullifzero(stdevp) (format 'zz9.99') as NumDev ,((maxspool -
avgspool) / maxspool * 100) (format 'zzz.9999') as PctMaxAvg
,count(*) (format 'zz9') as VCnt ,avg(css) (format
'zzz,zzz,zzz,zz9') as AvgSpool ,max(css) (format
'zzz,zzz,zzz,zz9') as MaxSpool ,sum(css) (format
'zzz,zzz,zzz,zz9') as SumSpool from DBC.Dbase , (select
DataBaseSpace.DatabaseId , DataBaseSpace.VProc ,
DataBaseSpace.CurrentSpoolSpace as css FROM DBC.DataBaseSpace
WHERE DataBaseSpace.CurrentSpoolSpace <> 0) DBS where
DBase.DatabaseID = DBS.DatabaseID group by UserDB;
.quit; [end]
```

Sample Script: Active AMP Users with Bad CPU/IO Access Ratios

```
/*************************************************************
*********/
/* DBC.AMPUSAGE - For the active users, */
/* get CPU Usage & Logical Disk I/Os & Skew by Users */
/* with more than 10,000 cpu seconds, */
/* cpu or disk io skew greater than 100X the average */
bteq <<- [end]
.logon systemfe,service
.export file activeampskew.out
.set defaults
.set width 130
LOCK DBC.Acctg for ACCESS
LOCK DBC.sessiontbl for ACCESS
SELECT
DATE
, TIME
, A.accountName (Format 'x(18)') (Title 'AMPusage//Acct Name
')
, A.username (Format 'x(12)') (Title 'User Name')
, DT.sessionNo (Format '9(10)')
, A.vproc (Format '99999') (Title 'Vproc')
, A.CPUTime (Format 'zz,zzz,zz9') (Title 'CPUtime')
, DT.AvgCPUTime (Format 'zz,zzz,zz9') (Title 'AvgCPUtime')
, A.CPUTime/NULLIFZERO(DT.AvgCPUTime)
(Format 'zzz9.99') (Title 'CPU//Skew')
(Named CpuRatio)
, A.DiskIO (Format 'zzz,zzz,zzz,zz9') (Title 'DiskIO')
, DT.avgDiskIO (Format 'zzz,zzz,zzz,zz9') (Title 'AvgDiskIO')
, A.DiskIO /NULLIFZERO(DT.avgDiskIO)
(Format 'zzz9.99') (Title 'Disk//Skew')
(Named DISKRatio)
FROM
DBC.AMPUsage A,
(SELECT
B.accountName
, C.sessionNo
, B.username
, AVG(B.CPUTime)
, SUM(B.CPUTime)
, AVG(B.DiskIO)
, SUM(B.DiskIO)
FROM
DBC.AMPUsage B, DBC.SessionInfo C
WHERE
B.accountname = C.accountName
GROUP BY 1, 2, 3
HAVING SUM(CPUTime) > 10000
```

```
) DT (accountName, sessionno, username, avgCPUtime,
sumCPUtime, avgDiskIO, sumDiskIO)
WHERE A.username = DT.username
AND A.accountname = DT.accountname
AND (CpuRatio > 100.00 OR DiskRatio > 100.00)
/* ADD THE FOLLOWING TO ZERO IN ON A GIVEN VPROC... */
/*AND VPROC in (243,244,251,252,259,260,267,268,275,276) */
ORDER BY 7, 1, 2, 3, 4, 5 ;
.export reset
.quit;[end]
```

Running Out of TDP Memory

Introduction

The symptoms for this problem can be as minimal as a set of mainframe jobs taking too long to blocking every user on accessright.

How to Verify

Use the TDP "DISPLAY CELLS" command

If causing MF sessions to block, this will appear in PMON

What To Do

Increase TDP memory cells. Cells can be increased dynamically without a TDP restart.

Lock Problems

Tabular Summary

The following table provides suggestions for analyzing and solving lock problems. For additional information, see Chapter 14: "Database Locks and Performance".

Tool	Analysis	Solution
Lock Display	Transaction locks	Determine which session is holding the lock that blocks others.
Teradata (PMON)	Blocked session	Abort the session causing blocked transactions.
Query Session	Blocked session	
ShowLocks	Host utility (HUT) locks; that is, locks placed by a client-based utility, such as Archive and Recovery	Submit RELEASE LOCKS as either an Archive and Recovery command or an SQL statement.

Lack of Disk Space

Tabular Summary

The following table provides suggestions for analyzing and solving lack of disk space. For additional information, see Chapter 9: "Managing Space".

Tool	Analysis	Solution
1 DBC.DiskSpace 2 Teradata Manager to run the error log analyzer.	The system is low on available permanent space.	• Run PACKDISK. • Set spool limits on users. • Add disk hardware.
	There is a very large spool.	• Cancel the job requiring the space. • Examine the structure of SQL queries for the cause of the large spool generation.
1 SHOWSPACE (Ferret) 2 DBC.Software_Event_Log to check the time and frequency of cylpacks.	The system has adequate disk space, but is out of free cylinders.	• Modify FreeSpacePercent (see page 5-12) to change the system default. • Use CREATE TABLE or ALTER TABLE to change a specific table-level value.

Hardware Faults

Tabular Summary

The following table provides suggestions for analyzing hardware faults.

Tool	Indication/Action
Teradata Manger (PMON)	Check for down resources.
UNIX	Check the */var/adm/streams* log.
DBC.Software_Event_Log	Check for hardware errors.

Skewing

Tabular Summary

The following table provides suggestions to help you determine:

Step	Problem	Action
1	If the system is CPU- or disk-bound, or the processing is skewed.	**a** Use Teradata Manager (PMON). Monitor physical resources to view CPU/disk usage by node. Monitor virtual resources to view CPU/disk usage by vproc. **b** Enter sar -u to view CPU usage. **c** Enter sar -d to view disk usage. **d** Look at DBC.ResusageSPMA to view CPU, disk, and BYNET usage. **e** Look at DBC.AMPUsage to find the user with the skewed processing. **f** Look at DBC.TableSize to identify tables with skewed data distribution.
2	If the processing is skewed, identify the user.	**a** Use Teradata Manager (PMON). Use Monitor Sessions and Identify. **b** Check DBC.SessionInfo or run Query Session to identify the current user. **c** Check DBC.LogOnOff to view historical users. **d** Check DBC.AMPUsage to identify the heavy resource user. **e** Check DBC.DiskSpace to identify large spool user.
3	If the processing is skewed, get information on the problematic query.	**a** To identify the query, ask the user or check DBC.AccessLog. **b** Run EXPLAIN on the query to determine if there is a bad join plan. Look for unexpected product joins. Verify that large tables are not joined before small tables. Look for estimates that are too high or too low?

Index

Numerics

2PC protocol 12–66

A

Access controls 1–8
Access Logging 1–8, 3–7
Account String Expansion. See ASE
Accounting tables, housekeeping 13–12
Aggregates on derived tables 12–40
Aggregates on views 12–38
AMP operations, performance and 15–8
AMPUsage
  ASE parameters and 3–5
  ResUsage and 3–2
AMPUsage view. See DBC.AMPUsage view
ASE 3–4
ASE parameters, AMPUsage and 3–5
Atomic Upsert, restrictions on 15–14

B

Baseline profile
  building 4–5
  performance metrics for 4–4
Baseline text suite, uses of 4–2
BETWEEN clause, and performance 12–23
blmstat utility 6–14
Block
  journal size 9–17
  maximum size 9–15
  minimum size 9–16
  splitting 9–16
Buddy backup 2–38
BYNET data, ResUsage 2–34

C

C2 security 3–7
CALENDAR system view 12–18
Canary queries 4–6
  production canaries 4–8
  system canaries 4–7
CASE expression 12–11
Client-specific monitoring tools 7–17
Clustering
  fallback and 13–16
  performance and 13–16
Collect rate, ResUsage 2–12
Collecting statistics, performance and 12–53
Columns, compressing 12–6
Concatenation, correlated subqueries and 12–10
Configuration utility, performance and 15–2
Constraints optimization 15–12
Correlated subqueries 12–8
  concatenation and 12–10
CREATE INDEX, LOCKING clause and 14–6
ctl utility 1–8, 6–15
Cylinder Read
  changing defaults of 10–12
  FSG Cache, viewing 10–10
  processing sequence 10–10
Cylinder splits
  FreeSpacePercent and 9–9
  PACKDISK and 9–9
Cylinders
  creating more space on 9–14
  data block allocation unit, minimum 9–16
  data block size, maximum 9–15
  data compression 9–15
  defragmentation 9–14
  freeing 9–10
    disk space, adding 9–13
    mini-cylpacks and 9–10
  freeing space on 9–4
  journal data block size 9–17
  running out of free 9–3
Cylinders Saved for PERM 5–4

D

Dashboard, Teradata Manager 11–12
Data
  collecting, guidelines for 1–7
  parallel efficiency 13–7
  skewing 13–4, 13–7, 13–8
  uneven
    Hash functions and 13–3
    identifying 13–3

Data compression 9–15
Data distribution
  aggregation and 13–7
  issue with respect to 13–2
  join processing and 13–7
  primary index and 13–8
  referential integrity and 13–7
Data protection
  clustering and fallback 13–16
  fallback option and 13–13
  nodes and disks, BYNET protection of 13–15
  redundant disk arrays 13–15
Database design, performance and 1–14
Database Initialization Program 1–9
Database Query Log. See DBQL
Database Window 1–8
Datablock size, performance impact of larger 5–27
DBC.Accessrights, housekeeping 13–10
DBC.AMPUsage view 1–8
DBC.AMPUsage view, ResUsage and 2–17
DBC.AMPUsage, Teradata Manager and 11–22
DBC.DiskSpace view 1–9
DBC.LogOnOff view 1–9
DBC.SessionInfo view 1–9
DBC.Software_Event_Log view 1–9
DBC.TableSize view 1–9
DBQL 3–9
DBSCacheCtrl 5–5, 5–6
DBSCacheThr 5–6, 5–42
dbscontrol utility 1–9
Deadlocks 14–5
  avoiding 14–9
  cause of 14–5
DeadLockTimeout 5–8
DefragLowCylProd 5–9
DEFRAGMENT utility 1–9
Defragmentation 9–14
  managing space and 9–3
Derived table optimization 12–42
DESFREE 2–19
DictionaryCacheSize 5–10
DIP. See Database Initialization Program
dipviews 1–9
DisableSyncScan 5–11
Disk arrays, redundant 13–15
Disk space
  lack of 16–19
  running out of 9–2
DISPLAY CELLS command, Teradata Manager and 11–26
DISPLAY IFP command, Teradata Manager and 11–26
DISPLAY POOL command, Teradata Manager and 11–27
DISPLAY SESSIONS command, Teradata Manager and 11–27
Dynamic Query Manager. See TDQM

E

EXPLAIN statement 1–10
  enhancements to 15–13
EXPLAIN, Optimizer and 12–69
Extended macros 2–18

F

Fallback option, data protection and 13–13
Fallback, clustering and 13–16
FreeSpacePercent 2–18, 5–12
  cylinder splits and 9–9
  determining value for 9–5
  operations disregarding 9–7
  operations honoring 9–6
  PACKDISK and 9–8
FSG Cache 10–3, 10–9
  Cylinder Read and 10–10
  space in 10–9
FSG Cache Percent 2–19, 10–7, 10–14
FSP. See FreeSpacePercent

G

Gateway Control utility
  DISPLAY GTW 7–17
  DISPLAY NETWORK 7–17
  DISPLAY SESSION 7–17
Gateway control utility
  Teradata Manager and 11–29
Gateway Global 7–17
Global temporary tables, performance and 12–48
GROUP BY, join optimization and 12–43

H

Hardware
  down, troubleshooting 16–2

faults, analyzing 16–20
Hash joins, and performance 12–26
Heartbeat queries. See Canary queries
Housekeeping
accounting tables 13–12
DBC.Accessrights and 13–10
environmental changes and 13–12
logging tables 13–11
HSI timestamp 7–18
HSI, Teradata Manager and 11–28
HTMemAlloc 5–13, 5–39, 5–40
HUTCNS utilities, Teradata Manager and 11–23

I

I/O bottlenecks, solutions to 10–7
IdCol Batch Size 5–14
Identity columns, performance and 12–52
Index Wizard. See Teradata Index Wizard
INSERT SELECT statements 12–19

J

Job control support tools, Teradata Manager and 11–29
Job scheduling, and peak utilization 16–7
Jobs
blocked, troubleshooting 16–3
bound, performance and 16–7
hung, troubleshooting 16–2
performance and 15–6
slow, troubleshooting 16–2
Join index
maintenance, enhancements to 15–16
NUSI and 12–36
performance and 12–31
Joins on derived tables 12–40
Joins on view 12–38
JournalDBSize 5–15

K

KILL command, Gateway Control utility 7–19
Killer query, catching 16–12

L

Large table/small table joins, performance and 12–46
Lock Display utility 1–10, 14–4
Lock Log table 5–16
Locking
CREATE INDEX and 14–6
deadlock 14–5
levels of 14–3
Lock Display utility 14–4
modes of 14–3
overview 14–2
requests and 14–10
rules for 14–12
transactions and 14–11
utilities, client, and 14–14
LOCKING ROW / NOWAIT 14–13
LockLogger 5–16
Locks, problems with 16–18
Locks, pseudo table, deadlock detection and 14–7
Locks. See Locking
Logging rate, ResUsage 2–12
Logging tables, housekeeping 13–11
LOGOFF command, TDP 7–19
LOGOFF command, Teradata Manager and 11–29
LOGOFF POOL command, TDP 7–19
LOGOFF POOL command, Teradata Manager and 11–29
Logon control 7–19
LOTSFREE 2–19

M

Macros
extended 2–18
ResUsage 2–9
MaxDecimal 5–17
MaxLoadTasks 5–18
MaxParseTreeSeg 5–19
Memory
adjusting for low available free 10–5
efficient of use 10–2
free 10–3, 10–5
FSG Cache 10–3
monitoring 10–16
ResUsage and 10–5
row redistribution and 10–6
running out of 16–17
shared 10–3
swapping, solutions to 10–7

Merge joins, and performance 12–25
MINFREE 2–19
Mini-cylinder packs, managing space and 9–3
Mini-cylpack, error codes and 9–13
MiniCylPackLowCylProd 5–9, 5–20
MultiTool. See Teradata MultiTool

N

NUSI
  blocksize and 12–28
  join index and 12–36
  performance and 12–27
NUSI INSERT, improvements to 15–13
NUSI. See also Secondary index

O

Operational Data Store workloads 15–8
Optimized DROP MACRO, performance and 15–7
Optimizer, enhancements to 15–19
Optimizer, EXPLAIN and 12–69
Ordered analytical functions 12–15
Outer joins, performance and 12–44

P

PACKDISK
  cylinder splits and 9–9
  FreeSpacePercent and 9–8
  others utilities and 9–8
PACKDISK utility 1–10
Parallel Dump, improvements to 15–21
Parallel Upgrade Tool, performance and 15–2
Partitioned primary index, performance and 12–50
Performance
  CPU saturation and 6–2
  database design and 1–14
  deadlocks and 1–3
  I/O wait and 6–3
  processing concurrency and 1–3
  resource bottlenecks and 1–3
  system components and 1–12
  system saturation and 1–3
  system upgrade and 1–16
  why monitor? 1–2
Performance Monitor 11–3
Performance Monitor Object, Teradata Manager and 11–20
Performance resources
  Database Initialization Program 1–9
  DBC.AMPUsage view 1–8
  DBC.DiskSpace view 1–9
  DBC.LogOnOff view 1–9
  DBC.SessionInfo view 1–9
  DBC.Software_Event_Log view 1–9
  DBC.TableSize view 1–9
  dipviews 1–9
  EXPLAIN statement 1–10
  stune file 1–11
  Teradata Manager 1–11
  TPCCONS 1–11
Performance tools
  Access controls 1–8
  Access Logging facility 1–8
  ctl utility 1–8
  Database Window 1–8
  dbscontrol utility 1–9
  DEFRAGMENT utility 1–9
  Lock Display utility 1–10
  PACKDISK utility 1–10
  Priority Scheduler 1–10
  qrysessn 1–10
  Recovery Manager 1–10
  ResUsage 1–10
  RSSmon utility 1–10
  sar utility 1–10
  Scan Disk 1–10
  ShowLocks utility 1–11
  SHOWSPACE utility 1–11
  Teradata MultiTool 1–11
  TOP 1–11
  vproc manager 1–11
  xctl utility 1–11
  xperfstate utility 1–11
Perm space, spool space and 9–18
Permanent Journal, data protection and 13–17
PermDBAllocUnit 5–22
PermDBSize 5–15, 5–22, 5–24
PJ. See Permanent Journal
PM/API, and performance 2–43
PPICacheThrP 5–28
Priority Scheduler 1–10
Priority scheduler 7–2
  usage considerations 7–6
Priority Scheduler Administrator 11–4
  Teradata Manager and 11–17
Production canary queries 4–8
Profiles, performance and 15–4

puma utility 6–16

Q

QCF 8–3
qrysessn 1–10
Query analysis
  resources 8–2
  tools 8–2
Query Capture Facility. See QCF
Query Configuration utility, Teradata Manager and 11–24
Query Session utility, Teradata Manager and 11–23
Query Session utility. See qrysessn

R

ReadAhead 5–29
ReadAheadCount 5–30
Reconfiguration utility, performance and 15–2
Reconfiguration, performance and 15–2
Recovery Manager 1–10
RedistBufSize 5–31
Redistribution row 10–6
Referential integrity, performance and 12–62
Res5100Cpu macro 2–18
Res5150Cpu macro 2–18
ResCPUByAMP macro 2–9, 2–25, 2–27
ResCpuByCpu macro 2–26, 2–28
ResCPUByNode macro 2–9, 2–25, 2–26
ResCPUByPE macro 2–9, 2–25, 2–26
ResCPUUsageByAMPView 2–8
ResCPUUsageByPEView 2–8
ResCylPack macro 2–18
ResGeneralInfoView 2–8
ResHostByLink macro 2–9, 2–20
ResHostHourTotal macro 2–20
ResHostTotal macro 2–20
ResIODayTotal macro 2–31
ResLDVByNode macro 2–9
ResMemMgmtByNode macro 2–9
ResNet macro 2–18
ResNETByNode macro 2–9
ResNetBySec macro 2–34
ResNetHourTotal macro 2–34
ResNetTotal 2–34
ResNode macro 2–9, 2–22, 2–31, 2–33, 2–34, 2–40
Resource Check Tools 6–21
ResPMA macro 2–19
ResPmaByNode macro 2–22, 2–31
ResPmaBySec macro 2–22, 2–31
ResPmaCpuDayTotal macro 2–22
ResPmaHourTotal macro 2–22, 2–31
ResPmaIODayTotal macro 2–34
ResPmaTotal macro 2–22, 2–31
ResShstGroupView 2–8
ResSldvGroupView 2–8
ResSvpr5100Cpu macro 2–26, 2–29
ResSvpr5150Cpu macro 2–26, 2–29
ResSvprRead macro 2–19
ResUsage 1–10
  AMPUsage and 3–2
  available free memory and 10–5
  BYNET data and 2–34
  collect rate 2–12
  components of 2–3
  CPU utilization and 2–22
  data collection 2–6
  DBC.AMPUsage view and 2–17
  disk space overhead 2–13
  disk utilization and 2–31
  extended macros and 2–18
  host traffic and 2–20
  logging rate 2–12
  logging, optimizing 2–14
  macros 2–9
  sar utility and 6–11
  tables 2–4
  Teradata Manager and 2–16, 11–21
  views 2–8
  xperfstate utility and 6–11
ResUsage tables
  ResUsageIpma 2–5
  ResUsageIvpr 2–5
  ResUsageScpu 2–5
  ResUsageShst 2–6
  ResUsageSldv 2–5
  ResUsageSobj 2–6
  ResUsageSpma 2–5
  ResUsageSvpr 2–5
ResUsageIpma 2–5
  overhead considerations 2–13
ResUsageIvpr 2–5
  overhead considerations 2–13
ResUsageScpu 2–5
  overhead considerations 2–13
ResUsageShst 2–6
  overhead considerations 2–13
ResUsageSldv 2–5
  overhead considerations 2–13

ResUsageSobj 2–6
ResUsageSpma 2–5
  overhead considerations 2–13
ResUsageSvpr 2–5
  overhead considerations 2–13
Roles, performance and 15–4
RollbackPriority 5–33
RollbackRSTransaction 5–34
RollForwardLock 5–35
RoundHalfwayMagUp 5–36
RSDeadLockInterval 5–37
RSG vprocs 5–37
RSSmon utility 1–10, 2–42

S

sar utility 1–10, 6–4
  ResUsage and 6–11
  xperfstate utility and 6–10, 6–11
Scan Disk 1–10
Secondary index
  index access and 12–28
  performance and 12–27
Session Mode 5–38
Session processing support tools 7–19
Sessions, performance and 15–6
SET SESSION ACCOUNT statement, performance and 7–8
ShowLocks utility 1–11
ShowLocks utility, Teradata Manager and 11–24
SHOWSPACE utility 1–11
SHOWSPACE utility, Teradata Manager and 11–25
SkewAllowance 5–13, 5–39
Skewing 13–4, 13–7, 13–8, 16–21
SMF 7–18
SMF, Teradata Manager and 11–28
Space Usage, Teradata Manager and 11–5
Space, managing
  defragmentation 9–3
  mini-cylinder packs 9–3
Sparse indexes, performance and 12–68
Spool space
  as trip wire 9–18
  increasing 9–18
  managing 9–18
  perm space and 9–18
SQL and performance
  2PC protocol 12–66
  aggregates on derived tables 12–40
  aggregates on view 12–38
  ALTER TABLE and data retrieval 12–2
  BETWEEN clause 12–23
  CASE expression 12–11
  collecting statistics 12–53
  columns, compressing 12–6
  concatenation and correlated subqueries 12–10
  correlated subqueries 12–8
  CREATE TABLE and data retrieval 12–2
  derived table optimization 12–42
  EXPLAIN and the Optimizer 12–69
  global temporary tables 12–48
  GROUP BY and join optimization 12–43
  hash joins 12–26
  identity column 12–52
  index access 12–28
  INSERT SELECT statements 12–19
  join index 12–31
  join index and NUSI, compared 12–36
  joins on derived tables 12–40
  joins on views 12–38
  large table/small table joins 12–46
  merge joins 12–25
  multiple aggregate distincts 12–17
  NUSI, using 12–27
  ordered analytical functions 12–15
  outer joins 12–44
  partitioned primary index 12–50
  random statified sampling 12–17
  referential integrity 12–62
  row redistribution, reducing 12–22
  secondary index 12–27
  sparse indexes 12–68
  updatable cursors 12–67
  volatile temporary tables 12–48
Statistics Wizard. See Teradata Statistics Wizard
StepsSegmentSize 5–41
stune file 1–11
SyncScanCacheThr 5–11, 5–42
System
  bottlenecks, identifying 16–6
  busy, troubleshooting 16–3
  disk space, lack of 16–19
  expanding, performance and 16–8
  memory, running out of 16–17
  peak periods, identifying 16–6
  peak periods, job scheduling and 16–7
System canary queries 4–7
System components, performance monitoring and 1–12
System conditions
  measuring
    changes in data access 1–6
    data growth 1–5

increase in active sessions 1–6
increase in system demand 1–6
resource utilization 1–5
response time 1–5
System Emulation Tool. See TSET
System Maintenance, Teradata Manager and 11–5
System upgrade, performance and 1–16

T

Target Level Emulation. See TLE
TargetLevelEmulation 5–44
tdnstat 7–17
TDP commands, Teradata Manager and 11–26
TDPUTCE 7–18
TDPUTCE, Teradata Manager and 11–28
TDQM 7–16
Teradata Administrator 11–5
Teradata Dynamic Query Manager. See TDQM
Teradata Index Wizard 8–7
Teradata Manager 1–11, 7–19
alerts, using 11–18
applications of 11–8
audit log, investigating 11–14
centralized alerts/event management 11–4
Dashboard of 11–12
DBC.AMPUsage and 11–22
defined 11–2
DISPLAY CELLS command and 11–26
DISPLAY IFP command and 11–26
DISPLAY POOL command and 11–27
DISPLAY SESSIONS command and 11–27
gateway control utility and 11–29
historical resource utilization, analyzing 11–16
HSI and 11–28
HUTCNS utilities and 11–23
job control support tools and 11–29
LOGOFF command and 11–29
LOGOFF POOL and 11–29
performance impact of 11–11
performance monitor and 11–3
Performance Monitor Object and 11–20
Priority Scheduler Administrator and 11–4, 11–17
Query Configuration and 11–24
Query Session and 11–23
real-time activity, monitoring 11–12
ResUsage and 11–21
ShowLocks and 11–24
SHOWSPACE and 11–25
SMF and 11–28
Space Usage and 11–5
system behavior, investigating 11–13
System Maintenance and 11–5
TDP commands and 11–26
TDPUTCE and 11–28
Teradata Administrator and 11–5
trend analysis 11–4
workload trends, analyzing 11–15
Teradata Manager, ResUsage and 2–16
Teradata MultiTool 1–11
Teradata Statistics Wizard 8–8
Teradata System Emulation Tool. See TSET
Teradata Visual EXPLAIN 8–5
TLE 8–4
TOP utility 1–11, 6–13
TPCCONS 1–11
TPump
join index and 15–16
performance and 15–16
Transaction rollback, and performance 14–16
TSET 8–6

U

UNION ALL set operator, performance and 15–12
Updatable cursors 12–67
Update performance, enhancements 15–13
Upgrade. See System upgrade

V

Value List compression 12–6
Views, ResUsage 2–8
Visual EXPLAIN. See Teradata Visual EXPLAIN
Volatile temporary tables, performance and 12–48
vproc manager 1–11
vprocs, RSG 5–37

W

WriteDBsToDisk 5–45

X

xctl utility 1–11, 6–15

xperfstate utility 1–11, 6–8
ResUsage and 6–11
sar utility and 6–10, 6–11