Oracle8*i* Parallel Server

概要および管理

リリース 8.1

ORACLE

Oracle8*i* Parallel Server 概要および管理 リリース 8.1

部品番号 A62772-1

第 1 版 1999 年 5 月

原本名：Oracle8*i* Parallel Server Concepts and Administration

原本部品番号：A67778-01

原本著者：Mark Bauer.

原本協力者：Wilson Chan, Andrew Holdsworth, Anjo Kolk, Rita Moran, Graham Wood, and Michael Zoll.

Copyright © 1999, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラムの使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当ソフトウェア（プログラム）のリバース・エンジニアリングは禁止されております。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

* オラクル社とは、Oracle Corporation（米国オラクル）または日本オラクル株式会社（日本オラクル）を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation（米国オラクル）およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Legend が適用されます。

Restricted Rights Legend

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication and disclosure of the Programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-14, Rights in Data -- General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

ドラフトのアルファ版およびベータ版ドキュメント

ドラフトのアルファ版およびベータ版ドキュメントはプレリリース状態のものです。これらのドキュメントは、オラクル社の機密かつ所有のドキュメントであり、デモおよび暫定使用のみを目的としたものです。タイプミスからデータの不正確さに至るまでのいくつかの誤りが存在することが考えられます。このドキュメントは予告なく変更する場合がありますが、当ソフトウェアを使用するハードウェアに限定するものではありません。オラクル社はプレリリースのドキュメントに対して、無謬性を保証しません。またそのドキュメントを使用したことによって損失および損害が発生した場合も一切責任を負いかねますのでご了承ください。

目次

はじめに xix

第 I 部　パラレル処理の基本

1　パラレル処理とパラレル・データベース

パラレル処理 1-1
パラレル処理の定義 1-2
パラレル処理における考慮事項 1-4
パラレル・システムの特徴 1-4
SMP と MPP のパラレル処理 1-5
統合運用のためのパラレル処理 1-5
パラレル・サーバー 1-6
パラレル処理の主な要素 1-6
スピードアップとスケールアップ : パラレル処理の目標 1-6
同期処理：重要な成功要素 1-9
ロック機能 1-10
メッセージ機能 1-11
パラレル処理の利点 1-11
スループットの改善 : スケールアップ 1-12
応答時間の改善 : スピードアップ 1-12
パラレル・データベースの利点 1-13
パフォーマンスの改善 1-13
高い可用性 1-13
柔軟性の向上 1-13
ユーザー数の増加 1-13

パラレル・サーバーの必要性 1-14
排他アクセス権限を持つ単一インスタンス 1-14
複数インスタンス・データベース・システム 1-15
分散データベース・システム 1-17
クライアント / サーバー・システム 1-20
パラレル実行の役割 1-21

2 パラレル処理の実現

4 つのレベルの拡張性 2-1
ハードウェアとネットワークの拡張性 2-2
オペレーティング・システムの拡張性 2-5
データベース管理システムの拡張性 2-5
アプリケーションの拡張性 2-6
パラレル処理が有効な状況 2-7
データ・ウェアハウジング・アプリケーション 2-7
異なるデータ・ブロックを更新するアプリケーション 2-7
フェイルオーバーと高い可用性 2-8
まとめ 2-8
パラレル処理が有効でない状況 2-9
効率的な処理の分割方法のガイドライン 2-10
概要 2-10
垂直方向の分割 2-10
水平方向の分割 2-12
一般的なパラレル処理の誤解 2-12

3 パラレル・ハードウェア・アーキテクチャ

概要 3-1
パラレル処理ハードウェア・システム 3-2
アプリケーション・プロファイル 3-2
ハードウェア要件およびオペレーティング・システム・ソフトウェア要件 3-3
高速なインターコネクト 3-3
グローバル・アクセス可能ディスク・サブシステムまたは共有ディスク・サブシステム 3-3
共有メモリー・システム 3-3
共有ディスク・システム 3-5
非共有システム 3-6

非共有システムの概要 3-6
大規模パラレル処理システム 3-7
非共有システムのまとめ 3-8
非共有 / 共有ディスク混合システム 3-9

第 II 部　Oracle Parallel Server の概要

4　Oracle でのパラレル処理のインプリメントの方法

パラレル・サーバーの有効化と無効化 4-1
同期 4-4
ブロック・レベル・ロック 4-4
行レベル・ロック 4-4
領域管理 4-5
システム変更番号 4-5
高パフォーマンス機能 4-6
高速コミット、グループ・コミットおよび遅延書込み 4-6
行ロックおよび複数バージョン読込みの一貫性 4-7
オンライン・バックアップおよびアーカイブ 4-7
キャッシュ・フュージョン 4-7
順序番号ジェネレータ 4-8
Lamport SCN 生成 4-8
空きリスト 4-8
空きリスト・グループ 4-9
ディスクの親和性 4-9
ジョブおよびインスタンスの親和性 4-10
透過的アプリケーション・フェイルオーバー 4-12
キャッシュの一貫性 4-13
パラレル・キャッシュ管理について 4-13
非 PCM キャッシュ管理について 4-17

5　Oracle Parallel Server 用の Oracle インスタンス・アーキテクチャ

概要 5-1
OPS の複数インスタンス・アーキテクチャの特性 5-4
システム・グローバル領域 5-5
バックグラウンド・プロセス 5-5

フォアグラウンドでのロックの取得 5-6
キャッシュ・フュージョン処理およびブロック・サーバー・プロセス 5-6
Oracle Parallel Server の構成ガイドライン 5-8

6 パラレル・サーバー用の Oracle データベース・アーキテクチャ

ファイル構造 6-1
制御ファイル 6-1
データ・ファイル 6-2
REDO ログ・ファイル 6-3
データ・ディクショナリ 6-6
順序ジェネレータ 6-6
CREATE SEQUENCE 文 6-6
CACHE オプション 6-6
ORDER オプション 6-7
ロールバック・セグメント 6-7
OPS のロールバック・セグメント 6-8
ロールバック・セグメントを制御するパラメータ 6-9
パブリックおよびプライベート・ロールバック・セグメント 6-9
インスタンスがロールバック・セグメントを取得する方法 6-10

7 ロック・メカニズムの概要

Oracle ロック・メカニズムの区別 7-1
概要 7-1
ローカル・ロック 7-2
インスタンス・ロック 7-4
LCK プロセス 7-6
LMON および LMD0 プロセス 7-7
ロックのコスト 7-7
Oracle のロック名 7-8
ロック名の形式 7-8
PCM ロック名 7-9
非 PCM ロック名 7-9
IDLM によるロック・メカニズムの調整 7-11
IDLM によるロック・モードの追跡 7-11
インスタンスによる IDLM リソースへのデータベース・リソースのマッピング 7-12

IDLM ロックとインスタンス・ロックの相互関係 7-13
IDLM はリソースに対して 1 インスタンスにつき 1 ロックを提供 7-15

8 統合分散ロック・マネージャ

統合分散ロック・マネージャとは 8-1
IDLM によるリソース・ロック要求の許可および調整 8-1
ロック要求の待機 8-2
非同期トラップ（AST）によるロック要求状態の通知 8-2
ロック要求の変換および許可 8-3
IDLM ロック・モード : リソース・アクセス権 8-6
IDLM 機能 8-8
分散アーキテクチャ 8-8
フォールト・トレラント 8-8
ロックのマスター化 8-9
デッドロックの検出 8-9
Lamport SCN 生成 8-9
グループ・ベースのロック 8-9
持続リソース 8-10
メモリー要件 8-10
MTS および XA に対するサポート 8-10
IDLM 統計を監視するビュー 8-11

9 パラレル・キャッシュ管理インスタンス・ロック

PCM ロックおよびその機能 9-1
PCM ロック 9-2
PCM ロックの割当ておよび解放 9-3
PCM ロックの動き 9-4
PCM ロックあたりのブロック数 9-6
ping: 更新の必要性の通知 9-8
データの分割による ping の回避 9-9
ロック・モードおよびバッファ状態 9-10
初期化パラメータによるブロックおよび PCM ロックの制御方法 9-12
GC_* 初期化パラメータ 9-12
データ・ブロックの取扱い 9-14
PCM ロックの 2 つの方法 : 固定および解放可能なロック 9-14

IDLM ロック要素および PCM ロック 9-14
PCM ロックあたりのブロック数 9-16
ファイン・グレイン・ロックの動き 9-18
ロック要素の動き 9-19
解放可能ロックのパフォーマンスへの影響 9-19
異なるファイルへのファイン・グレイン・ロックおよびハッシュ・ロックの適用 9-20
Oracle によるブロックへのロックの割当て 9-20
ファイル対ロックのマッピング 9-20
ブロック・クラスあたりのロック数 9-22
ロック要素番号 9-22
例 : PCM ロックへのブロックのマッピング 9-23
GC_FILES_TO_LOCKS のハッシュ・ロック設定 9-23
GC_FILES_TO_LOCKS のハッシュ・ロック設定のその他の例 9-25
GC_FILES_TO_LOCKS の DBA ロック設定の例 9-27

10 非 PCM インスタンス・ロック

概要 10-1
トランザクション・ロック（TX） 10-3
表ロック（TM） 10-3
システム変更番号（SC） 10-4
ライブラリ・キャッシュ・ロック（N[A-Z]） 10-4
ディクショナリ・キャッシュ・ロック（Q[A-Z]） 10-5
データベース・マウント・ロック（DM） 10-5

11 領域管理および空きリスト・グループ

Oracle での空き領域の取扱い方法 11-1
概要 11-2
データベース記憶構造 11-2
空き領域管理のための構造体 11-4
例 : 空きリスト・グループ 11-8
空き領域の管理のための SQL オプション 11-11
複数インスタンスでの空き領域の管理 11-11
空き領域の分割による複数の空きリストの作成 11-11
空きリスト・グループによるデータの分割 11-12
インスタンスへの空きリストおよび空きリスト・グループの割当て方法 11-13

インスタンス、ユーザーおよびロックに関連付けられた空きリスト 11-14
インスタンスと空きリストの関連付け 11-14
ユーザー・プロセスと空きリストの関連付け 11-15
PCM ロックと空きリストの関連付け 11-16
エクステントの割当ての制御 11-17
新しいエクステントの自動割当て 11-17
新しいエクステントの事前割当て 11-17
セグメントの高水位標の移動 11-18

12 アプリケーションの分析

分析に必要な詳細度 12-1
アプリケーション・プロファイル 12-2
アプリケーションの機能および表アクセス・パターンの分析 12-2
読込み専用表 12-2
ランダム SELECT および UPDATE 表 12-3
INSERT、UPDATE または DELETE 表 12-4
実装の計画 12-5
分割のガイドライン 12-5
概要 12-5
アプリケーションの分割 12-6
データの分割 12-7

第 III 部 Oracle Parallel Server の開発手順

13 パラレル・サーバー用のデータベース設計

概要 13-1
ケース・スタディ : 初期のデータベース設計から OPS へ 13-2
"Eddie Bean" カタログ販売 13-2
表 13-3
ユーザー 13-3
アプリケーション・プロファイル 13-3
表へのアクセスの分析 13-4
表アクセス分析ワークシート 13-4
ケース・スタディ : 表アクセスの分析 13-8
ユーザー別のトランザクション量の分析 13-9

トランザクション量分析ワークシート 13-9
ケース・スタディ : トランザクション量の分析 13-10
ユーザーおよびデータの分割 13-13
ケース・スタディ : 初期の分割計画 13-13
ケース・スタディ : 詳細な分割計画 13-14
索引の分割 13-16
ハッシュ・ロックとファイン・グレイン・ロックの実装 13-16
設計のインプリメントおよびチューニング 13-17

14 複数インスタンス用のデータベースおよびオブジェクトの作成

複数インスタンス環境用のデータベースの作成 14-1
作業の概略 14-1
データベース作成用の初期化パラメータの設定 14-2
データベースの作成および起動 14-2
CREATE DATABASE オプションの設定 14-3
複数インスタンスをサポートするデータベース・オブジェクトの作成 14-4
追加のロールバック・セグメントの作成 14-4
OPS のオンライン REDO ログの構成 14-7
追加データファイル用のロックの提供 14-9
CREATE DATABASE オプションの値の変更 14-10

15 PCM インスタンス・ロックの割当て

PCM ロックの使用およびメンテナンスの計画 15-2
インスタンス・ロックの計画およびメンテナンス 15-2
PCM ロックの割当て 15-2
データ・ファイルおよびデータ・ブロックの検査 15-3
ワークシートの使用による必要な PCM ロックの分析 15-4
データ・ブロックへの固定 PCM ロックのマッピング 15-5
インスタンス間での PCM ロックの分割 15-5
GC_FILES_TO_LOCKS の設定 : 各データ・ファイルの PCM ロック数 15-6
GC_FILES_TO_LOCKS の構文 15-6
固定ロックの例 15-8
解放可能ロックの例 15-8
ガイドライン 15-9
GC_FILES_TO_LOCKS の設定のヒント 15-10

成長の余地 15-10
有効なロック数のチェック 15-11
有効なロック割当てのチェック 15-11
表領域を読込み専用に設定 15-11
ファイルの妥当性のチェック 15-12
パラメータ値を変更しないデータ・ファイルの追加 15-12
その他の GC_* パラメータの設定 15-12
GC_RELEASABLE_LOCKS の設定 15-13
GC_ROLLBACK_LOCKS の設定 15-13
PCM ロックのチューニング 15-14
false ping の検出 15-14
PCM ロック変換の所要時間 15-16
PCM ロック変換の完了を待っているセッションの判別 15-17
必要な PCM ロック数とリソースの合計 15-17

16 リソースおよびロック用の IDLM 容量の確保

概要 16-1
IDLM 容量の計画 16-2
リソースおよびロックの動的割当ての回避 16-2
必要なロックおよびリソースの計算 16-2
リソース使用率の監視 16-3
非 PCM リソース数の計算 16-4
Oracle 初期化パラメータの調整 16-6
表ロック数の最小化によるパフォーマンスの最適化 16-6
DML_LOCKS を 0（ゼロ）に設定 16-7
表ロックの使用禁止 16-7

17 空きリスト・グループによるデータ分割

概要 17-2
データベース・オブジェクト用の空き領域の分割方法の決定 17-2
データベース・オブジェクト特性 17-2
空き領域ワークシート 17-5
CREATE 文の FREELISTS および FREELIST GROUPS の設定 17-6
FREELISTS オプション 17-6
FREELIST GROUPS オプション 17-6

クラスタ用の空きリストの作成 17-7
索引用の空きリストの作成 17-8
インスタンス、ユーザーおよびロックと空きリスト・グループとの関連付け 17-9
インスタンスと空きリスト・グループとの関連付け 17-9
ユーザー・プロセスと空きリスト・グループとの関連付け 17-10
PCM ロックと空きリスト・グループの関連付け 17-10
エクステントの事前割当て（オプション） 17-10
ALLOCATE EXTENT オプション 17-10
MAXEXTENTS、MINEXTENTS および INITIAL パラメータの設定 17-12
INSTANCE_NUMBER パラメータの設定 17-12
エクステントの事前割当ての例 17-13
エクステントの動的割当て 17-14
ブロック・データベース・アドレスからロック名への変換 17-14
ALLOCATE EXTENT 構文での !*blocks* 17-14
未使用領域の識別および割当て解除 17-15
未使用領域の判別方法 17-15
未使用領域の割当て解除 17-15
削除および更新により使用可能となった領域 17-15

第 IV 部　Oracle Parallel Server システムのメンテナンス手順

18　複数インスタンスの管理

概要 18-2
Oracle Parallel Server Management 18-2
パラメータ・ファイルによる複数インスタンスの定義方法 18-3
複数インスタンスの共通パラメータ・ファイルの使用方法 18-3
複数インスタンスの個別パラメータ・ファイルの使用方法 18-4
IFILE によるパラメータ・ファイルの埋込み方法 18-5
PFILE による非デフォルト・パラメータ・ファイルの指定方法 18-8
複数インスタンス用の初期化パラメータの設定 18-8
GC_* グローバル・キャッシュ・パラメータ 18-8
複数インスタンスのパラメータに関する注意 18-9
すべてのインスタンスで同じ値にする必要があるパラメータ 18-10
必要なロックの量の決定および LM_* パラメータの設定 18-11
複数インスタンスのデータベース・オブジェクトの作成方法 18-11

インスタンスの起動 18-12
パラレル・サーバーの使用可能化およびインスタンスの起動 18-12
OPS を使用禁止にした状態での起動 18-13
共有モードでの起動 18-14
インスタンスの指定 18-16
現行インスタンスとデフォルト・インスタンスの区別 18-16
インスタンスへの SQL 文の適用方法 18-16
インスタンスへの Server Manager コマンドの適用方法 18-17
Cluster Manager 18-20
OPS のクラスタ管理 18-20
インスタンス・グループの指定方法 18-21
複数インスタンス上でのユーザー認証のためのパスワード・ファイルの使用方法 18-23
インスタンスの停止 18-24
パラレル問合せのインスタンスの制限方法 18-25
PARALLEL_SERVER_INSTANCES 18-26
インスタンス登録およびクライアント / サービス接続 18-26
クライアントからサービスへのアクセス方法 18-26
クライアント対サービス接続の構成 18-28
データベース・インスタンス登録 18-28
接続時フェイルオーバー 18-29
クライアント・ロード・バランシング 18-29
接続ロード・バランシング 18-29
パラレル実行のロード・バランシング 18-30
Managed Standby およびスタンバイ・データベース 18-30

19 チューニングによるパフォーマンスの最適化

一般ガイドライン 19-1
概要 19-2
全インスタンスの統計の保管 19-2
保管する統計 19-2
一度に 1 つのパラメータを変更する 19-3
競合 19-3
ロック変換の検出 19-3
アプリケーション内のロック競合位置の特定 19-4
高い可用性を得るためのチューニング 19-7

エラーの検出 19-8
IDLM ロックの回復および再マスター化 19-8
障害インスタンスの回復 19-8

20 キャッシュ・フュージョンとインスタンス間パフォーマンス

キャッシュ一貫性矛盾を解決するときのキャッシュ・フュージョンの役割 20-2
キャッシュ・フュージョンが一貫性読込みブロックを作成する方法 20-2
write/write 競合の解消を改善するためのデータの分割 20-4
キャッシュ・フュージョンによる拡張性の向上 20-4
コンテキスト切替えと CPU 使用率の減少 20-5
ユーザー・モード IPC による CPU 使用率の減少 20-5
ブロックの ping のための I/O の削減と X-to-S ロック変換の削減 20-6
高速インターコネクトを介した一貫性読込みブロック転送 20-6
OPS 用のインターコネクトおよびインターコネクト・プロトコル 20-6
インターコネクト処理への影響 20-6
サポートしているインターコネクト・ソフトウェア 20-7
パフォーマンス期待値 20-7
キャッシュ・フュージョンのブロック要求の待ち時間 20-8
キャッシュ・フュージョンとインスタンス間パフォーマンスの監視 20-8
キャッシュ・フュージョンと OPS のパフォーマンス監視の目標 20-8
OPS での待ち時間の統計 20-8
OPS とキャッシュ・フュージョンを監視するための統計情報 20-9
CATPARR.SQL による OPS のデータ・ディクショナリ・ビューの作成 20-10
グローバル動的パフォーマンス・ビュー 20-10
グローバル・キャッシュとキャッシュ・フュージョンの統計情報の分析 20-12
グローバル・ロックの統計情報の分析 20-15
IDLM リソース、ロック、メッセージおよびメモリー・リソースの統計情報の分析 20-17
IDLM メッセージの統計情報 20-19
OPS の I/O 統計情報の分析 20-21
ロック変換のタイプ別分析 20-22
ラッチ、OPS および IDLM 関連の統計情報の分析 20-24
V$SYSTEM_EVENTS を使用してパフォーマンスの問題を識別する方法 20-26
V$SYSTEM_EVENTS の明確な OPS 関連イベント 20-26
まとめ 20-27

21 データベースのバックアップ

バックアップ方法の選択 21-2
REDO ログ・ファイルのアーカイブ 21-2
アーカイブ・モード 21-3
自動アーカイブと手動アーカイブ 21-3
アーカイブ・ファイルのフォーマットとアーカイブ先 21-5
制御ファイルの REDO ログ履歴 21-6
アーカイブ・ログのバックアップ 21-7
チェックポイントおよびログ・スイッチ 21-9
チェックポイント 21-9
チェックポイントの強制 21-9
ログ・スイッチの強制 21-10
クローズ状態のスレッドに対するログ・スイッチの強制 21-10
データベースのバックアップ 21-11
オープン・データベース・バックアップとクローズド・データベース・バックアップ 21-11
Recovery Manager でのバックアップ 21-12
オペレーティング・システムでのバックアップ 21-14

22 データベースの回復

概要 22-1
インスタンス障害の回復 22-2
単一ノード障害 22-2
複数ノード障害 22-3
ファースト・スタート・チェックポイント処理 22-3
ファースト・スタート・ロールバック 22-3
インスタンス回復のためのデータ・ファイルへのアクセス 22-4
インスタンス回復のためのデータベースの凍結 22-4
Oracle インスタンス回復のフェーズ 22-5
メディア障害の回復 22-6
完全メディア回復 22-7
不完全メディア回復 22-8
REDO ログ・ファイルの復元と回復 22-9
災害時回復 22-10
パラレル回復 22-14
Recovery Manager を使用したパラレル回復 22-14

オペレーティング・システム・ユーティリティを使用したパラレル回復 22-14
OPS でのファースト・スタート・パラレル・ロールバック 22-15
Managed Standby およびスタンバイ・データベース 22-15

23　単一インスタンスからパラレル・サーバーへの移行

概要 23-1
変換の決定 23-2
変換する理由 23-2
変換しない理由 23-2
変換の準備 23-2
ハードウェア要件とソフトウェア要件 23-3
単一インスタンスから複数インスタンスへのアプリケーションの変換 23-3
管理上の問題 23-3
単一インスタンスから複数インスタンスへのデータベースの変換 23-4
変換のトラブルシューティング 23-9
変換後のデータベース回復 23-9
ロールバック・セグメント表領域の損失 23-9
パラメータ・ファイルの不適切な NFS マウント 23-9

第 V 部　参照情報

A　以前のバージョンとの相違点

8.0.4 と 8.1 の相違点 A-1
キャッシュ・フュージョンによるアーキテクチャの変更 A-1
新しいビュー A-2
GMS の削除 A-2
パラレル・トランザクション回復から " ファースト・スタート・パラレル・ロールバック " へ A-2
インスタンス登録の変更 A-3
リスナーのロード・バランシング A-3
診断プログラムの拡張 A-4
Oracle Parallel Server Management（OPSM） A-4
パラレル・サーバーのインストールとデータベース構成 A-4
ジョブとインスタンスの親和性 A-4
廃止されたパラメータ A-5
リリース 8.0.3 とリリース 8.0.4 の相違点 A-5

新しい初期化パラメータ A-5
廃止された初期化パラメータ A-5
廃止された起動パラメータ A-5
動的パフォーマンス・ビュー A-6
グループ・メンバーシップ・サービス A-6
リリース 7.3 とリリース 8.0.3 の相違点 A-6
新しい初期化パラメータ A-6
廃止された GC_* パラメータ A-6
変更された GC_* パラメータ A-7
動的パフォーマンス・ビュー A-7
グローバル動的パフォーマンス・ビュー A-7
統合分散ロック・マネージャ A-7
インスタンス・グループ A-8
グループ・メンバーシップ・サービス A-8
ファイン・グレイン・ロック A-8
クライアント側のアプリケーション・フェイルオーバー A-8
Recovery Manager A-8
リリース 7.2 とリリース 7.3 の相違点 A-9
初期化パラメータ A-9
データ・ディクショナリ・ビュー A-9
動的パフォーマンス・ビュー A-9
空きリスト・グループ A-9
ファイン・グレイン・ロック A-9
インスタンス登録 A-10
ソートの向上 A-10
XA パフォーマンスの向上 A-11
XA 回復機能の強化 A-11
遅延トランザクション回復 A-12
接続時のロード・バランシング A-13
ソート処理におけるキャッシュのバイパス A-14
遅延ロギング・ブロック・クリーンアウト A-14
パラレル問合せプロセッサの親和性 A-15
リリース 7.1 とリリース 7.2 の相違点 A-15
不要になった領域の事前割当て A-15
データ・ディクショナリ・ビュー A-15

動的パフォーマンス・ビュー A-16
空きリスト・グループ A-16
表ロック A-16
ロック・プロセス A-16
リリース 7.0 とリリース 7.1 の相違点 A-17
初期化パラメータ A-17
動的パフォーマンス・ビュー A-17
バージョン 6 とリリース 7.0 の相違点 A-17
バージョン間の互換性 A-17
ファイル操作 A-17
遅延ロールバック・セグメント A-19
REDO ログ A-19
空き領域リスト A-20
SQL*DBA A-20
初期化パラメータ A-21
アーカイブ A-21
メディア回復 A-22

B 制約

互換性 B-1
Export ユーティリティと Import ユーティリティ B-1
共有モードと排他モード間の互換性 B-2
制約 B-2
1 回に割り当てられる最大ブロック数 B-2
共有モードでの制約 B-2

索引

はじめに

このマニュアルでは、Oracle Parallel Server（OPS）について説明し、『Oracle8*i* 管理者ガイド』および『Oracle8*i* 概要』の内容を補足します。

このマニュアルで説明するパラレル処理の概念と手順を読むことによって、パラレル処理を実現するための知識を習得できます。このマニュアルの情報は、あらゆるオペレーティング・システムで稼働している OPS に適用します。

注意：『Oracle8*i* Parallel Server 概要および管理』では、Oracle8*i* および Oracle8*i* Enterprise Edition の各製品について、特徴および機能を説明しています。Oracle8*i* と Oracle8*i* Enterprise Edition の基本機能は同じです。ただし、Oracle8*i* Enterprise Edition のみで利用できる拡張機能がいくつかあります。これらの拡張機能の一部はオプションになっています。たとえば、クライアント・アプリケーション・フェイルオーバー機能を使用するには、Enterprise Edition と Parallel Server Option が必要です。

対象読者

このマニュアルは、Oracle Parallel Server を使用して作業を行うデータベース管理者およびアプリケーション開発者を対象にしています。

このマニュアルの構成

第 I 部 : パラレル処理の基本

第 1 章「パラレル処理とパラレル・データベース」	この章では、パラレル処理とパラレル・データベースのテクノロジについて説明します。このテクノロジにより、オンライン・トランザクション処理および意思決定支援アプリケーションの効率が大幅に改善されます。
第 2 章「パラレル処理の実現」	この章では、パラレル処理およびパラレル・データベースのテクノロジを効果的に取り入れることにより、処理速度と拡張性を改善する方法について説明します。
第 3 章「パラレル・ハードウェア・アーキテクチャ」	この章では、パラレル処理を実行できる各種ハードウェア・システムについて説明し、それぞれの長所と短所を概説します。

第 II 部 : Oracle Parallel Server の概要

第 4 章「Oracle でのパラレル処理のインプリメントの方法」	この章では、OPS がハイ・パフォーマンスのパラレル処理を実現する方法について説明します。
第 5 章「Oracle Parallel Server 用の Oracle インスタンス・アーキテクチャ」	この章では、排他モードで稼働する Oracle サーバーとは異なる、Oracle 複数インスタンス・アーキテクチャの機能について説明します。
第 6 章「パラレル・サーバー用の Oracle データベース・アーキテクチャ」	この章では、OPS の複数インスタンスに関連する Oracle データベース・アーキテクチャの機能について説明します。
第 7 章「ロック・メカニズムの概要」	この章では、OPS 内部のロック・メカニズムの概要について説明します。
第 8 章「統合分散ロック・マネージャ」	この章では、OPS のリソースに対するアクセスを制御するうえでの統合分散ロック・マネージャの役割について説明します。
第 9 章「パラレル・キャッシュ管理インスタンス・ロック」	この章では、PCM ロックの概念について簡単に説明します。PCM ロックの計画と割当ては、Oracle Parallel Server のデータベース管理者が経験する最も複雑な作業の 1 つです。

第 10 章「非 PCM インスタンス・ロック」	この章では、最も一般的な非 PCM インスタンス・ロックについて説明します。
第 11 章「領域管理および空きリスト・グループ」	この章では、領域管理の概念について説明します。
第 12 章「アプリケーションの分析」	この章では、OPS アプリケーション設計を最適化する場合の概念的なフレームワークについて説明します。

第 III 部 : OPS システムの開発手順

第 13 章「パラレル・サーバー用のデータベース設計」	この章では、OPS 用に最適化したシステムを設計するための一般的な方法について説明します。
第 14 章「複数インスタンス用のデータベースおよびオブジェクトの作成」	この章では、データベースを作成するときの OPS 固有の事項について説明します。
第 15 章「PCM インスタンス・ロックの割当て」	この章では、インスタンスの初期化ファイルのパラメータに値を指定してデータ・ファイルに PCM ロックを割り当てる方法について説明します。
第 16 章「リソースおよびロック用の IDLM 容量の確保」	この章では、システムで必要なすべてのロックとリソース用に、統合分散ロック・マネージャ内で十分なスペースを使用可能にすることで、共有リソースに対する競合を減らし、OPS の最大限のパフォーマンスを実現する方法について説明します。
第 17 章「空きリスト・グループによるデータ分割」	この章では、空きリストおよび空きリスト・グループを割り当ててデータを分割する方法について説明します。データを分割することによって、複数インスタンスを使用するときに生じる空き領域獲得のための競合を最小限に抑えることができます。

第 IV 部 : OPS システムのメンテナンス手順

第 18 章「複数インスタンスの管理」	この章では、OPS のインスタンスを管理する方法について説明します。
第 19 章「チューニングによるパフォーマンスの最適化」	この章では、チューニングの概要について説明します。
第 20 章「キャッシュ・フュージョンとインスタンス間パフォーマンス」	この章では、キャッシュ・フュージョンの詳細について説明し、キャッシュ・フュージョンおよびインスタンス間のパフォーマンスを監視する方法を説明します。

第 21 章「データベースのバックアップ」	この章では、オンライン REDO ログ・ファイルをアーカイブし、データファイル、データベースの制御ファイル、インスタンスのパラメータ・ファイルのバックアップを定期的に作成することにより、データを保護する方法について説明します。
第 22 章「データベースの回復」	この章では、パラレル・サーバーでの Oracle の回復機能について説明します。
第 23 章「単一インスタンスからパラレル・サーバーへの移行」	この章では、データベース変換、つまり、パラレル・サーバー・オプションを使用して単一インスタンスの Oracle データベースから複数インスタンスの Oracle データベースに変換する方法について説明します。

第 V 部 : 参照情報

付録 A「以前のバージョンとの相違点」	この付録では、Oracle の旧リリースと現行リリースの、OPS に関する相違点について説明します。
付録 B「制約」	この付録では、OPS の制約について説明します。

関連マニュアル

このマニュアルでは、読者がすでに『Oracle8*i* 概要』と『Oracle8*i* 管理者ガイド』を読んでいることを想定しています。

このマニュアルで使用する表記規則

このマニュアルで使われている規則は、次のとおりです。

- 本文
- 構文図および表記法
- コード例

本文

本文で使われている規則は、次のとおりです。

大文字

大文字は、コマンド・キーワード、オブジェクト名、パラメータ、ファイル名などを示す場合に使用します。

次のようになります。「プライベート・ロールバック・セグメントを作成する場合は、パラメータ・ファイルの ROLLBACK_SEGMENTS パラメータにそのセグメント名を指定してください。」

かぎカッコ

かぎカッコは、マニュアル・タイトルまたは強調する言葉を示します。

構文図および表記

構文図および表記法は、SQL コマンドの構文、種々の機能、ヒントおよびその他の要素を示します。構文図および例の読み方、これらに基づいた SQL 文の書き方は、次のとおりです。

キーワード

キーワードは、SQL 言語で特別な意味を持った言葉です。このマニュアルの構文図で、キーワードは大文字で示してあります。SQL 文でキーワードを使用するときは、構文図に示されたとおりに入力する必要があります。ただし、大 / 小文字の区別は問いません。たとえば、CREATE TABLE 文を入力するときは、文頭でキーワード CREATE を使用し、CREATE TABLE 構文図に示されたとおりに入力する必要があります。

パラメータ

パラメータは、構文図でプレースホルダの役割を果たします。パラメータは、小文字で示してあります。パラメータは通常、データベース・オブジェクト名、Oracle データ型名または

式を示します。構文図の中でパラメータを見つけたときは、適切な型のオブジェクトや式に置き換えて SQL 文で使用してください。たとえば、CREATE TABLE 文を作成するには、構文図の *table* パラメータを作成するテーブル名（たとえば EMP など）に置き換えてください。（本文中でパラメータ名はイタリック体で示してあります。）

このマニュアルの構文図に出現するパラメータ、SQL 文で使用するときに置き換える値の例は、次のとおりです。

パラメータ	説明	例
table	パラメータで指定された型のオブジェクト名に置き換える必要があります。	emp
'text'	前後を引用符で囲んだ文字リテラルに置き換える必要があります。	'Employee Records'
condition	TRUE または FALSE に評価される条件値に置き換える必要があります。	ename > ’A’
date *d*	定数の日付または DATE データ型の式に置き換える必要があります。	TO_DATE ('01-Jan-1996', DD-MON-YYYY')
expr	あらゆるデータ型の式に置き換えることができます。	sal + 1000
integer	整数値に置き換える必要があります。	72
rowid	ROWID データ型の式に置き換える必要があります。	00000462.0001.0001
subquery	別の SQL 文に含まれている SELECT 文の値に置き換える必要があります。	SELECT ename FROM emp
statement_name *block_name*	SQL 文または PL/SQL ブロックの識別子に置き換える必要があります。	s1 b1

サンプル・コード

SQL および SQL*Plus のコマンドおよび文は、本文と区別してモノスペース・フォントで示してあります。次に例を示します。

```
INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');
ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
```

例文には、カンマや引用符などの句読点が含まれていることがあります。例文の句読点は、すべて必須です。例文は、すべてセミコロン（;）で終了します。セミコロンまたはその他の終了文字が文を終了するために必須であるかどうかは、アプリケーションによって異なります。

例文の大文字は、Oracle SQL のキーワードを示します。ただし、実際の SQL 文で、キーワードの大 / 小文字の区別はありません。

例文の小文字は、そのワードが単なる例として使用されていることを示します。たとえば、表、列またはファイルの名前を小文字で表します。

第I部

パラレル処理の基本

1

パラレル処理とパラレル・データベース

この章では、パラレル処理とパラレル・データベースのテクノロジを紹介します。このテクノロジにより、オンライン・トランザクション処理（OLTP）と意志決定支援システム（DSS）の効率が大幅に改善されます。このテクノロジを採用し、その強力なマルチプロセッシング機能を十分に活用することが管理者の課題といえます。

この課題を果たすためには、マルチプロセッシングのしくみ、必要なリソース、このテクノロジを効率的に適用できる状況と適用できない状況について理解する必要があります。この章では、次の項目に対する回答を示します。

- パラレル処理
- パラレル・サーバー
- パラレル処理の主な要素
- パラレル処理の利点
- パラレル・データベースの利点
- パラレル実行の役割
- パラレル・サーバーの必要性

パラレル処理

この項では、パラレル処理とその用途について説明します。

- パラレル処理の定義
- パラレル処理における考慮事項
- パラレル・システムの特徴
- SMP と MPP のパラレル処理
- 統合運用のためのパラレル処理

パラレル処理の定義

パラレル処理では、1 つの大きなタスクが多数の小さなタスクに分割され、複数のノードで同時に実行されます。この結果、大きなタスクの実行時間が短くなります。

注意： ノードとは、通常は独立した 1 台のマシンに搭載されている独立したプロセッサのことです。ただし、1 台のマシンに複数のプロセッサを搭載させることもできます。

タスクには、効率よく複数のタスクに分割でき、パラレル処理に適しているものがあります。たとえば、係員が 1 人しかいない銀行では、顧客は全員 1 つの列に並ぶことになります。しかし、係員が 2 人の場合には、顧客への対応業務を効率よく 2 人の係員で分担できるので、顧客は 2 倍の速さでサービスを受けられます。これはパラレル処理が効果的なソリューションとなる例です。

タスクを分割しても効果が得られない場合もあります。たとえば、前の例で、ローンの申込みはすべて銀行支配人の承認が必要だとします。この場合は、パラレル処理によってサービスが改善されるとはかぎりません。なぜなら、ローンの窓口業務を処理する係員が何人いても、申込みはすべて銀行支配人の承認を得るために 1 列に並ぶことになるからです。この根本的な制約は、最大限のパラレル処理でも克服できません。

次の 2 つの図は、1 つの問合せの順次処理とパラレル処理を比較しています。図 1-1 は、1 つの問合せがシングル・タスクとして処理される順次処理を示しています。

図 1-1 シングル・タスクの順次処理

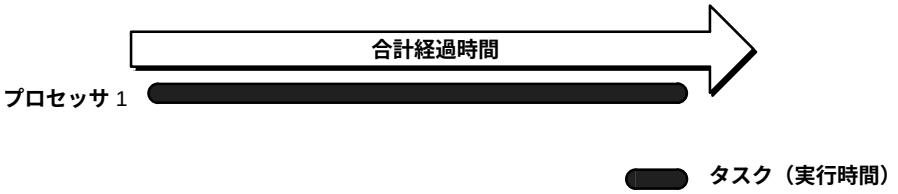
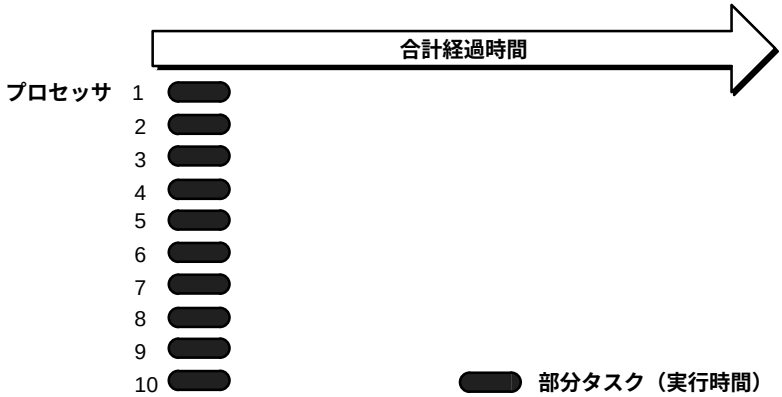


図 1-2 は、1 つの問合せが複数の小さなタスクに分割され、各コンポーネント・タスクが個別のインスタンスで実行されるパラレル処理を示しています。

図 1-2 コンポーネント・タスクを並列で実行するパラレル処理



次の図では、複数のオンライン・トランザクション処理（OLTP）環境で複数の個別タスクを順次処理する場合とパラレル処理する場合を比較しています。図 1-3 は、複数の個別タスクの順次処理を示しています。

図 1-3 複数の個別タスクの順次処理

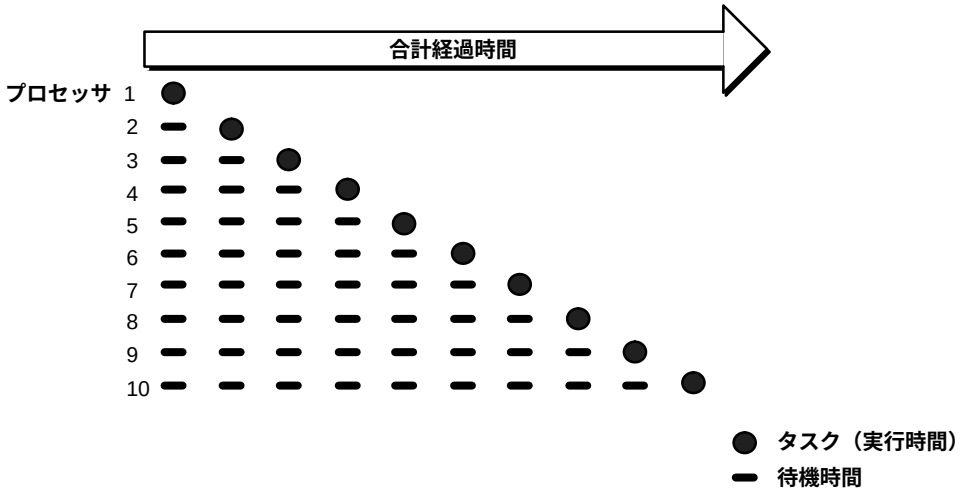


図 1-4 は、複数の個別タスクのパラレル処理を示しています。

図 1-4 複数の個別タスクを並列で実行するパラレル処理

順次処理では、1 つのリソースに対して各タスクが競合します。タスク 1 だけが待機せずに実行されます。タスク 2 は、タスク 1 が完了するまで待機する必要があります。さらに、タスク 3 はタスク 1 とタスク 2 が完了するまで待機する必要があります。この図では、各タスクが同じ大きさで示されていますが、実際にはタスクの大きさは異なる場合があります。

これに対して、対称マルチプロセッサを搭載したパラレル・サーバーでは、CPU リソースの分割のしかたによって各タスクに割り当てられる CPU リソースが多くなります。各タスクは待機することなくそれぞれのプロセッサでただちに実行されます。

パラレル処理における考慮事項

効率的なパラレル処理を実現するには、次の 2 つの課題があります。

- いくつかのタスクを同時に " 並列で " 実行できるようにタスクを構成する。
- 順次処理を行う必要があるタスクの実行順序を確保する。

パラレル・システムの特徴

パラレル処理システムの特徴は、次のとおりです。

- システムの各プロセッサがタスクを同時に処理できる。
- タスクの同期をとることが必要な場合がある。
- 一般にノードはデータ、ディスクおよびその他のデバイスなどのリソースを共有する。

SMP と MPP のパラレル処理

パラレル処理アーキテクチャでは、次のハードウェアがサポートされます。

- クラスタ化ハードウェアおよび大規模パラレル・プロセッシング（MPP）ハードウェア。このようなハードウェアの各ノードは、それぞれがメモリーを装備している。
- シングル・メモリー・システム。" 対称マルチプロセッシング "（SMP）ハードウェアとも呼ばれ、1 つのメモリー・リソースを複数のプロセッサが使用する。

クラスタ化された MPP マシンは複数のメモリーを装備しており、通常は各ノードがそれぞれメモリーを所有しています。このようなシステムでは、必要なメモリーとバス・コンポーネントを使用してメモリーのボトルネックを排除することにより、パフォーマンスとコストが大幅に改善されます。

1 種類のハードウェアしかサポートしていないデータベース管理システムでは、アプリケーションの移植性、および新規ハードウェア・システムへのアプリケーションの移行の可能性、アプリケーションの拡張性が制限されます。Oracle Parallel Server（OPS）では、クラスタおよび MPP システムの両方を利用可能なため、このような制限はありません。パラレル・サーバー・オプションのない Oracle では、シングル CPU マシンまたは SMP マシンのいずれかを使用します。

統合運用のためのパラレル処理

パラレル・データベース・ソフトウェアでは、システムの処理性能を効率的に活用し、オンライン・トランザクション処理（OLTP）アプリケーション、意思決定支援システム（DSS）アプリケーション、OLTP と DSS の混在するシステム（" ハイブリッド " システム）など、多様なアプリケーションを処理する必要があります。

OLTP アプリケーションは、トランザクションが小さく、CPU および I/O の使用率が低いことが特徴です。一方、DSS アプリケーションのトランザクションは大きく、CPU および I/O の使用率も高くなります。

パラレル・データベース・ソフトウェアは、問合せ専用データベースとして機能するように特殊化されていることが多いです。ただし、専用サーバーは 1 つの機能を実行するように設計されているため、統合運用のための一般的な基盤となることはありません。このような運用形態には、オンライン意思決定支援システム、バッチ・レポート・システム、データ・ウェアハウジング・システム、OLTP、分散操作システム、高可用性システムがあります。専用サーバーは、DSS アプリケーションなどの大規模なデータベースの分野で最も有効に活用されてきました。

多くの機能を持つパラレル・データベース・ソフトウェアは、オープン・システム・ハードウェアで卓越したコスト・パフォーマンスを実現し、エンタープライズ・コンピューティングに関する多様なニーズに対応します。オンライン・バックアップ、データ・レプリケーション、移植性、相互運用性、多種クライアント・ツールのサポートなどの機能をパラレル・サーバーに備えることにより、アプリケーションの統合運用、分散操作、混在するアプリケーションによる作業負荷をサポートできるようになります。

パラレル・サーバー

複数のコンピュータが、データ、ソフトウェア、周辺装置へのアクセスを共有できるハードウェア・アーキテクチャは多数あります。パラレル・サーバーは、1 つの物理データベースを " 共有 " する複数インスタンスを実行することで、このようなアーキテクチャを有効活用します。適切なアプリケーションでは、より大きい作業負荷に対応したスピードアップおよびスケールアップを実現し、優れたパフォーマンスによって複数のマシン上のユーザーが 1 つのデータベースにアクセスできます。

パラレル・サーバーは、複数のノードの CPU を使用してトランザクション・ストリームを処理することで、トランザクションをパラレル処理します。この場合、各 CPU がそれぞれ 1 つのトランザクションを処理します。パラレル・データ操作言語（PDML）を使用すれば、1 つのトランザクションを複数のノードで実行できます。データ・アクセス所要時間の短いオンラインの挿入および更新トランザクションからなるアプリケーションが多いので、これは効果的な方法です。パラレル・データベースは、CPU 間で作業負荷を均衡化するだけでなく、データの整合性を保ちながらデータへの同時アクセスを提供します。

関連項目： 1-14 ページの「パラレル・サーバーの必要性」（Oracle の構成についての説明）

パラレル処理の主な要素

この項では、パラレル処理の主な要素について説明します。

- スピードアップとスケールアップ : パラレル処理の目標
- 同期処理：重要な成功要素
- ロック機能
- メッセージ機能

スピードアップとスケールアップ : パラレル処理の目標

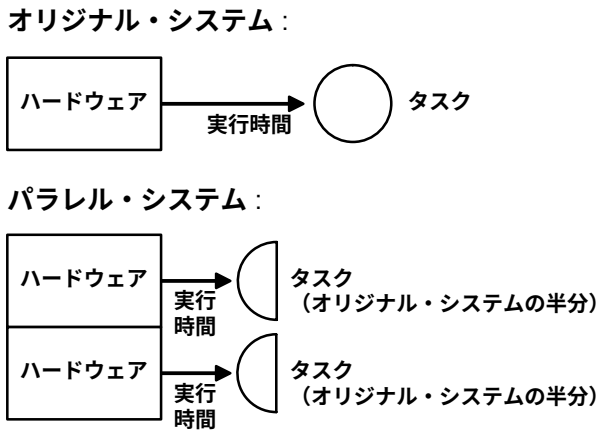
パラレル処理のパフォーマンスに関する目標は、次の 2 つの重要な項目で評価できます。

- スピードアップ
- スケールアップ

スピードアップ

スピードアップとは、ハードウェアを増設して同じタスクを実行した場合に、元のシステムと比較して実行速度がどれだけ改善されるかということです。ハードウェアを追加すると、一定量のタスクでの実行時間の短縮が評価されます。図 1-5 に、シングル・システムで実行した場合の半分の所要時間で、パラレル・ハードウェア・システムがオリジナル・タスクを半分ずつ実行するようすを示します。

図 1-5 スピードアップ



プロセッサを増設するとシステム応答時間が短縮されスピードアップを実現できます。

$$\text{スピードアップ} = \frac{\text{オリジナル・システムの実行時間}}{\text{パラレル・システムの実行時間}}$$

計算式で使われている各項の内容は、次のとおりです。

オリジナル・システムの実行時間（*Time_Original*）	特定のタスクを小規模なシステムで実行したときにかかった時間です。
パラレル・システムの実行時間（*Time_Parallel*）	同じタスクを大規模なパラレル・システムで実行したときにかかった時間です。

たとえば、あるタスクの実行時間がオリジナル・システムでは 60 秒、2 つのパラレル・システムでは 30 秒である場合、スピードアップ値は 2 になります。

$$2 = \frac{60}{30}$$

しかし、直線的なスピードアップは経験できないでしょう。スピードアップはむしろ、対数で表現されることが多いでしょう。たとえば、サイズ "x" のタスクを時間 "n" で実行できる

システムがあるとします。このシステムでサイズ 2x のタスクを実行すると、所要時間は 3n となる場合があります。

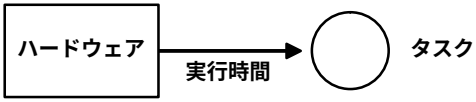
> **注意：** ほとんどの OLTP アプリケーションでは、スケールアップだけが実現され、スピードアップは期待できません。実際は、同期によるオーバーヘッドが原因でスピードダウンが発生する可能性があります。

スケールアップ

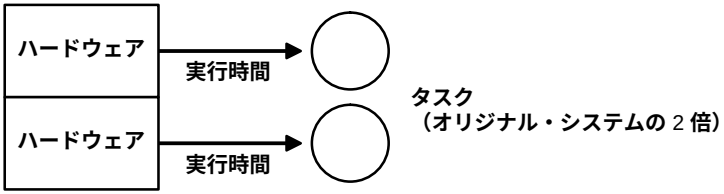
スケールアップとは、システムを拡張した場合に、同じ時間内にどれだけ多くのタスクを実行できるようになるかを係数で表したものです。スケールアップの計算式では、ハードウェアの追加によって一定の時間に実行できるタスクの量の増加が評価されます。

図 1-6 スケールアップ

オリジナル・システム：



パラレル・システム：



トランザクションの量が増加してもスケールアップの値が高いと、CPU などのハードウェア・リソースを追加することで、システム応答時間が一定に保たれます。

スケールアップは次の計算式で算出できます。

$$\text{スケールアップ} = \frac{\text{パラレル・システムのトランザクション処理量}}{\text{オリジナル・システムのトランザクション処理量}}$$

計算式で使われている各項の内容は、次のとおりです。

オリジナル・システムのトランザクション処理量（*Volume_Original*）	特定の時間内に小規模なシステムで処理されるトランザクションの量です。
パラレル・システムのトランザクション処理量（*Volume_Parallel*）	特定の時間内にパラレル・システムで処理されるトランザクションの量です。

たとえば、オリジナル・システムが特定の時間内に 100 トランザクションを処理し、パラレル・システムが同じ時間内に 200 トランザクションを処理する場合、スケールアップ値は 2 になります。値 2 は、理想的な直線的なスケールアップを示します。これは、ハードウェア数を 2 倍にすると、同じ時間内に処理できるデータ量も 2 倍になることを意味します。

同期処理：重要な成功要素

同時実行タスクを調整することを、同期といいます。同期は、正確な処理のために必要な機能です。パラレル処理を成功させるには、タスクを分割して同期の必要性を低くすることが重要です。同期の必要性が低くなると、スピードアップとスケールアップが向上します。

ノード間のパラレル処理では、パラレル・プロセッサを高速でインターコネクトすると有効です。ノード間で大量データを送受信する必要がある場合、この同期によるオーバーヘッドが非常に高くなることがあります。ノード内のパラレル処理では、メッセージ機能は必要ありません。かわりに共用メモリーが使用されます。ノード間のメッセージ機能とロック機能は、統合分散ロック・マネージャ（IDLM）によって処理されます。

同期の負荷は、リソースの量、およびそのリソースを使用するユーザーとタスクの数によって異なります。調整する同時実行タスクの数が少ない場合は同期の負荷も少なくなりますが、多数の同時実行タスクを調整する場合は同期の負荷も多くなります。

オーバーヘッド

同期に長時間かかる場合、リソースに対して競合が集中的に発生しています。

注意： 同期のために費やす時間が長すぎると、パラレル処理を実行する利点が損なわれる可能性があります。同期にかかる時間が少なければ、スピードアップおよびスケールアップが向上します。

応答時間とは、通常の処理にかかる時間と待機時間を合計した時間です。表 1-1 に、同時実行プロセスの増加によって、どのようにオーバーヘッドが増加するかを示します。3 つのプロセスが同時に 1 つのサービスを要求し、これらの要求が順次処理された場合、プロセス 1 の応答時間は 1 秒です。そしてプロセス 2 の応答時間は 2 秒（プロセス 1 の完了の待機時間が 1 秒、処理時間が 1 秒）、プロセス 3 の応答時間は 3 秒（待機時間が 2 秒、処理時間が 1 秒）となります。

表 1-1 プロセスの増加によるオーバーヘッドの増加

プロセス番号	処理時間	待機時間	応答時間
1	1 秒	0 秒	1 秒
2	1 秒	1 秒	2 秒
3	1 秒	2 秒	3 秒

実際には、1 つのタスクで複数のメッセージが必要な場合もあります。タスクが継続的に同期を待つ必要がある場合、タスク 1 つにつき複数のメッセージが必要となることがあります。

同期のコスト

同期は、処理の正確性を保つ上で必要なパラレル処理の一要素ですが、パフォーマンスとシステム・リソースの点から、同期のコストを管理する必要があります。同期はさまざまなパラレル処理ソフトウェアで可能になっていますが、コストを効率的に抑えることができる場合と、そうでない場合があります。

同期を非常に低いコストで達成できる場合もあります。一方、同期のコストが非常に高い場合もあります。たとえば、1 つの表に対して複数のノードから挿入が実行される場合は、多数の同期が必要となります。同一のデータ・ブロックに複数のノードが挿入を実行する場合、競合が集中的に発生します。これは、このデータ・ブロックが複数のノード間で受け渡されるためです。この種の同期は可能ですが、オーバーヘッドが非常に大きくなる場合があります。

関連項目： 第 8 章「統合分散ロック・マネージャ」、第 12 章「アプリケーションの分析」および第 19 章「チューニングによるパフォーマンスの最適化」

ロック機能

ロックは、タスクを同期化するリソース制御機構です。パラレル処理で必要なタスクの同期を実現するには、さまざまな種類のロック・メカニズムが必要です。

統合分散ロック・マネージャ（以降、統合 DLM または IDLM と記述する）は、OPS で使用される内部ロック機能です。統合 DLM は、パラレル・サーバーが稼働するノード間でのリソース共有を調整します。パラレル・サーバーのインスタンスは、IDLM を使用して相互に通信し、データベース・リソースの変更を調整します。各ノードは、同一リソースに対して競合する場合を除き、他のノードとは独立して動作します。

IDLM を使用すると、複数のアプリケーションによるリソース（データ、ソフトウェア、周辺装置など）へのアクセスを同期化できるため、複数のノードで稼働しているアプリケーションが同一リソースに対して同時に出した要求を調整できます。

IDLM はアプリケーションのために次の処理を実行します。

- リソースの現在の " 所有者 " を追跡する。
- リソースに対するアプリケーション・プロセスからのロック要求を受け入れる。
- リソースのロックが使用可能になったときに、要求を出しているプロセスに通知する。
- リソースをブロックしているプロセスに対してロックを解放するかダウングレードするように通知する。
- プロセスのためにリソースへのアクセスを獲得する。

関連項目： 第 7 章「ロック・メカニズムの概要」（Oracle データベースの内部ロック・メカニズムについて）および第 8 章「統合分散ロック・マネージャ」

メッセージ機能

パラレル処理において最大のパフォーマンスを得るには、高速で効率的なノード間通信が必要です。最適なシステムとは、IDLM との効率的な通信を実現する、高帯域幅、低待ち時間のシステムです。帯域幅とは、1 秒間に送信できるメッセージの合計サイズです。待ち時間とは、1 つのメッセージをインターコネクトに投入し、応答を受信するまでの秒数です。

ほとんどの MPP システムおよびクラスタは、十分な高帯域幅を備えたネットワークを使用しています。一方、待ち時間はオペレーティング・システムの問題であり、インターコネクトのためのソフトウェアおよびプロトコルによって大きく左右されます。MPP システムおよびほとんどのクラスタは一般的に、高帯域幅、低待ち時間のシステムとのインターコネクトを使用します。一部のクラスタでは、比較的低い帯域幅、高待ち時間のシステムとのイーサネット接続を使用することがあります。

パラレル処理の利点

特定の種類のアプリケーションでは、パラレル処理により次のような利点が得られます。

- スループットの改善 : スケールアップ
- 応答時間の改善 : スピードアップ

図 1-3 に示したように、大きなタスクを小さなコンポーネント・タスクに分割するか、または待機時間を短くすることで応答時間を改善できます。

表 1-2 に、パラレル処理を適切に実行した場合にスピードアップとスケールアップを実現できる作業負荷の種類を示します。

表 1-2 さまざまな作業負荷のスピードアップおよびスケールアップ

作業負荷	スピードアップ	スケールアップ
OLTP	不可	可
DSS	可	可
バッチ（複合）	可能性あり	可
パラレル問合せオプション	可	可

スループットの改善 : スケールアップ

複数のタスクをそれぞれ独立して実行できる場合は、タスクを複数の CPU またはノードに分散できます。これによりスケールアップが実現し、同一時間内にデータベースで実行できるプロセスの数が増加します。

プロセスの処理速度を 10 倍にできる場合、同一時間内での処理量は 10 倍になります。たとえば、パラレル問合せ機能ではスケールアップを実現できます。問い合せるデータの量が 10 倍になった場合、またはユーザー数が増加した場合でも、応答時間は変わりません。パラレル問合せ機能のない OPS でもスケールアップは実現されますが、この場合、1 つの問合せを複数のノードで順次実行する必要があります。

DSS、OLTP およびレポート・アプリケーションでは、複数のプログラムをそれぞれ個別のノードで実行することでスケールアップを実現できます。また、バッチ・プログラムを書き直して多数のパラレル・ストリームに分割し、使用可能な複数の CPU を利用することで、スピードアップを実現できます。

応答時間の改善 : スピードアップ

DSS アプリケーションおよびパラレル問合せでは、パラレル処理によってスピードアップを実現できます。それぞれのトランザクションの処理速度が改善されます。ただし、OLTP アプリケーションではスケールアップだけが実現され、スピードアップは実現しません。
OLTP アプリケーションでは、それぞれのプロセスが独立しています。パラレル処理でも、順序表に対する挿入または更新は同じ速度で実行されます。実際には、同期のオーバーヘッドが原因で、多少のスピードダウンが発生することがあります。実行するそれぞれの操作は小さいので、操作をパラレル化するのは適当ではありません。同期のオーバーヘッドがパラレル処理の利点を上回るからです。

バッチ処理でもスピードアップを実現できます。しかし、スピードアップの程度は、タスク間の同期のためのコストおよび量に左右されます。

パラレル・データベースの利点

特定のアプリケーションでは、パラレル・データベースのテクノロジの次のような利点を活用できます。

- パフォーマンスの改善
- 高い可用性
- 柔軟性の向上
- ユーザー数の増加

パフォーマンスの改善

アプリケーションが使用できる CPU の数が増えると、スピードアップとスケールアップが向上します。パフォーマンスの改善は、ノード間のロックおよび同期アクティビティの頻度や規模に左右されます。ロック・オペレーションはプロセッサおよびメッセージを集中的に使用して実行されるため、待ち時間が多くなります。ロック・オペレーションおよびデータベース競合の量は、IDLM のスループットおよびパフォーマンスと同様、システムの拡張性を決定する重要な要因となります。

高い可用性

ノードはそれぞれ独立しているため、あるノードで障害が発生してもシステム全体のダウンにはつながりません。障害が発生したノードは、障害のないノードのうちの 1 つによって回復されるため、ユーザーは引き続きデータにアクセスできます。つまり、シングル・ノード・システムでノードに障害が発生した場合に比べ、データの可用性がたいへん高くなります。また、データベースの可用性も大幅に向上します。

柔軟性の向上

OPS 環境は、たいへん柔軟性の高い環境です。インスタンスを必要に応じて割り当てたり、割当てを解除したりできます。たとえば、データベースへのアクセス要求が増加したときに、インスタンスを一時的に多く割り当てることができます。また、インスタンスが不要になったときは、割当てを解除して他の用途に使用できます。

ユーザー数の増加

パラレル・データベース・テクノロジによりメモリーの制限を解消できるため、数千におよぶユーザーが 1 つのシステムを使用できるようになります。

パラレル・サーバーの必要性

この項では、さまざまなアプリケーションのパフォーマンスを改善する次の Oracle 構成について説明します。

- 排他アクセス権限を持つ単一インスタンス
- 複数インスタンス・データベース・システム
- 分散データベース・システム
- クライアント / サーバー・システム

パラレル・サーバーは、多数のユーザーが使用できるパフォーマンスの高いリレーショナル・データベースを提供する Oracle オプションの 1 つです。各自のニーズにあわせて各構成を組み合せて使用できます。パラレル・サーバーは分散データベース環境のサーバーとして使用できます。また、クライアント / サーバー構成では、特定のアプリケーションの要件に合うように、さまざまな Oracle 構成を組み合せてハイブリッド・システムを構築できます。

注意： サポートされる Oracle 構成は、プラットフォームによって異なります。使用する構成が使用しているプラットフォームでサポートされていることを確認してください。

パフォーマンスを最適化するには、特定のアプリケーションの要件および使用できるリソースに応じてシステムを構成し、その構成を最大限に活用できるようにデータベースおよびアプリケーションを設計、調整します。この場合、既存のハードウェアまたはソフトウェアを新規システムまたは将来のシステムに移行することも考慮してください。

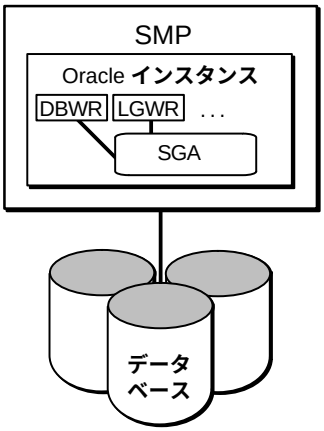
以降の項には、各自の要件に最も適した Oracle 構成を考えるときに役立つ情報を記載しています。

関連項目： 第 3 章「パラレル・ハードウェア・アーキテクチャ」

排他アクセス権限を持つ単一インスタンス

図 1-7 に、対称マルチプロセッサ（SMP）で稼働する単一インスタンス・データベース・システムを示します。データベース自体は一組のディスク上にあります。

図 1-7 単一インスタンス・データベース・システム



1 つのデータベースにアクセスする 1 つのインスタンスのパフォーマンスを改善するには、1 台の大型コンピュータでインスタンスを稼働します。大型コンピュータ 1 台の場合、複数ノード間での調整は必要ないため、複数ノード・システムで 2 台の小型コンピュータを使用する場合よりもパフォーマンスが改善されます。ただし、小型コンピュータ 2 台の方が大型コンピュータ 1 台の場合よりもコストが低く抑えられる事も多いです。

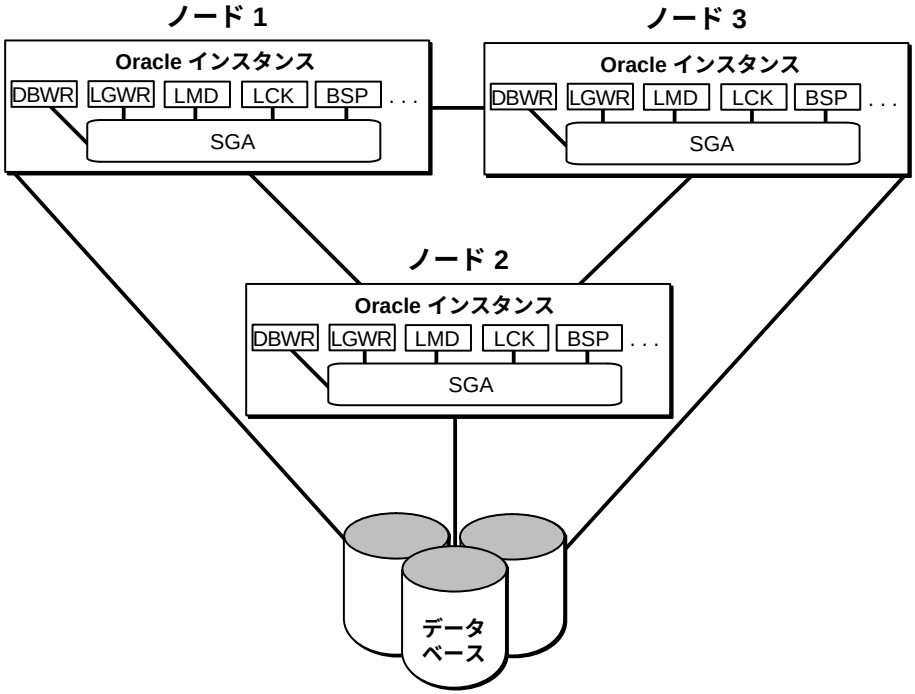
シングル・コンピュータ・システムから複数ノード・システムへ移行する場合、OPS 向けにデータベースを再設計し、アプリケーションをチューニングすると、そのコストは非常に高くなります。このような状況では、パラレル・サーバーに移行するよりも大型コンピュータ 1 台のシステムを使用する方が効率的であるかどうかを考慮してください。

関連項目：『Oracle8*i* 概要』（単一インスタンスの Oracle について）

複数インスタンス・データベース・システム

図 1-8 に、クラスタまたは MPP 上で稼働している OPS オプションを示します。この構成を " 複数インスタンス・データベース・システム " と呼びます。OPS は、複数のノード上にあるインスタンス間でのデータの受渡しを、最小限に抑えるように構成できるアプリケーション向けの優れたデータベース・システムです。

図 1-8 複数インスタンス・データベース・システム



このシステムでは、フォアグラウンド・プロセスの他に、それぞれのインスタンスで LMD、LCK および BSP の各プロセスが必要です。これらのプロセスでは、インターコネクトを使って他のインスタンスと直接通信することによってグローバル・ロックを調整します。

注意： BSP はブロック・サーバー・プロセスであり、OPS 環境にしか存在しません。BSP は、一貫性のある読込みブロックをノードから他のノードへと送信するほかに、要求元ノードへの発信メッセージを管理します。LMD プロセスでは、インスタンスが保持しているロックを他のインスタンスで必要になったときのために、着信ロック要求を管理します。

関連項目： BSP の詳細は、5-6 ページの「キャッシュ・フュージョン処理およびブロック・サーバー・プロセス」および第 20 章「キャッシュ・フュージョンとインスタンス間パフォーマンス」を参照してください。

OPS では、インスタンスはデータベースに 1 対 1 で対応していません。排他モードでは、インスタンスとデータベースは 1 対 1 で対応しています。共有（パラレル）モードでは 1 つのデータベースに対して複数のインスタンスが存在します。

通常、あるアプリケーションが大型システムのデータベースへの排他アクセス権限を持っている場合、このアプリケーションを 1 つだけで実行したときのパフォーマンスは、このアプリケーションを複数ノード環境の小型ノードで実行したときのパフォーマンスよりも高くなります。これは、複数ノード環境に移行すると、自動的に同期のコストが増えるからです。このようなパフォーマンスの相違は、実行するアプリケーションの特性と、データベースへのアクセスを共有するその他のアプリケーションの特性に応じて異なります。

次の特性のいずれかまたは両方を備えたアプリケーションは、パラレル・サーバーの個々のインスタンス上での実行に適しています。

- 主にデータの読取りだけを行う（データを更新しない）アプリケーション
- 別々のデータブロック（のグループ）を変更する、または同じデータ・ブロックを異なる時点で変更するアプリケーション

関連項目： 4-1 ページの「パラレル・サーバーの有効化と無効化」、第 8 章「統合分散ロック・マネージャ」および『Oracle8*i* 概要』（DBWR、LGWR、LMD および LCK バックグラウンド・プロセスについて）

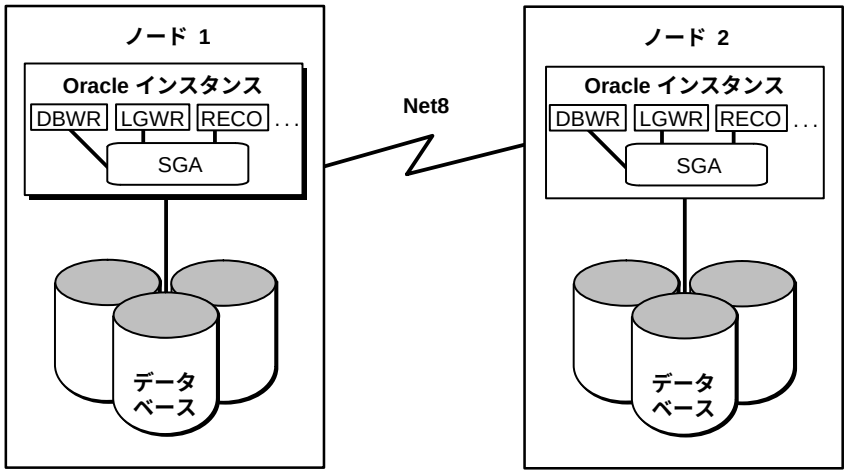
分散データベース・システム

複数の Oracle サーバーとデータベースをリンクし、分散データベース・システムを形成できます。分散データベース構成には複数のデータベースが含まれています。これらの各データベースへは、1 つのサーバーのインスタンスが直接アクセスしたり、他のインスタンスがサーバー対サーバーの協調動作を介してアクセスしたりすることができます。それぞれのノード上のサーバーが他ノード上のリンクされたデータベースを操作できますが、データはノード間で永続的に分割されています。一方、パラレル・サーバーでは、複数のインスタンスが 1 つのデータベースに直接アクセスできます。

注意： Oracle Parallel Server は、分散データベースの構成要素の 1 つになることができます。

図 1-9 に、分散データベース・システムを示します。分散データベース・システムでは、各インスタンスに RECO バックグラウンド・プロセスが必要です。OPS 構成ではないので統合分散ロック・マネージャは必要ないため、LCK、LMON または LMD バックグラウンド・プロセスはありません。

図 1-9 分散データベース・システム

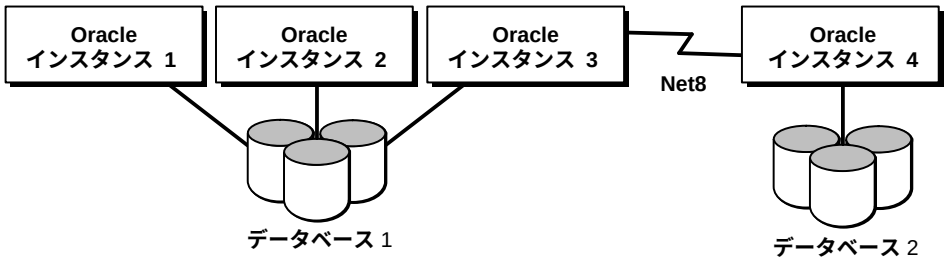


サーバーからリモート・データベースへの Net8 を介したアクセスは透過的であるため、分散システムの複数のデータベースは 1 つの論理データベースとして扱うことができます。

データを分割して複数のデータベースに格納し、オーバーラップを最小限に抑えられる場合は、パラレル・サーバーではなく分散データベース・システムを使用し、Net8 によってデータベース間でデータを共有します。パラレル・サーバーを使用すると、自動的に共通のデータベースを介してノード間でデータが共有されます。

分散データベース・システムでは、物理的に離れている複数の個別サイトにデータを常駐させることができます。それぞれの個別ノードがネットワークで接続されていれば、ユーザーは物理的に離れたデータベースのデータにアクセスできます。パラレル・サーバーでは、待ち時間が短いこと、高帯域幅のノード間通信が要求されるため、すべてのデータが 1 箇所に保管されている必要があります。しかし、図 1-10 で示すように、パラレル・サーバーを分散データベース・システムの一部に組み込むことができます。

図 1-10 分散データベースに組み込まれている Oracle Parallel Server



複数のデータベースで構成されるシステムでは、それぞれのデータベースを個別に管理する必要があり、分散データベース・システムではデータベースとネットワーク・プロトコルの管理を調整する必要があります。パラレル・サーバーを使用すると複数のデータベースを整理統合できるため、管理作業が簡単になります。

複数のデータベースで構成されるシステムでは、1 つのインスタンスが 1 つのデータベースにアクセスするシステムよりも可用性が高くなります。それは、分散データベース・システムで 1 つのインスタンスに障害が発生しても、他のデータベースのデータにはアクセスできるからです。この場合、障害のあるインスタンスが所有しているデータベースだけがアクセス不能になります。ただしパラレル・サーバーを使用している場合は、1 つのインスタンスで障害が発生しても、すべてのデータ（障害が発生したノードで稼働しているインスタンスがアクセスしているデータを含む）に引き続きアクセスできます。

1 つのデータベースにアクセスするパラレル・サーバーでは、通常、どのノードが表への書込みを実行するかに関係なく、どのノードのトランザクションでも複数の表に同時に書込みを実行できるため、分散更新、分散挿入、分散削除、よりコストのかかる 2 フェーズ・コミットを実行する必要がなくなります。

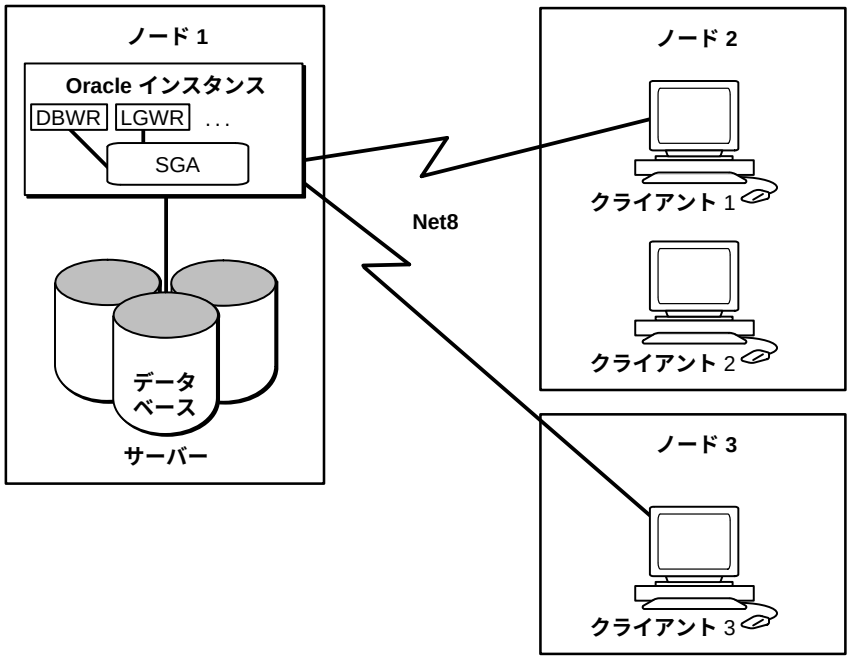
> **関連項目：**『Oracle8*i* バックアップおよびリカバリ・ガイド』（インスタンスの回復について）および『Oracle8*i* 分散システム』（Oracle の分散データベース機能について）

クライアント / サーバー・システム

Oracle 構成はすべて、クライアント / サーバー環境で実行できます。Oracle では、リモート・コンピュータでクライアント・アプリケーションを実行し、このアプリケーションから Net8 を使用してネットワーク経由で Oracle サーバーにアクセスできます。この構成でのパフォーマンスは、通常シングル・サーバー・ノードの性能によって制限を受けます。

図 1-11 に、Oracle クライアント / サーバー・システムを示します。

図 1-11 クライアント / サーバー・システム



注意： クライアント / サーバー処理は、どの Oracle 構成にも適しています。使用している Oracle のプラットフォームでのクライアント / サーバー処理の実現方法は、プラットフォーム固有のマニュアルを参照してください。

クライアント / サーバー構成では、Oracle サーバーが稼働しているコンピュータの処理量を減らすことができます。1 つのマシンで実行しているアプリケーションの数が多すぎる場合には、これらのアプリケーションを複数のマシンに分散することでパフォーマンスを改善で

きます。ただし、データベース・サーバーがその処理能力の限界に達している場合には、大型のシステムまたは複数ノード・システムへの移行を考慮します。

複数の計算集中型アプリケーションを実行する場合、複数ノード・システムの 1 つのノードでいくつかのアプリケーションを実行し、Oracle およびその他のアプリケーションを別のノードまたは他の複数ノードで実行することができます。このように実行すると、パラレル・マシンのノードのうち、1 つのノードをサーバー・ノードとして、他の複数のノードをクライアント・ノードとして使用できます。

データベースが高スループットの個別部分で構成されている場合、高パフォーマンスのノード上で稼働しているパラレル・サーバーは、データベースの各部分を迅速に操作し、なおかつ頻繁には行われないデータベース全体にまたがる操作も処理できます。

クライアント / サーバー構成では、クライアントとデータベースとの通信はすべてネットワークを介して行う必要があります。多くのバッチ・アプリケーションでは、通信するデータ量が膨大であるため、この構成が不適切な場合があります。

関連項目：『Oracle8*i* 概要』の「クライアント / サーバー・アーキテクチャ」

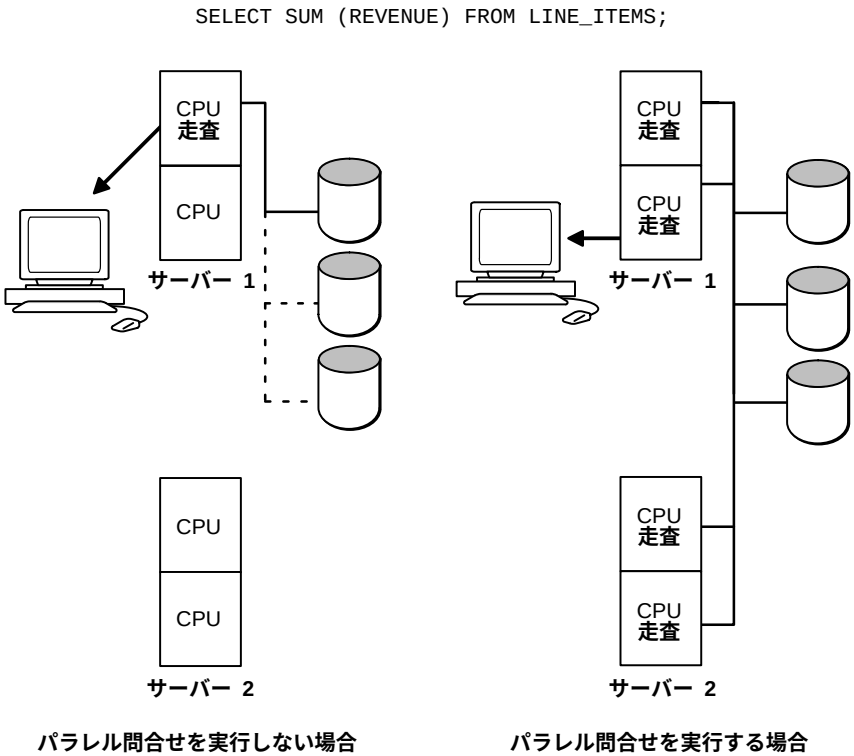
パラレル実行の役割

Oracle のパラレル実行機能により、特定のタイプの SQL 文の処理を複数の問合せサーバー・プロセスに分割できます。

OPS は、ノード間でのパラレル実行のためのフレームワークを提供します。パラレル実行機能は、Oracle にパラレル・サーバー・オプションが付いているかどうかにかかわらず、同じように動作します。パラレル・サーバー・オプションが付いている場合とそうでない場合の唯一の相違点は、OPS では 1 つの問合せ処理または他のパラレル処理を複数のノードで実行できることです。

一部のアプリケーション、特にデータ・ウェアハウス・アプリケーションでは、個々の問合せを処理するときに、通常のオンラインの挿入または更新トランザクションよりも大量の CPU リソースを消費し、多くのディスク I/O が必要です。マルチプロセッサ・システムを活用するには、データ・サーバーが個々の問合せを、同時に処理できる作業単位として並列化する必要があります。図 1-12 に、パラレル問合せ処理の例を示します。

図 1-12 パラレル問合せ処理の例



問合せが並列に実行されないと、ディスクからデータが 1 つの I/O ストリームで順次読み込まれます。LINE_ITEMS 表のすべての行を走査し、すべての行の REVENUE の合計を算出する処理を 1 つの CPU が実行します。問合せを並列化すると、ディスクからデータが複数の I/O ストリームで並列に読み込まれます。複数の CPU がそれぞれ表の一部を並列に走査し、その結果を集計します。パラレル問合せでは、複数の CPU だけでなく使用できる I/O 帯域幅の利点も活用できます。

関連項目：『Oracle8*i* 概要』および『Oracle8*i* チューニング』（パラレル実行の詳細）

2

パラレル処理の実現

There is an old network saying: Bandwidth problems can be cured with money.Latency problems are harder because the speed of light is fixed—you can't bribe God.

— David Clark, MIT

スピードアップとスケールアップを達成するには、パラレル処理およびパラレル・データベース・テクノロジを効率よく実現する必要があります。つまり、システムを最初からパラレル処理向けに設計および構築します。この章では、次の項目について説明します。

- 4 つのレベルの拡張性
- パラレル処理が有効な状況
- パラレル処理が有効でない状況
- 効率的な処理の分割方法のガイドライン
- 一般的なパラレル処理の誤解

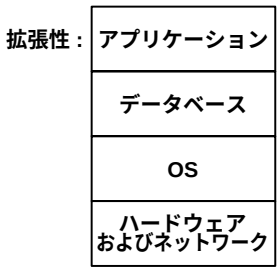
4 つのレベルの拡張性

パラレル処理とパラレル・データベースを効率よく実現するには、次の 4 つのレベルの拡張性を最適化する必要があります。

- ハードウェアとネットワークの拡張性
- オペレーティング・システムの拡張性
- データベース管理システムの拡張性
- アプリケーションの拡張性

注意： アプリケーションの設計が適切でないと、そのシステムで可能な拡張性を十分に利用できないことがあります。同様に、アプリケーションの拡張性が高くても、実行するハードウェアの拡張性がない場合は、期待したパフォーマンスを得ることはできません。

図 2-1 拡張性のレベル



ハードウェアとネットワークの拡張性

ハードウェアの拡張性では、インターコネクト（相互接続）が重要なポイントとなります。なぜなら、高速バス接続であれ、低速なイーサネット接続であれ、すべてのシステムはなんらかの方法で各 CPU を接続する必要があるためです。したがって、インターコネクトの帯域幅と待ち時間がハードウェアの拡張性の決め手となります。

関連項目： 3-3 ページの「ハードウェア要件およびオペレーティング・システム・ソフトウェア要件」

帯域幅と待ち時間

ほとんどのインターコネクトでは、十分な帯域幅があります。実際には、帯域幅が大きいと待ち時間の値も高くなることがあります。

ハードウェアの拡張性は、待ち時間が非常に低いことに依存しています。ロック調整のために生じる通信量の特徴として、LMD プロセス間で多数の非常に短いメッセージが送受信されます。

たとえば、高速道路で 100 人の乗客を 1 台のバスで運ぶことと、100 台の乗用車で運ぶことの違いを考えてみてください。後者の場合、効率性は乗用車が高速道路にすばやく出入りできるかどうかという点に大きく依存しています。高速道路で多くの車が通行できるように 5 車線が設けられていても、入り口ランプが 1 車線しかなければ、" 高速 " 道路への入り口で渋滞が生じる可能性があります。

その他のノード間の処理、たとえばパラレル問合せなどは、高い帯域幅に依存します。

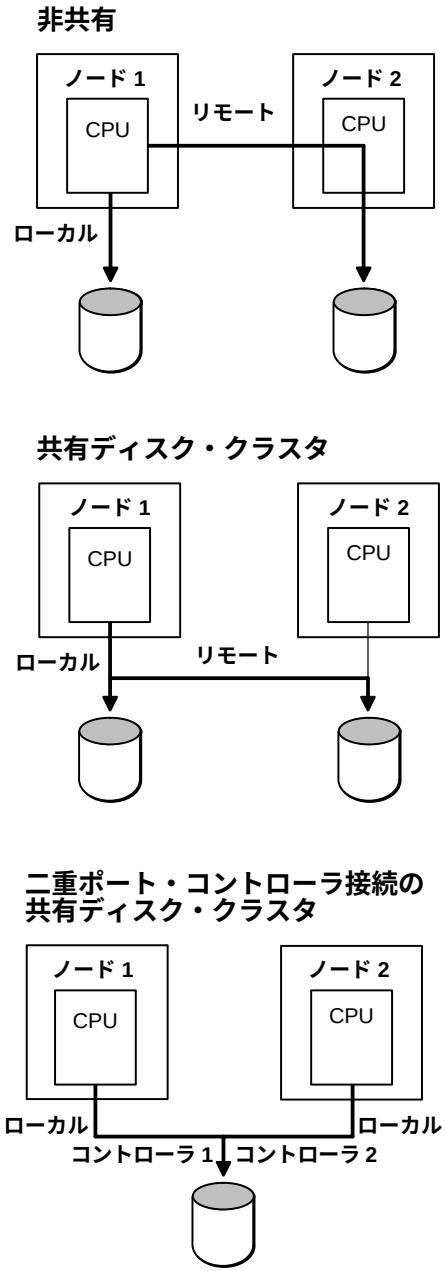
ディスク入出力

ローカル I/O は、リモート I/O（他ノードを経由する I/O）よりも実行速度が速くなります。リモート I/O の量が多いシステムでは、拡張性が低くなります。このような場合、データを分割してローカル・データにします。図 2-2 に、その違いを示します。

注意： 各種クラスタ化システムが、さまざまなハードウェア・ベンダーから発売されています。二重ポート・コントローラを接続した共有ディスク・クラスタでは、すべてのノードの待ち時間が同一になります。しかし、MPP（非共有）システムでは、そうでない場合があります。

図 2-2 で示した共有ディスク・アーキテクチャの詳細は、次の第 3 章「パラレル・ハードウェア・アーキテクチャ」を参照してください。

図 2-2 非共有ディスクと共有ディスクのローカル I/O とリモート I/O

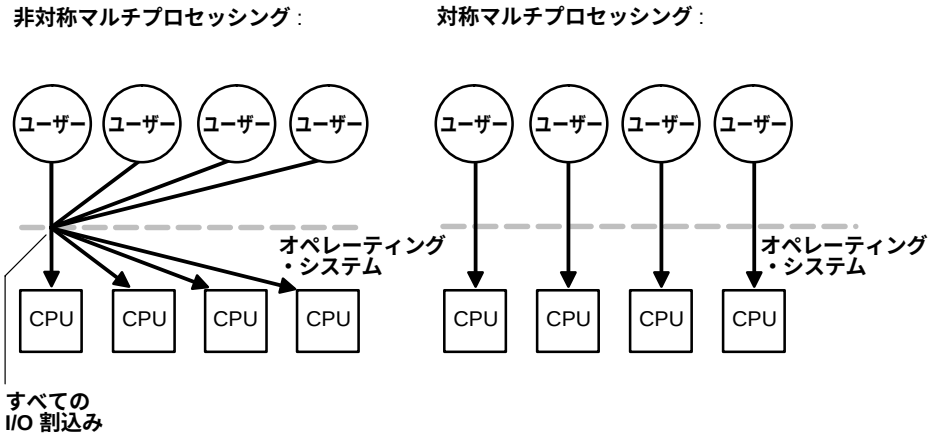


オペレーティング・システムの拡張性

システムの最終的な拡張性は、オペレーティング・システムの拡張性にも依存します。この項では、この要素の分析方法について説明します。

共有メモリー・システムを実現するノードが 1 つだけ（つまり、対称マルチプロセッサの 1 メモリーに複数の CPU が接続されたシステム）の場合、ソフトウェアの拡張性が重要なポイントになります。オペレーティング・システムでの同期のとり方が、システムの拡張性を決める要因になり得ます。たとえば、非対称マルチプロセッシングでは、1 つの CPU がすべての I/O 割込みを処理する必要があります。複数のユーザー・プロセスがオペレーティング・システムからリソースを要求するようなシステムについて考えてみます。

図 2-3 非対称マルチプロセッシングと対称マルチプロセッシング



非対称マルチプロセッシングでは、オペレーティング・システムが 1 度に 1 つのリソース要求しか処理できないため、ハードウェアの潜在的な拡張性を実現できません。要求がオペレーティング・システムに入力されるたびに、他の要求を排除するためのロックがかけられます。これとは対照的に、対称マルチプロセッシングではこのような制約は発生しません。

データベース管理システムの拡張性

パラレル・サーバー・アーキテクチャでは、そのアーキテクチャが内部並列か外部並列かということが重要な相違点になります。これは拡張性に大きな影響を及ぼします。重要な違いは、オブジェクト・リレーショナル・データベース管理システム（ORDBMS）が問合せをパラレル化するのか、または外部プロセスが問合せをパラレル化するのかということです。

ディスクの親和性によって、ノードがリモート・デバイスではなくローカル・デバイスに主にアクセスすることが保証されるので、パフォーマンスが改善されます。効率的な同期メカニズムによって、より一層のスピードアップとスケールアップが可能になります。

関連項目： 4-9 ページの「ディスクの親和性」、『Oracle8*i* チューニング』のパラレル実行に関する各章

アプリケーションの拡張性

アプリケーションの設計は、システムの他の要素の拡張性を利用する上で重要です。

注意： アプリケーションは、明示的にスケーラブルに設計する必要があります。

---

1 行しかない表に対してすべてのノードが更新する場合、ハードウェアおよびソフトウェア、データベースの拡張性が高くても、1 つのデータ・ブロックで同期します。一意の順序番号が生成されるプロセスについて考えてみます。

```
UPDATE ORDER_NUM
SET NEXT_ORDER_NUM = NEXT_ORDER_NUM + 1;
COMMIT;
```

この順序番号を更新するすべてのノードは、この表の同一行にアクセスする前に待機する必要があります。この状況は本質的にスケーラブルではありません。この場合、拡張性を改善するには、順序番号を使用します。

```
INSERT INTO ORDERS VALUES
   (order_sequence.nextval, ... )
```

この例では、順序番号を事前に割り当て、キャッシュに入れることで拡張性を改善できます。ただし、ビジネス・ルール（アプリケーション要件）のために、一部のアプリケーションを拡張できない場合があります。その場合は、ルール（要件）のコストを判断する必要があります。

注意： クライアントとサーバー・マシンは、スケーラブルな方法で接続する必要があります。それには、ネットワークもスケーラブルにする必要があります。

関連項目： 第 13 章「パラレル・サーバー用のデータベース設計」および第 12 章「アプリケーションの分析」

パラレル処理が有効な状況

この項では、パラレル・サーバーの利点を活用できるアプリケーションについて説明します。

- データ・ウェアハウジング・アプリケーション
- 異なるデータ・ブロックを更新するアプリケーション
- フェイルオーバーと高い可用性
- まとめ

データ・ウェアハウジング・アプリケーション

多くの場合、データ・ウェアハウジング・アプリケーションはデータを更新、挿入または削除する頻度が少ないため、Oracle Parallel Server（OPS）に適しています。問合せ処理集中型アプリケーションおよび更新アクティビティがあまり行われないアプリケーションでは、少ないオーバーヘッドで異なる複数のインスタンスからデータベースにアクセスできます。

データ・ブロックを変更しない場合は、複数のノードが同じブロックをバッファ・キャッシュに読み込むことができ、追加の I/O またはロック・オペレーションを行わずにそのブロックの問合せを実行できます。

意思決定支援アプリケーションは、データの更新がたまにしか行われないので、OPS に適しています。また、財務トランザクションのデータベースは、勤務時間帯のアクセスの大半が問合せであり、オフピークの時間帯にデータを更新するので、これも OPS に適しています。

異なるデータ・ブロックを更新するアプリケーション

更新するデータ・ブロックが異なるか、または同一のデータ・ブロックを異なるタイミングで更新するアプリケーションもパラレル・サーバーに適しています。このような例として、各ユーザーが別々の表を所有して使用するタイム・シェアリング環境があげられます。

バッファ・キャッシュに入っているブロックを更新する必要があるインスタンスは、これらのブロックを変更している間、1 つまたは複数のインスタンス・ロックを排他モードで維持する必要があります。このようなインスタンス・ロックの競合を減らすには、パラレル・サーバーとそのサーバー上で実行するアプリケーションを調整する必要があります。そのためには、各インスタンスとアプリケーションでのデータの使用方法を計画し、それに応じて各表を分割します。

分割データを処理する OLTP

異なるデータのセットを変更するオンライン・トランザクション処理アプリケーションでは、パラレル・サーバーを最大限に活用できます。このような例として、支店（ノード）が所有する各自の口座にはアクセスするが、他の支店の口座にはたまにしかアクセスしない銀行支店システムがあります。

大規模データベースにランダムにアクセスする OLTP

データベースにランダムにアクセスするアプリケーションでも、パラレル・サーバーの利点を活用できます。ただし、パラレル・サーバーの使用が有効となるのは、データベースがどのノードのバッファ・キャッシュよりも著しく大きい場合だけです。このような例として、個々のレコードが同時に複数のノードからアクセスされることのない陸運局のシステムがあります。また、アーカイブされた税金のレコードやリサーチ・データもこの例としてあげられます。これらの例では、インスタンスがデータブロックへの排他アクセス権限を持っていても、ほとんどのアクセスは I/O が許可されます。この種のアプリケーションでは、ファイン・グレイン・ロックなどの Oracle 機能を使用すると、パフォーマンスが大幅に向上します。

部門別処理アプリケーション

1 つのデータベースに入っている複数の異なる表を変更するアプリケーションも、OPS に適しています。このような例として、3 つのノードがそれぞれ専用ノードとして在庫処理、人事処理、売上げ処理を個別に処理するシステムがあります。この場合、管理するデータベースは 3 つではなく 1 つだけです。

フェイルオーバーと高い可用性

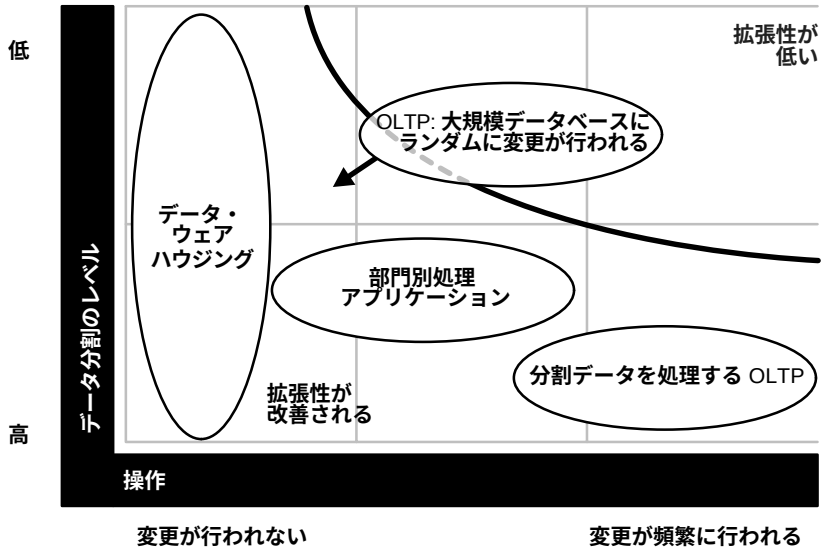
高い可用性を要求するアプリケーションでは、Oracle Parallel Server のフェイルオーバー機能が有利に働きます。1 つのインスタンスでのデータベースとの接続が失われた場合は、他のインスタンスを使って自動的に再接続するようにアプリケーションを作成できます。

まとめ

図 2-4 に、各種のアプリケーションの相対的な拡張性を示します。左端の楕円で示されたデータ・ウェアハウジング・アプリケーションは、他のアプリケーションに比べて更新が少なく、データ分割のレベルが高いので、拡張性に優れています。また、分割データを処理する OLTP および部門別処理アプリケーションは、データを頻繁に更新する場合でも拡張性に優れています。

大規模データベースに対してランダムに変更を加える OLTP アプリケーションは、パラレル・サーバーに不適切なアプリケーションとして以前はみなされていました。ところが、インターコネクトによるノード間通信の進歩とともに、これらのアプリケーションの拡張性は改善されています。特に、一方のインスタンスで変更された表を他方のインスタンスが読み込むといったアプリケーションでは、拡張性が大きく改善されました。このような構成では、前のリリースよりも拡張性が大幅に向上しています。

図 2-4 アプリケーションの拡張性



パラレル処理が有効でない状況

次のガイドラインでは、パラレル処理が有効でない状況について説明しています。

- 通常、同期のコストが高すぎるためにスループットが低下する場合は、パラレル処理は有効ではありません。

  多数のノードから多数のユーザーが小さなデータ・セットを変更する場合、非常に多数の同期が実行されます。ただし、これらのユーザーがデータを読み込むだけの場合は、同期は必要ありません。

- 1 つのブロックまたは 1 つの行に対してインスタンス間で競合が発生する場合、パラレル処理は有効ではありません。

  たとえば、主に順序番号生成の手段として、1 つの行だけの表を使用するのは、効率的ではありません。どのプロセスでも行の選択、更新、解放を順番に実行するため、このような表はボトルネックになります。

効率的な処理の分割方法のガイドライン

この項では、同期を減らしてシステムのパフォーマンスを改善する分割方法を選択するための一般的なガイドラインを示します。

- 概要
- 垂直方向の分割
- 水平方向の分割

概要

機能や位置などに応じて処理の 3 つの要素を分割し、これらの要素が相互に影響を及ぼさないようにできます。この 3 つの要素を次に示します。

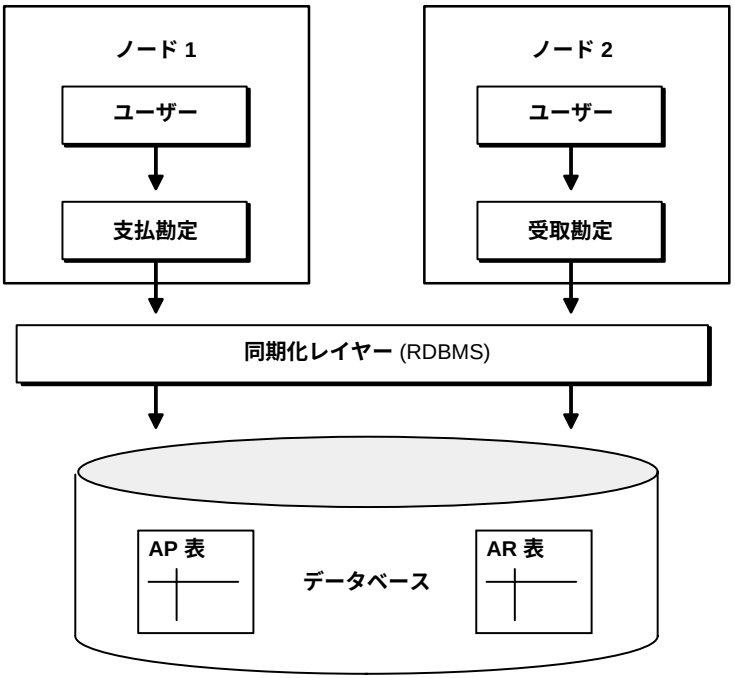
- ユーザー
- アプリケーション
- データ

データにアクセスするユーザー・グループに基づいてデータを分割します。アプリケーションの場合は、同一データにアクセスするグループ別に分割します。また、地域別のデータ分割やデータを使用するユーザーの所在による分割も考慮します。

垂直方向の分割

垂直方向の分割により、同期をあまり実行せずに多数のタスクを多数のリソース上で実行できます。図 2-5 に、垂直方向の分割を示します。

図 2-5 垂直方向の分割



この例では、ユーザー、アプリケーション、データが会社の支払勘定機能と受取勘定機能に分割されています。支払勘定機能と受取勘定機能は個別のノードに入っています。この例では、ほとんどの同期が 1 つのノードで発生するため、非常に効率的です。ローカル・ノードでの同期は、ノード間の同期よりもコストが低くなります。

リソース・サブセットに対してタスクを分割して、同期を減らします。タスクを分割すると、小さいタスク・セットによる共有リソースへのアクセスが必要になります。

水平方向の分割

水平方向の分割の概念を示すために、図 2-6 に在庫表の行を示します。OPS に 4 つのインスタンスがあり、それぞれが専用のノードに置かれている場合は、各インスタンスがデータのサブセットだけにアクセスするように分割します。

図 2-6 水平方向の分割

ノード 1	ノード 2	ノード 3	ノード 4
行 1 ～ 10	行 11 ～ 20	行 21 ～ 30	行 31 ～ 40

この例では、インスタンスがそれぞれ別の行セットにアクセスするため、必要な同期は非常に少なくなります。同様に、所在位置によって分割されたユーザーは、他のデータを使用せずに特定のデータのみにアクセスすることが多くなります。複数のユーザーが同じデータにアクセスしない場合は、同期の必要性が非常に少なくなります。

関連項目： 分割の詳細は、『Oracle8*i* チューニング』を参照してください。

一般的なパラレル処理の誤解

パラレル処理に関するさまざまな誤解から、パラレル処理に対して非現実的な期待が持たれることがあります。次の点を考慮してください。

- パラレル処理に切り替えても、自動的に期待どおりの効果が得られるわけではありません。優れたパラレル処理を実現するには、かなりの時間をかけてデータベースの設計とアプリケーションのチューニングを行う必要があります。
- 拡張性を確定する要因は、システム内のノードまたは CPU の数だけではなく、インターコネクトの帯域幅と待ち時間、同期の量とコストも含まれます。

  一部のアプリケーションでは、1 回の同期（ホットショット）のコストが非常に高いために問題となることがあります。一方、競合の少ないデータで同期を多数実行できるアプリケーションもあります。
- パラレル処理を実行しても、自動的に可用性が改善されるわけではありません。可用性が改善されるかどうかは、システム・アーキテクチャに依存します。

  たとえば、一部の MPP システムでは、CPU の 1 つが使用不能になるとマシン全体が使用不能になります。これと対照的に、クラスタではノードの 1 つが使用不能になっても、他のノードは引き続き使用できます。これは、MPP システムでも同じです。
- すべてのアプリケーションが効率よく拡張性を改善できるように設計されているわけではありません。

3

パラレル・ハードウェア・アーキテクチャ

Oracle Parallel Server（OPS）は、さまざまなアーキテクチャで実行できます。この章では、パラレル・サーバーを使用した各種ハードウェア・システムについて説明し、それぞれの長所と短所を示します。

- 概要
- ハードウェア要件およびオペレーティング・システム・ソフトウェア要件
- 共有メモリー・システム
- 共有ディスク・システム
- 非共有システム
- 非共有 / 共有ディスク混合システム

概要

この項では、次の項目について説明します。

- パラレル処理ハードウェア・システム
- アプリケーション・プロファイル

Oracle 構成では、マシン内部およびマシン間、ノード間のパラレル処理をサポートしています。OPS を単一インスタンスで構成されたシングル・ノードで実行すると、オーバーヘッドが生じるだけで利点はありません。標準の Oracle では、共有メモリー・システムで特別な操作を行うことなく一部のパラレル処理機能を利用することができます。

このマニュアルでは、非共有 / 共有ディスク・アーキテクチャでの OPS を中心に説明していますが、このマニュアルで扱っているアプリケーション設計の題材は、標準の Oracle システムにも関連のある内容です。

パラレル処理ハードウェア・システム

パラレル処理ハードウェア・システムは、共有する特定のリソース別に分類されます。この章では、次のカテゴリについて説明します。

- 共有メモリー・システム
- 共有ディスク・システム
- 非共有システム

これらのシステムは、ノード間の通信方法によって、" 密結合 " または " 疎結合 " と表現することもできます。

Oracle では、これらのパラレル処理システムをすべてサポートします。ただし、非共有システムでは、あるノードが別のノードのディスクにアクセスできることが前提になっています。たとえば、IBM SP2 は仮想共有ディスク機能を備えています。この場合、ディスクはソフトウェアを介して共有されます。

注意 : サポートされる Oracle 構成は、プラットフォームごとに異なります。使用する構成がプラットフォームでサポートされている構成かどうかを確認してください。

アプリケーション・プロファイル

オンライン・トランザクション処理（OLTP）アプリケーションは、対称マルチプロセッサ・システム（SMP）で実行するとパフォーマンスが最も高くなる傾向があります。適切な分割がされていれば、クラスタおよび大規模パラレル処理システム（MPP）でも良好なパフォーマンスが得られます。意思決定支援（DSS）アプリケーションは、SMP およびクラスタ、MPP で実行すると高いパフォーマンスが得られる傾向があります。アプリケーションを実行するための必要な性能を備えたシステムを選択します。

ハードウェア要件およびオペレーティング・システム・ソフトウェア要件

ハードウェアのベンダーはそれぞれ独自の方法でパラレル処理を実現しますが、OPS に必要な共通要素は次のとおりです。

- 高速なインターコネクト
- グローバル・アクセス可能ディスク・サブシステムまたは共有ディスク・サブシステム

高速なインターコネクト

ロック・マネージャおよびクラスタ・マネージャの通信で使用する、高帯域幅で、待ち時間の短いノード間通信機能です。インターコネクトとして、イーサネット、FDDI、その他の専用インターコネクト方式を使用できます。主インターコネクトに障害が発生しても、通常はバックアップのインターコネクトを使用できます。バックアップのインターコネクトがあるため、高い可用性が保証され、一箇所で障害が発生しても他に影響を与えません。

グローバル・アクセス可能ディスク・サブシステムまたは共有ディスク・サブシステム

疎結合システムまたは大規模パラレル処理システムでは、すべてのノードが共有ディスクに同時にアクセスできます。このため、Oracle8*i* の複数のインスタンスは、データベースに同時にアクセスできます。このような共有ディスク・サブシステムは、共有 SCSI または 2 ポート SCSI（UNIX での一般的な SCSI）を介して実現する方法が最も一般的です。IBM SP などの一部の MPP プラットフォームでは、ディスクがノードに対応付けられており、仮想共有ディスク・ソフトウェア・レイヤーによってすべてのノードがグローバル・アクセスを実行できるようになります。

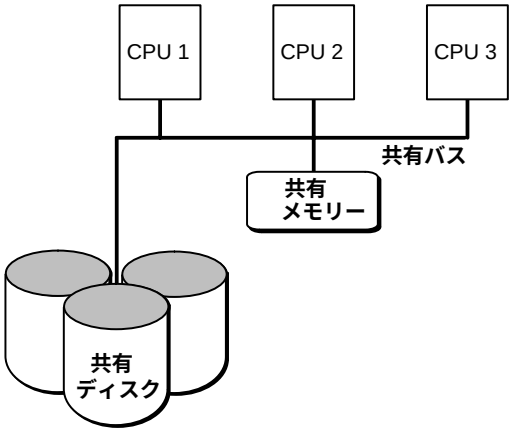
注意： 統合分散ロック・マネージャ（IDLM）は、データ・ブロックの変更、キャッシュの一貫性の維持、障害が発生したノードの回復、トランザクション・ロック、ディクショナリ・ロック、SCN ロックを調整します。

共有メモリー・システム

密結合の共有メモリー・システムを図 3-1 に示します。このシステムの特徴は、次のとおりです。

- 複数の CPU がメモリーを共有する。
- それぞれの CPU が、すべての共有メモリーに共通バスを介して完全にアクセスできる。
- ノード間通信は共有メモリーを使って行われる。
- パフォーマンスはメモリー・バスの帯域幅によって制限される。

図 3-1 密結合共有メモリー・システム



対称マルチプロセッサ（SMP）マシンは、クラスタ内の 1 ノードとして構成されることがよくあります。密結合システムでは、OPS で処理する複数の SMP ノードをインストールできます。この場合、複数の CPU がメモリーを共有し、すべての CPU がメモリー・バスを介してメモリーにアクセスできます。密結合システムには、Pyramid、Sequent、Sun SparcServer などがあります。

OPS を単一の SMP マシン上で稼働させると、IDLM アクセスによって不要なオーバーヘッドが多量に発生するため、このような稼働方法は意味がありません。

密結合システムでは、さまざまな要因によりパフォーマンスが制限されます。このような要因には、メモリーの帯域幅、CPU 間通信の帯域幅、システムで使用できるメモリー、I/O の帯域幅、共通バスの帯域幅などが含まれます。

パラレル処理に関する共有メモリー・システムの利点は、次のとおりです。

- メモリー・アクセスは、ノード間通信よりもコストが安い。つまり、ロック・マネージャを使用した同期よりも内部同期の方が高速である。
- 共有メモリー・システムは、クラスタよりも容易に管理できる。

パラレル処理に関する共有メモリー・システムの短所は、次のとおりです。

- バスの帯域幅と待ち時間、および使用可能メモリーの上限によって拡張性が制限される。

共有ディスク・システム

通常、共有ディスク・システムは疎結合されています。図 3-2 に、疎結合共有ディスク・システムを示します。このシステムの特徴は、次のとおりです。

- それぞれのノードは、1 つまたは複数の CPU とこれらの CPU に対応付けられたメモリーで構成される。
- メモリーはノード間で共有されない。
- 共通高速バスを介して通信が行われる。
- それぞれのノードが、同一ディスクにアクセスし、その他のリソースにもアクセスする。
- ハードウェアで SMP がサポートされている場合、SMP をノードとして使用できる。
- 高速バスの帯域幅によって、システムのノードの数（拡張性）が制限される。

図 3-2 疎結合共有ディスク・システム

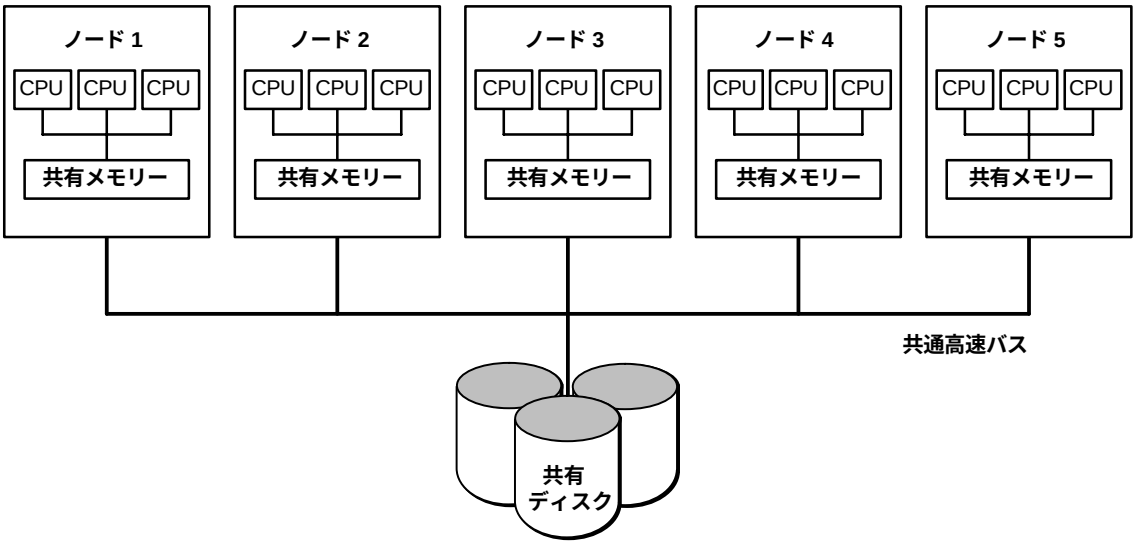


図 3-2 に示すクラスタは、複数の密結合ノードで構成されています。この場合、IDLM が必要です。疎結合システムには、VAX clusters または Sun clusters などがあります。

ノード間でメモリーを共有していないため、それぞれのノードに固有のデータ・キャッシュがあります。ノード間でキャッシュの一貫性を維持しなければならないため、ロック・マネージャが必要となります。さらに、クラスタ内のすべてのノードが同一データを参照できるようにするため、Oracle レベルでの IDLM によるインスタンス・ロックが必要です。

ロックを維持し、データ・キャッシュの一貫性を保つことで、追加のオーバーヘッドが発生します。パフォーマンスに及ぶ影響は、ノード間通信を行う高速バスの帯域幅、IDLM のパフォーマンスなど、ハードウェアおよびソフトウェアの要素に依存します。

パラレル処理に関する共有ディスク・システムの利点は、次のとおりです。

- 共有ディスク・システムでは、高い可用性が実現する。1 つのノードが使用不能になっても、引続きすべてのデータにアクセスできる。
- 共有ディスク・システムの概念は、データベースが 1 つということ。これは、非共有システムよりも有利である。
- 共有ディスク・システムは増加拡張できる。

パラレル処理に関する共有ディスク・システムの不利な点は、次のとおりです。

- ノード間の同期が必要なため、IDLM オーバーヘッドが発生し、高速なインターコネクトが必要となる。
- 作業負荷を適切に分割できないと、同期のオーバーヘッドが高くなることがある。
- 共有ディスク・ソフトウェアを実行する際に、オペレーティング・システムのオーバーヘッドが発生する。

非共有システム

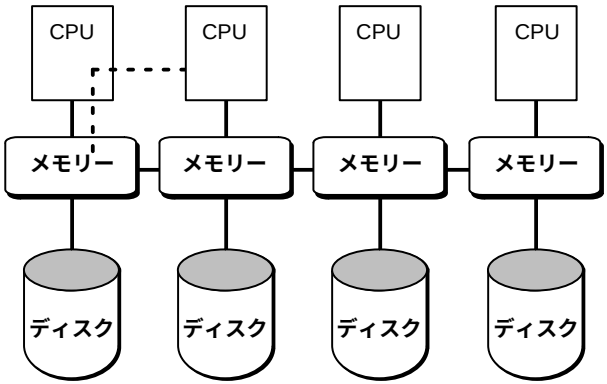
通常、非共有システムは疎結合されています。この項では、次の項目について説明します。

- 非共有システムの概要
- 大規模パラレル処理システム
- 非共有システムのまとめ

非共有システムの概要

非共有システムでは、それぞれのディスクは、1 つの CPU に接続されています。表またはデータベースがこのディスクに入っている場合、これらのデータにアクセスできるかどうかはこのディスクが接続している CPU に依存します。図 3-3 に、非共有システムを示します。

図 3-3 非共有システム



非共有システムでは、メモリーへのアクセスではなくディスクへのアクセスが重要です。ただし、CPU とディスクを増設することで拡張性を改善できます。どのディスクにでもどの CPU からも透過的にアクセスできるオペレーティング・システムであれば、OPS は、非共有システムのディスクにアクセスできます。ただし、このアクセスは待ち時間の点でコストがかかります。

大規模パラレル処理システム

大規模パラレル処理（MPP）システムの特性は、次のとおりです。

- MPP システムのサイズは 2 ～ 3 のノード構成から数千ものノード構成までが可能である。
- それぞれのノードのプロセッサが低コストであるため、プロセッサあたりのコストが非常に低い。
- それぞれのノードに非共有メモリーが対応付けられている。
- それぞれのノードが各自のデバイスを備えているが、障害発生中に、他のノードが障害発生ノードのデバイスにアクセスできる。
- ノードは格子状または網状、ハイパーキューブ状に接続される。
- どのノードにも Oracle インスタンスを常駐させることができる。

システムの特性で示しているように、MPP システムは数千におよぶノードから構成することもできます。それぞれのノードに固有の Oracle インスタンスを導入し、インスタンスのすべての標準機能を備えることができます。（Oracle インスタンスは、システム・グローバル領域とすべてのバックグラウンド・プロセスで構成されます。）

MPP では、各ノードに固有のメモリーを持っているため、データベース操作（ソートやバッファ・キャッシュなど）を実行する場合に大量の実メモリーにアクセスできます。ディ

スク I/O を行わないようにするために実行時間の長い問合せおよびソートでは、この利点が重要になります。MPP システムのメモリーの合計量が 2GB を超えることがあるので、アドレス領域の制限が 2GB の 32 ビット・マシンでは、この利点は使用できません。疎結合システムの場合と同様、MPP でもシステム内のすべてのノード間でキャッシュの一貫性を維持する必要があります。このため、キャッシュの管理にオーバーヘッドがかかります。MPP システムには、nCUBE2 Scalar Supercomputer、Unisys OPUS、Amdahl、Meiko、Pyramid、Smile および IBM SP などがあります。

非共有システムのまとめ

この項では、パラレル処理に関する非共有システムの利点と短所について説明します。

利点

- 増加拡張できる。
- システムの拡張性に実質的な限度はない。
- MPP は一般に、DSS アプリケーションなど、読込み専用のアプリケーションに適している。
- 障害が発生しても、その影響は局所的なものにとどまる。あるノードで障害が発生しても、他のノードは使用可能な状態のままである。

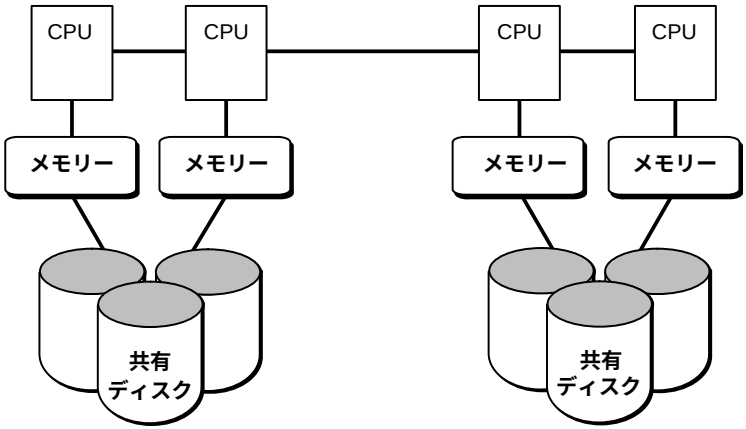
短所

- 必要な調整が多くなる。
- 別のノードに属するディスクに処理を実行しているプロセスに余分なオーバーヘッドが要求される。
- オンライン・トランザクション処理システムのように、更新または挿入の負荷が高い場合は、競合を減らすためにデータ依存ルーティングを考慮する必要がある。

非共有 / 共有ディスク混合システム

混合システムを使うと、たいへん有利なことがあります。混合システムでは、非共有ディスクと共有ディスクの両方の利点が結合され、それぞれのシステムの制限を相互に補うことができます。図 3-4 に、混合システムを示します。

図 3-4 2 つの共有ディスク・システムで構成された非共有システム



この図では、2 つの共有ディスク・システムを接続し、非共有システムと同一のハードウェア冗長性を持つシステムを構成しています。1 つの CPU で障害が発生しても、他の CPU は引き続きすべてのディスクにアクセスできます。

第 II 部

Oracle Parallel Server の概要

4

Oracle でのパラレル処理のインプリメントの方法

この章では、Oracle Parallel Server（OPS）がパフォーマンスの高いパラレル処理を実現する方法について概要を説明します。主な説明事項には次のものがあります。

- パラレル・サーバーの有効化と無効化
- 同期
- 高パフォーマンス機能
- キャッシュの一貫性

関連項目： 第 7 章「ロック・メカニズムの概要」（Oracle 内でのロックの階層について）

パラレル・サーバーの有効化と無効化

OPS は、有効化にすることも無効化にすることもできます。

Oracle + オプション	パラレル・サーバー無効化	Parallel Server 有効化	
		単一ノード	複数ノード
OPS のインストールなし	可 : デフォルト	不可	不可
OPS インストール済み	可 : デフォルト	可 : 単一共有	可 : 複数共有

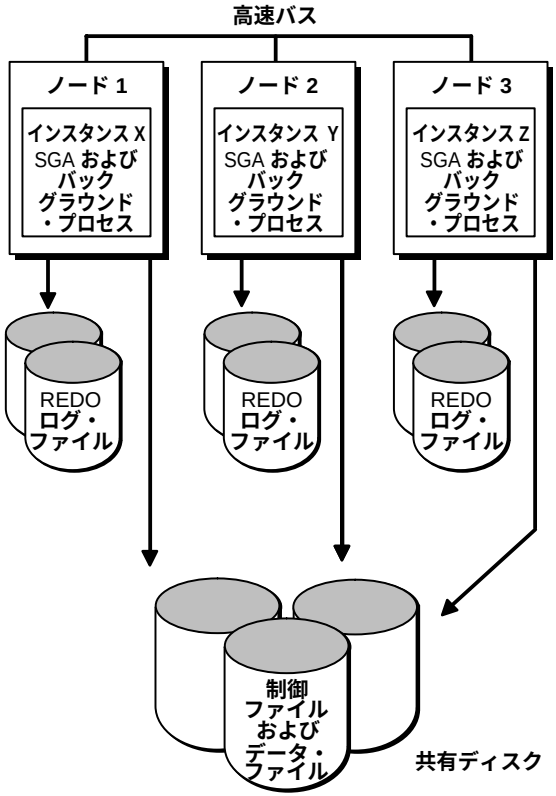
パラレル・サーバーを無効化にした場合は、1 つの Oracle インスタンスだけがデータベースをマウントまたはオープンできます。このモードは、データベースを作成する場合、およびデータベースの完全回復を行う場合に必要です。Oracle の標準機能だけで現在のニーズを満

たせる場合は、OPS をインストールして無効にしておくと役立ちます。あとで必要になったときに OPS を有効にできます。

OPS を有効にした場合は、パラレル・サーバーの 1 つ以上のインスタンスが、同一のデータベースをマウントします。すべてのインスタンスが同一のデータベースをマウントし、同一のデータ・ファイルに対して読込みおよび書込みを行います。単一共有モードは、1 つのインスタンスだけで実行する OPS 構成です。グローバル操作もありますが、この場合は必要ありません。リソース獲得のための競合はありませんが、インスタンスの動作はクラスタ構成の場合と同じであり、統合分散ロック・マネージャ（IDLM）のオーバーヘッドなどが伴います。複数共有モードは、複数のインスタンスを実行する OPS 構成です。

図 4-1 に、標準的な OPS 構成を示します。この構成では、それぞれ異なるノード上で 3 つのインスタンスが同じデータベースにアクセスしています。

図 4-1 ディスクを共有する共有モード



注意： それぞれのインスタンスは、他のインスタンスの REDO ログ・ファイルにアクセスできます。

関連項目： 18-12 ページの「複数インスタンス上でのユーザー認証のためのパスワード・ファイルの使用方法」

同期

ノード間の同期は、標準の Oracle では特に考慮する必要はありません。しかし、OPS を使用するときは、同期が必要な局面について幅広く理解している必要があります。これには、次のような事項が含まれます。

- ブロック・レベル・ロック
- 行レベル・ロック
- 領域管理
- システム変更番号

OPS の排他モード（1 つのデータベースに 1 つのインスタンスだけがマウントするモード）では、すべての同期はインスタンス内部で行われます。共有モードでは、IDLM コンポーネントを利用して同期が行われます。

ブロック・レベル・ロック

インスタンス間でのブロック・アクセスはブロック単位で行われます。あるインスタンスが排他モードであるブロックをロックしているときは、他のインスタンスはそのブロックにアクセスできません。Oracle は、データベースからのブロックの読込みが必要になるたびに、インスタンス・ロックと呼ばれるロックを取得する必要があります。ロックの所有権はブロックを読み込むインスタンスに割り当てられます。

OPS は複数のメモリーを備えた環境で実行されるので、それぞれのインスタンスのメモリー内に同じブロックのコピーが重複して存在する可能性があります。IDLM を使用したノード間同期によって、同じブロックのすべてのコピーの妥当性が保証されます。

ブロック・レベル・ロックは、OPS が有効になっている場合にだけ行われます。ブロック・レベル・ロックは、ユーザーおよびアプリケーションに対して透過的です。（行レベル・ロックは、OPS が有効または無効のどちらの状態であっても動作します。）

関連項目： 第 9 章「パラレル・キャッシュ管理インスタンス・ロック」

行レベル・ロック

OPS および標準の（単一インスタンス）Oracle には、バッファ・キャッシュでのブロック・レベル・ロックだけでなく、行レベル・ロックの機能もあります。さらに明確に言えば、行レベル・ロックはブロック内に格納されます。

次の例について考えてみてください。インスタンス 1 は、行 1 を更新するためにファイル 2 のブロック 10 を読み込みます。インスタンス 2 も、行 2 を更新するためにファイル 2 のブロック 10 を読み込みます。この場合、インスタンス 1 は、ブロック 10 のインスタンス・ロックを獲得し、行 1 をロックして更新します（UPDATE 文なので、行ロックは暗黙的に行われます）。

インスタンス 2 は、インスタンス 1 に更新済みのブロックをディスクに書き込ませます。インスタンス 1 はブロック 10 のロックの所有権を放棄するので、インスタンス 2 がその所有権を取得できるようになります。次に、インスタンス 2 は行 2 をロックして UPDATE を実行します。

領域管理

OPS の領域管理は、空きリストおよび空きリスト・グループを使うことで効率的に処理できます。

領域管理の要点は、挿入時の領域の割当て方法についての問題点を考えるとよくわかります。たとえば、ある表でさらに大きな領域が必要になった場合、他のユーザーが同じ領域を使用していないことを確認するにはどうすればよいでしょうか。また、同じディスクの同じファイルの同じ領域に対する挿入操作を、2 つのノードが同時に実行しないようにするにはどうすればよいでしょうか。

次の例について考えてみてください。インスタンス 1 は、行を挿入するためにファイル 2 のブロック 10 を読み込みます。インスタンス 2 も、別の行を挿入するためにファイル 3 のブロック 20 を読み込みます。どちらのインスタンスも、それぞれに行を挿入します。これらの挿入操作への領域割当てをすべて 1 つのブロックが請け負うと、そのブロックはインスタンス間で頻繁に ping と呼ばれるインスタンス間でのブロックの受け渡しが発生することになります。つまり、インスタンス 2 で挿入が必要になるとインスタンス 1 は所有権を失い、逆の場合はインスタンス 2 が所有権を失うという状態の繰返しが起きます。この状況では頻繁に競合が発生し、パフォーマンスが低下します。

これに対して、空きリスト・グループを使用すれば、効率よく領域を管理できます。2 つのインスタンスが同じオブジェクト（表など）に挿入する場合でも、各インスタンスがそのオブジェクト用の専用の空きリストのセット（空きリスト・グループ）を持っていれば、同じブロックをめぐる競合は起こりません。各インスタンスは、そのオブジェクトに属する異なるブロックに挿入することになります。

システム変更番号

標準の Oracle では、システム変更番号（SCN）の維持および増分処理はインスタンスが SGA 内で行います。OPS の共有モードでは、SCN をグローバルに維持する必要があります。SCN のインプリメント方法は、プラットフォームごとに異なります。SCN は、IDLM または Lamport SCN スキームで処理するか、ハードウェア・クロックまたは SCN 専用サーバーを使って処理できます。

関連項目： 4-8 ページの「Lamport SCN 生成」、10-4 ページの「システム変更番号（SC）」、およびシステム固有の Oracle マニュアル

高パフォーマンス機能

OPS を使用すると、リソースを共有している複数のプロセッサで構成されるシステムの利点を活用でき、そのためにトランザクション処理機能が犠牲になることもありません。次の各項では、OPS のパフォーマンスを最適化するいくつかの機能について説明します。

- 高速コミット、グループ・コミットおよび遅延書込み
- 行ロックおよび複数バージョン読込みの一貫性
- オンライン・バックアップおよびアーカイブ
- 順序番号ジェネレータ
- Lamport SCN 生成
- 空きリスト
- 空きリスト・グループ
- ディスクの親和性

標準の（単一インスタンス）Oracle は、メモリー内のバッファ・キャッシュを使用するので、データベース操作に必要なディスク I/O による負荷が減少します。パラレル・サーバーでは、各ノードに他のノードとは共有しない専用メモリーがあるので、OPS は異なるノードのバッファ・キャッシュを調整し、しかもパフォーマンス低下の原因となるその他のディスク I/O を最小限に抑える必要があります。Oracle のパラレル・キャッシュ管理テクノロジでは、Oracle の高パフォーマンス機能が維持されると同時に、複数のバッファ・キャッシュの調整も行われます。

関連項目：『Oracle8*i* 概要』（高パフォーマンスを提供する各機能について）

高速コミット、グループ・コミットおよび遅延書込み

高速コミット、グループ・コミットおよび遅延書込みはインスタンス毎に機能し、排他モードでも共有モードでも同様に機能します。

Oracle は、データの必要なインスタンスがあると、そのインスタンスのバッファ・キャッシュにないデータ・ブロックだけをディスクから読み込みます。データ・ブロックの書込みは遅延されるので、複数のトランザクションからの変更が同じデータ・ブロックに含まれていることがよくあります。

最適化を計るため、Oracle は次のような場合にだけ変更済みのデータ・ブロックをディスクに書き込みます。

- しばらく使用されていないデータ・ブロックがあり、新しいデータ用のバッファ・キャッシュ領域が必要になったとき（共有モードまたは排他モード）
- チェックポイントの時点（共有モードおよび排他モード）

- 他のインスタンスでブロックが必要な場合（共有モードのみ）

Oracle は、強制読込みまたは強制書込みが原因でディスクに不要な書込みを行う場合もあります。

関連項目： 15-14 ページの「ファイルの妥当性のチェック」

行ロックおよび複数バージョン読込みの一貫性

Oracle の行ロック機能を使用すると、異なるノード上にある複数のトランザクションが、同じデータ・ブロックの異なる行をロックして更新でき、どのトランザクションも、他のトランザクションのコミットを待つ必要はありません。行を変更しても、まだコミットしていない時点では、変更前の行の値がすべてのインスタンスの読込みアクセスに対して提供されます。これを複数バージョン読込みの一貫性といいます。

オンライン・バックアップおよびアーカイブ

OPS では、標準の Oracle でサポートしているすべてのバックアップ機能をサポートします。データベース全体または個々の表領域をオンラインまたはオフラインでバックアップできます。

ARCHIVELOG モードで Oracle を使用している場合は、オンライン REDO ログ・ファイルをアーカイブした時点で、そのファイルへの上書きができるようになります。OPS では、各インスタンスで専用の REDO ログ・ファイルを自動アーカイブすることも、1 つまたは複数のインスタンスですべてのインスタンスの REDO ログ・ファイルを手動でアーカイブすることもできます。

ARCHIVELOG モードでは、オンラインおよびオフラインの両方でバックアップを作成できます。NOARCHIVELOG モードで Oracle を実行している場合は、オフライン・バックアップだけです。本番データベースについては ARCHIVELOG モードでの運用をお薦めします。

キャッシュ・フュージョン

キャッシュ・フュージョンは、インスタンス間の通信を改善し、読取りアクセスと書込みアクセスに伴うキャッシュ一貫性矛盾のために発生する ping を削減します。キャッシュ・フュージョンでは、インスタンスがブロックの更新を要求した場合、そのブロックが他のインスタンスによって保持されているときは、ブロックの一貫性読込みコピーを作成して要求元インスタンスへ直接送るので、ping は発生しません。キャッシュ・フュージョンでは、ブロックを保持しているインスタンスのメモリー・キャッシュから要求元のインスタンスのメモリー・キャッシュへブロックを直接コピーします。

関連項目： キャッシュ・フュージョン・アーキテクチャの詳細は、5-6 ページの「キャッシュ・フュージョン処理およびブロック・サーバー・プロセス」を参照してください。キャッシュ・フュージョンで使用されるロック・メカニズムの詳細は、20-2 ページの「キャッシュ一貫性矛盾を解決するときのキャッシュ・フュージョンの役割」を参照してください。

順序番号ジェネレータ

OPS を使用すると、複数のインスタンスの複数のユーザーが一意の順序番号を生成するときに、インスタンス間での協調動作および競合を最小限に抑えることができます。

順序番号ジェネレータを使用すると、複数のインスタンスから順序にアクセスして番号を増加できます。したがって、インスタンス間に順序番号の競合は起きず、トランザクションのコミットを待つ必要もありません。各インスタンスで専用の順序キャッシュを使用して、順序番号へのアクセスの高速化を図ることもできます。OPS では、IDLM ロックによってインスタンス間の順序が調整されます。

Lamport SCN 生成

システム変更番号（SCN）は、1 つのインスタンス内または全インスタンスを通じて、イベントを順序付けるために Oracle が使用する論理タイムスタンプです。たとえば、Oracle は各トランザクションに SCN を割り当てます。概念的には、SCN を生成するグローバル・シリアル・ポイントがあります。ところが実際には、並行して SCN の読込みおよび生成ができます。SCN 生成スキームの 1 つに、Lamport SCN 生成スキームがあります。

Lamport SCN 生成スキームは、全インスタンスで並列に SCN を生成する、高速で拡張性のあるスキームです。このスキームでは、ロック・メッセージを含めて、全インスタンス間のすべてのメッセージが SCN をピギーバック方式で運びます。ピギーバック方式で運ばれる SCN は、Oracle 内の因果関係を伝播します。因果関係に対するこのような配慮が存在するかぎり、複数のインスタンスは並列に SCN を生成でき、インスタンス間に特別なコミュニケーションは必要ありません。

プラットフォーム固有のしきい値よりも MAX_COMMIT_PROPAGATION_DELAY の方が大きい場合、Oracle では通常 Lamport SCN 生成スキームを使用します。プラットフォーム固有のしきい値は、通常ではデフォルト値が使用されます。標準的な設定値は 7 秒です。インスタンスの起動後にアラート・ログを調べれば、Lamport SCN 生成スキームの使用状況を知ることができます。

関連項目： システム固有の Oracle マニュアル

空きリスト

標準 Oracle では、ブロックに対する競合を緩和するための手段として複数の空きリストを使用できます。空きリストは、エクステント内の空き領域のあるデータ・ブロックのリストです。空き領域のあるデータ・ブロックは、表やクラスタなどのデータベース・オブジェクトに対して挿入または更新が行われるときに使用されます。この方法では、異なるインスタンスのトランザクションから同じ表にデータを挿入するときでも、インスタンス間に競合が起きることはありません。1 つまたは複数のインスタンスに対応付けられた空き領域リストを使用して、新しい行を挿入するための空き領域を見つけるからです。空きリストは、ブロックの共通プールから取り出すことも、複数の空きリストを分割してファイル内の個々のエクステントをそれぞれ特定のオブジェクトに割り当てることもできます。

単一空きリストを使用しているときに複数の挿入が行われると、複数のプロセスが単一空きリストから領域を割り当てようとするために、直列化が起きます。複数の空きリストを使用する利点は、領域が必要になったときにプロセスが特定のブロック・プールを検索できることです。これにより、空き領域獲得のためのユーザー間の競合が緩和されます。

関連項目： 第 11 章「領域管理および空きリスト・グループ」

空きリスト・グループ

OPS では、空きリスト・グループを使用して、空きリストを含む単一ブロックにアクセスするインスタンス間の競合を排除できます。デフォルトで使用可能な空きリスト・グループは 1 つだけです。したがって、オブジェクトのすべての空きリストは、セグメント・ヘッダー・ブロックに常駐します。

OPS 環境で単一ブロックに複数の空きリストが存在すると、空きリストが存在するブロックの ping または強制読込み / 書込みがすべてのインスタンス間で起きます。この問題を回避するには、空きリストをグループ化して、各インスタンスに 1 グループずつ割り当てます。これで、各インスタンスが空きリストを含む専用のブロックを持つことができます。各インスタンスが専用の空きリストを使用するので、空きリストを含む同一ブロックに複数のインスタンスがアクセスしてインスタンス間で競合が起きることはありません。

関連項目： 第 17 章「空きリスト・グループによるデータ分割」（OPS 環境で最適なパフォーマンスを得るための空きリストの適切な使用方法について）、21-11 ページの「データベースのバックアップ」

ディスクの親和性

ディスクの親和性によって、パラレル・データ操作言語（PDML）または問合せ操作を実行するインスタンスが決定します。OPS 構成では、ディスクの親和性は PDML にとって特に重要な意味をもちます。複数の文で一貫性のある親和性を保つことで、バッファ・キャッシュのヒット率が向上し、強制読込み / 書込みが減少します。

ほとんどの PDML では、パーティションによって並列性を細分化します。一方、パラレル問合せは、行 ID によって細分化されます。PDML に親和性を実現するには、パーティションのインスタンスへのマッピングが必要です。MPP システムでのパーティションの親和性を決めるのに、パーティションのセグメント・ヘッダーを使用します。MPP の各ノードがローカル・デバイスへアクセスする様にすれば、パフォーマンスが改善されます。これにより、すべてのノードのバッファ・キャッシュ・ヒット率が向上します。

その他の OPS 構成では、パーティションのインスタンスへの固定的なマッピングが行なわれます。すべての OPS/MPP 構成では、パーティションとインスタンスの親和性を考慮してプロセスおよび作業が割り当てられます。

関連項目：『Oracle8*i* 概要』（パラレル・データ操作言語（PDML）および並列度について）『Oracle8*i* チューニング』（PDML のチューニングおよびオプティマイザ・ヒントについて）ポート固有のインストール・ガイドおよび構成ガイド（各ポートのディスクの親和性について）

ジョブおよびインスタンスの親和性

この機能を使用すると、どのインスタンスがどのジョブを処理するかを制御できます。パッケージ DBMS_JOB を使用すれば、それぞれのジョブの機能を考慮して最も有意義な方法でクラスタ全体にジョブを分配できます。選択されたインスタンスの SNP プロセスだけがジョブを実行できるので、負荷が均衡化され、ブロックの競合を抑制します。

たとえば、OPS とレプリケーションを同時に使用すると、遅延トランザクション・キューで ping が発生することがよくあります。それは、クラスタ環境のすべてのインスタンスが遅延トランザクション・キューからトランザクションを伝播させるからです。表に対するアクティビティを 1 つのインスタンスだけに制限するには、DBMS_JOB を使用してキュー内のジョブの処理を特定の OPS インスタンスに割り当てます。

次の例では、レプリケーションを使用してジョブの親和性を説明していますが、この機能はレプリケーション以外のシナリオでも使用できます。

DBMS_JOB パッケージの使用方法

この例では、DBMS_JOB の定数によってジョブおよびインスタンス間での " マッピングを行わない " ことが指定されています。つまり、どのインスタンスもジョブを実行できます。

DBMS_JOB.SUBMIT ジョブ・キューへジョブを送るには、次の構文を使用します。

```
DBMS_JOB.SUBMIT( JOB OUT BINARY_INTEGER,
WHAT IN VARCHAR2, NEXT_DATE IN DATE DEFAULTSYSDATE,
INTERVAL IN VARCHAR2 DEFAULT 'NULL',
NO_PARSE IN BOOLEAN DEFAULT FALSE,
INSTANCE IN BINARY_INTEGER DEFAULT ANY_INSTANCE,
FORCE IN BOOLEAN DEFAULT FALSE)
```

パラメータ INSTANCE および FORCE を使用してジョブおよびインスタンスの親和性を制御します。INSTANCE のデフォルト値は 0（ゼロ）であり、どのインスタンスもジョブを実行できます。特定のインスタンスでジョブを実行するには、INSTANCE 値を指定します。INSTANCE の値が負数または NULL であると、Oracle でエラー ORA-23319 が表示されます。

FORCE パラメータのデフォルトは FALSE です。FORCE を TRUE に設定すると、任意の正の整数をジョブ・インスタンスとして受け入れます。FORCE が FALSE のとき、指定したインスタンスが稼動中でないと、Oracle でエラー番号 ORA-23428 が表示されます。

DBMS_JOB.INSTANCE ジョブの実行を特定のインスタンスに割り当てるには、次の構文を使用します。

```
DBMS_JOB.INSTANCE(  JOB IN BINARY_INTEGER,
INSTANCE IN BINARY_INTEGER,
FORCE IN BOOLEAN DEFAULT FALSE)
```

この例で、FORCE パラメータのデフォルトは FALSE です。INSTANCE の値を 0（ゼロ）に設定すると、ジョブの親和性が変更され、使用可能なすべてのインスタンスが FORCE の値にかかわらずジョブを実行できます。INSTANCE 値が正の数で FORCE パラメータが FALSE の場合は、指定したインスタンスが稼動中であればジョブの親和性が変更され、稼動中でなければ Oracle でエラー ORA-23428 が表示されます。

FORCE パラメータを TRUE に設定すると、任意の正の整数がジョブ・インスタンスとして受け入れられてジョブの親和性が変更されます。INSTANCE の値が負数または NULL であると、Oracle でエラー ORA-23319 が表示されます。

DBMS_JOB.CHANGE ジョブに対応付けられたユーザー定義パラメータを変更するには、次の構文を使用します。

```
DBMS_JOB.CHANGE(  JOB IN BINARY_INTEGER,
WHAT IN VARCHAR2 DEFAULT NULL,
NEXT_DATE IN DATE DEFAULT NULL,
INTERVAL IN VARCHAR2 DEFAULT NULL,
INSTANCE IN BINARY_INTEGER DEFAULT NULL,
 FORCE IN BOOLEAN DEFAULT FALSE )
```

この例では、INSTANCE および FORCE という 2 つのパラメータが出現します。INSTANCE のデフォルト値は NULL であり、ジョブの親和性を変更できないことを示します。

FORCE のデフォルト値は FALSE です。指定したインスタンスが稼動中でないと、Oracle でエラー ORA-23428 が表示され、また、INSTANCE 値が負数であると、エラー ORA-23319 が表示されます。

DBMS_JOB.RUN DBMS_JOB.RUN の FORCE パラメータのデフォルトは FALSE です。FORCE を TRUE に設定すると、インスタンスの親和性とは無関係に、どのインスタンスでもジョブをフォアグラウンド・プロセスで実行できます。FORCE が FALSE のときは、指定したインスタンスだけがジョブをフォアグラウンドで実行できます。FORCE が FALSE であるときに、接続先インスタンスが正しくない場合は、Oracle でエラー ORA-23428 が表示されます。

```
DBMS_JOB.RUN( JOB IN BINARY_INTEGER,
FORCE IN BOOLEAN DEFAULT FALSE)
```

関連項目： DBMS_JOB の詳細は、『Oracle8*i* 管理者ガイド』および『Oracle8*i* パッケージ・プロシージャ リファレンス』を参照してください。

透過的アプリケーション・フェイルオーバー

アプリケーション・フェイルオーバーとは、データベースとの接続が途切れた場合に、アプリケーションが自動的に再接続できる能力のことです。アクティブ・トランザクションはロールバックしますが、新しいデータベース接続と元のデータベース接続はまったく同じです。このことは、接続が終了した原因にかかわらず、どの場合にも当てはまります。

透過的アプリケーション・フェイルオーバーによって、アプリケーションにサービスを提供するインスタンスが残っている限り、クライアントは接続が失われたことに気が付きません。DBA は、どのアプリケーションをどのインスタンスで実行するかを制御し、また、アプリケーションごとにフェイルオーバー順序を作成します。

関連項目： 18-29 ページの「接続ロード・バランシング」、22-2 ページの「インスタンス障害の回復」、および『Oracle8*i* チューニング』

キャッシュの一貫性

キャッシュの一貫性は、オブジェクトのコピーを複数作成した場合に、それらのコピーの一貫性を維持するための手法です。この項では、次の項目について説明します。

- パラレル・キャッシュ管理について
- 非 PCM キャッシュ管理について

パラレル・キャッシュ管理について

OPS では、パラレル・キャッシュ管理（PCM）と呼ばれるテクノロジを使用して、個別の Oracle インスタンスが 1 つ以上のノードで同時に実行されます

PCM では、IDLM ロック（IDLM）を使用して、OPS のインスタンスによるリソースへのアクセスを調整します。データベース・リソースの例としては、ロールバック・セグメント、ディクショナリ・エントリ、データ・ブロックなどがあります。最も頻繁に要求されるデータベース・リソースは、データ・ブロックです。

別々のノード上にあるインスタンスのバッファ・キャッシュについて、そのキャッシュの一貫性を維持するのにパラレル・キャッシュ・マネージャを使用します。PCM バッファ・キャッシュ・ロックに関連付けられている一連のグローバル定数（GC_*）初期化パラメータは、ディクショナリ・キャッシュやライブラリ・キャッシュなどで使用するロックと同じではありません。

PCM では、任意の SGA にマスター・コピー・データ・ブロックを入れておくと、他の SGA でマスターのコピーを要求した場合に、両方の SGA のコピーがまったく同じであることを保証します。したがって、インスタンスでトランザクションがコミットされたかどうかに関係なく、すべての SGA では、インスタンスによって加えられたあらゆる変更を反映した最新のブロック・コピーが維持されます。

1 つのバッファ・キャッシュ内でデータ・ブロックを変更すると、他のバッファ・キャッシュ内のすべての既存コピーは現行のものではなくなります。変更操作が完了すると、新しいコピーを取得できます。

PCM では、I/O 操作および IDLM の使用を最小限に抑えてキャッシュの一貫性を確保します。キャッシュの一貫性を確保するための I/O 操作およびロック・オペレーションが実行されるのは、あるインスタンスのバッファ・キャッシュにデータ・ブロックの現行バージョンが入っていて、別のインスタンスが更新のためにそのブロックを要求している場合だけです。

単一 OPS インスタンスで複数のトランザクションを実行するときは、一連のデータ・ブロックへの読込みアクセスを複数のトランザクションで共有できるので、追加のインスタンス・ロック・オペレーションは必要ありません。したがって、競合や競争は発生しません。この状態は、他のインスタンスで実行しているトランザクションがそのブロックへの書込みアクセスを要求するまで継続します。

共有モードでは、IDLM によってインスタンス・ロックの状態を管理します。排他モードでは、すべてのロックはローカルであり、IDLM でデータベース・リソースを調整する必要はありません。

インスタンスがインスタンス・ロックを使用するのは、リソースのマスター・コピーの所有権を示すためです。インスタンスがデータベース・リソースのマスターまたは " 所有者 " になると、そのインスタンスはデータベース・リソースのインスタンス・ロックの所有者になり、固定ロックを保持します。しかし、解放可能なロックは当然解放されます。

リソースのマスター・コピーは、リソースのコピーであり、更新可能であることを示します。他のインスタンスが更新のためにリソースを要求するまで、インスタンスはインスタンス・ロックを手放しません。その後、他のインスタンスがリソースのマスター・コピーを所有すると、そのインスタンスがインスタンス・ロックの所有者になります。

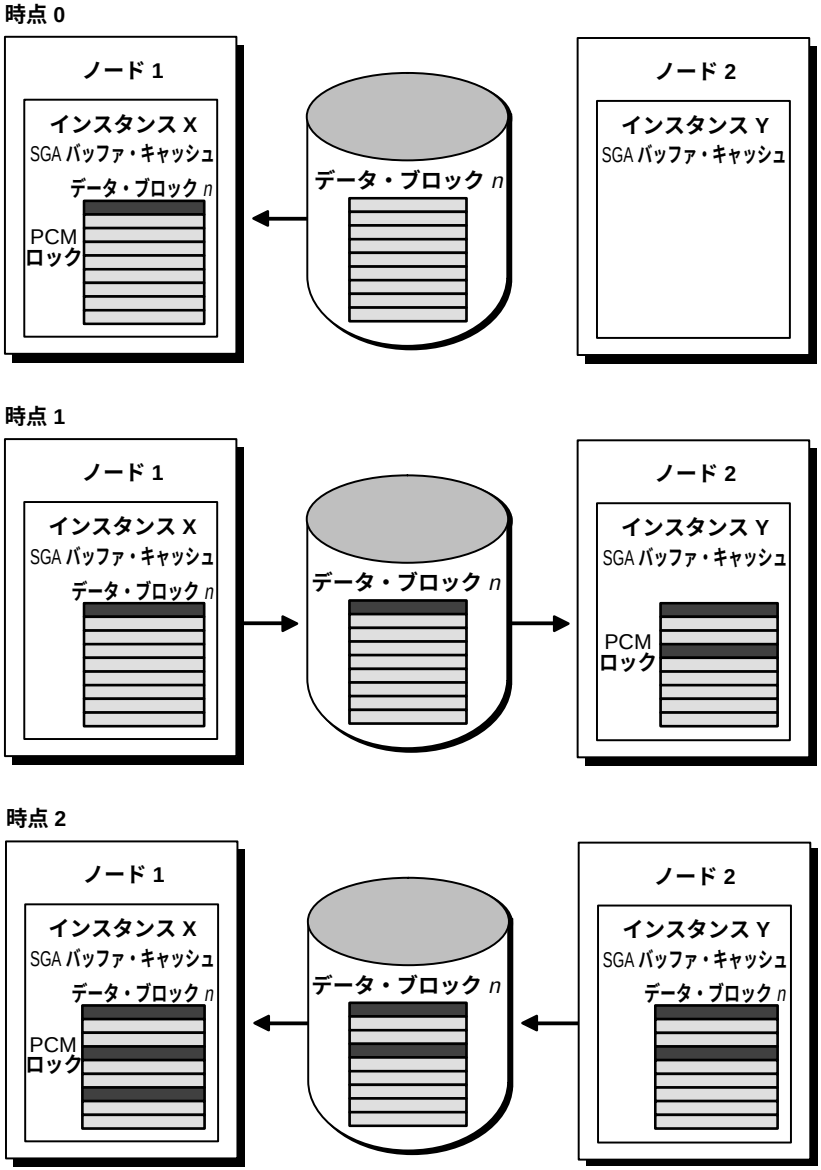
注意： トランザクションとパラレル・キャッシュ管理は、Oracle 内の自立機構です。PCM ロックは、すべての形式のトランザクション・ロックから独立して機能します。

例 次の例と図 4-2 について考えてみます。この例では、1 つの PCM ロックが 1 つのブロックを扱っていますが、実際には複数のブロックを扱うことができます。

- インスタンス X は、行 1 を含むデータ・ブロック *n* に対する PCM ロックの所有者になり、その行を更新する。
- インスタンス Y は、行 4 を更新するためにそのブロックを要求する。
- インスタンス X は、そのデータ・ブロックをディスクに書き込み、PCM ロックを解放する。
- インスタンス Y は、ブロックおよび PCM ロックの所有者になり、行 4 を更新する。
- インスタンス X は、行 7 を更新するためにブロックを要求する。
- インスタンス Y は、そのデータ・ブロックをディスクに書き込み、ブロックおよび PCM ロックを解放する。
- インスタンス X は、ブロックおよび PCM ロックの所有者になり、行 7 を更新する。
- 他のインスタンスがこのブロックを要求するまで、インスタンス X はトランザクションをコミットした後も引き続き PCM ロックおよびブロックのマスター・コピーを所有する。

関連項目： 9-10 ページの「バッファ状態およびロック・モードの変化」

図 4-2 複数のインスタンスによる同一データ・ブロックの更新



PCM ロックおよび行ロックの独立性

PCM ロックおよび行ロックの操作は、それぞれ独立しています。PCM ロックが関与するブロック・セット中に行ロックが存在しても、行ロックにいっさい影響を与えずにインスタンスから PCM ロックを解放できます。行ロックはトランザクションの実行中に取得されます。インスタンスがデータ・ブロックなどのデータベース・リソースを更新のために読み込むとき、データベース・リソースに対して PCM ロックを取得します。したがって、他のインスタンスがブロックを要求した場合は、1 つのトランザクションの実行中に同じ PCM ロックの解放と取得が何度も行われることがあります。

これに対して、行ロックの場合は、行に対する変更がコミットまたはロールバックされない限りトランザクションは行ロックを解放しません。Oracle では、同時性を制御する内部機構によって個々のトランザクションを切離して処理するので、一方のトランザクションがデータの変更をコミットするまで、他方のトランザクションはデータの変更を認識できません。行ロックの同時性制御機構は、パラレル・キャッシュ管理に依存しない独立した機構です。つまり、同時性を制御するために、PCM ロックは必要ありません。また、PCM ロック・オペレーションは、個々のトランザクションのコミットまたはロールバックとは無関係に行われます。

インスタンス・ロック・モード

インスタンスは、必要なアクセスの種類に応じて、共有モードまたは排他モードのどちらかで、一連のデータ・ブロックに対するインスタンス・ロックを取得できます。

- 排他ロック・モードでは、インスタンスは一連のブロックを更新できる。

  インスタンスがデータ・ブロックを更新する必要があるときに、そのブロックに対して他のインスタンスがインスタンス・ロックをすでに保持している場合は、IDLM ロックを使用して他のインスタンスによって保持されたインスタンス・ロックを解放するように要求します。必要であれば、他のインスタンスはブロックをディスクに書き込みます。

- 読込みロック・モードでは、インスタンスはブロックの読込みだけができる。

  インスタンス・ロックの対象となっているブロックを読み込むだけで変更する必要がない場合は、共有モードで複数のインスタンスが 1 つのインスタンス・ロックを共有できます。これによって、すべてのインスタンスでは、それぞれのメモリーに常駐しているブロックのコピーが現行コピーであることが保証され、また、インスタンス・ロック・オペレーションによって他のインスタンスにブロックを要求しなくても、ディスクからの現行コピーの読込みが保証されます。これは、読込み専用としてアクセスされるデータベース部分に関しては、インスタンス・ロックを解放する必要がないことを意味します。多数のアプリケーションでは、読込み専用アクセスが処理時間の大部分を占めています。

- NULL ロック・モードでは、許可を得る必要なしにインスタンスがブロックのロックを保持できる。

  このモードを使用すると、ロックの取得と解放を繰り返す必要がありません。ロックのモードを別のモードに切り替えるだけで済みます。

関連項目： 第 15 章「PCM インスタンス・ロックの割当て」（データ・ファイルの PCM ロックの割当てについて）

非 PCM キャッシュ管理について

OPS では、Oracle のすべてのキャッシュが複数のインスタンス間で確実に同期化されます。よく知られた例として、1 つのノードでブロックを変更したときにその変更を他のノードにも適用するための同期処理があります。ただし、同期にはもっと広い意味があります。

OPS によるインスタンス間でのキャッシュの同期方法を理解すると、システム・パフォーマンスに影響を与えるオーバーヘッドのことも理解できます。5 つのノードで構成されたパラレル・サーバーがあり、ノードの 1 つでユーザーが表を削除したとします。削除した表の定義のコピーが、5 つのディクショナリ・キャッシュのすべてに含まれています。したがって、ノードの 1 つでキャッシュから表を削除した場合は、他の 4 つのディクショナリ・キャッシュから同じ表のコピーを削除する必要があります。OPS では、IDLM を使ってこの処理が自動的に行われます。他のノードのユーザーは、ロック状況の変更を通知されます。

ライブラリおよび表情報を各ノードに保存することには、大きな利点があります。DROP TABLE コマンドを使って他のキャッシュを強制的にフラッシュすることもありますが、これによるパフォーマンスへの影響はわずかで、複数のキャッシュを使用できることで得られる利点とは比べものになりません。

関連項目： 4-5 ページの「領域管理」および 4-5 ページの「システム変更番号」（非 PCM キャッシュ管理に関するその他の例）

5

Oracle Parallel Server 用の Oracle インスタンス・アーキテクチャ

[Architecture] is music in space, as if it were a frozen music...

— Schelling, *Philosophie der Kunst*

この章では、Oracle Parallel Server（OPS）アーキテクチャの機能について説明し、標準の（排他モードの）Oracle サーバーとの相違を示します。

- 概要
- OPS の複数インスタンス・アーキテクチャの特性
- システム・グローバル領域
- バックグラウンド・プロセス
- Oracle Parallel Server の構成ガイドライン

概要

OPS アーキテクチャでは、各 Oracle インスタンスが次のものをそれぞれ専用に持っています。

- システム・グローバル領域（SGA）
- バックグラウンド・プロセス
- ORACLE_SID（すべてのインスタンスで同じ値を使用することもできます）
- REDO ログの集合

OPS 環境では、すべてのインスタンスが次のファイル集合を共有して、アクセスします。

- データ・ファイル

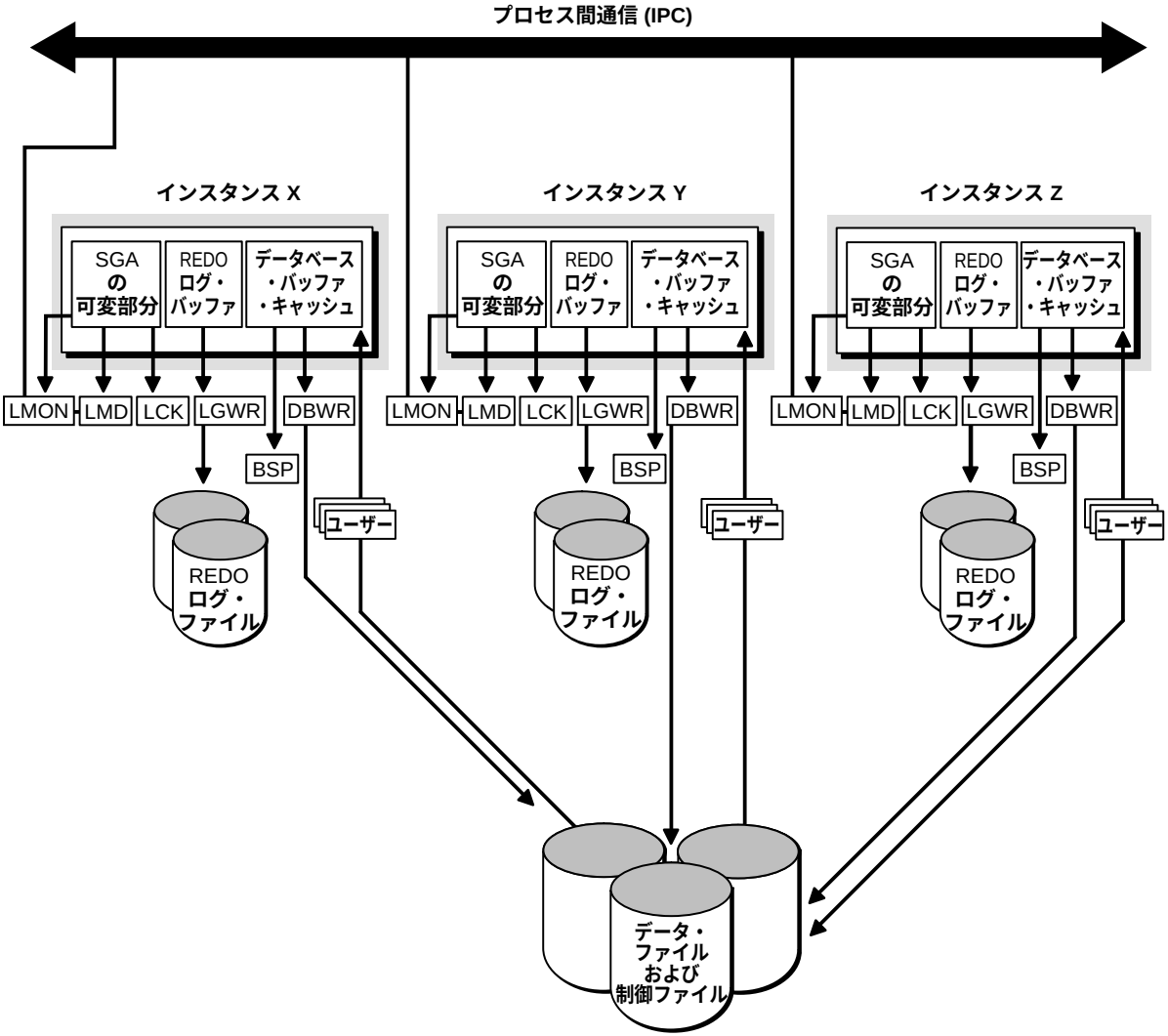
- 制御ファイル
- REDO ログ

OPS インスタンスには、次のものが含まれています。

- SGA に追加された PCM ロック領域。これは、共有リソースまたは " ロック要素 " が使用するためのものです。
- 統合分散ロック・マネージャ（IDLM）コンポーネント。これは、グローバル・ロックおよびリソースの領域です。
- 追加のバックグラウンド・プロセス LCK*n*。これは、パラレル・サーバーの複数のインスタンス間で共有リソースのロックを調整するためのものです。
- 追加のバックグラウンド・プロセス LMON および LMD0。これは、グローバル・ロックおよびリソースを管理するものです。

図 5-1 に、OPS の基本要素を示します。DBWR プロセスがデータを書き込み、ユーザーがデータを読み取っています。フォアグラウンド（FG）プロセスおよびバックグラウンド・プロセス LMD、LCK は、インターコネクトを使用して他のインスタンスと直接通信します。

図 5-1 Oracle Parallel Server の基本要素



関連項目：『Oracle8*i* 概要』の「メモリー構造とプロセス」

OPS の複数インスタンス・アーキテクチャの特性

次に、OPS の特性のまとめを示します。

- Oracle インスタンスはネットワーク内の 1 つ以上のノードで起動できます。
- 各インスタンスは、独立した 1 つのシステム・グローバル領域（SGA）およびバックグラウンド・プロセスの集合を持っています。
- すべてのインスタンスが同じデータ・ファイルおよび制御ファイルを共有します。
- 各インスタンスは、それぞれ専用の REDO ログ・ファイルの集合を持っています。

> **注意：** REDO ログは、万一のインスタンス障害に備えて、すべてのインスタンスからアクセスできることが必要です。一部の MPP プラットフォームのように REDO サーバーが設置されている場合は、OPS システム全体に対して REDO ログの集合が 1 つだけあれば十分です。

- アーカイブ済みログはプライベートなものですが、メディア回復のためにすべてのインスタンスからアクセスできることが必要です。
- すべてのインスタンスでは、同じデータベースに同時にトランザクションを実行でき、また、各インスタンスでは複数のユーザーがトランザクションを実行できます。
- 行レベル・ロックが保持されます。

パラレル・サーバーは、非パラレル・サーバーと同じ方法で管理できます。たとえば、ユーザーまたはオブジェクトの作成は、どの単一インスタンスでも実行できます。ただし、特定のインスタンスに接続しなければ実行できない管理タスクもあります。

クライアント / サーバー・アーキテクチャを使用すると、データベースにアクセスするアプリケーションは、パラレル・サーバーのインスタンスと同じノードで実行することも、別のノードで実行することもできます。パラレル・サーバーは、分散データベース・システムの一部として組み込むことができます。標準 Oracle Server がデータ・ファイルを所有しているか（排他モード）、パラレル・サーバーが所有しているか（排他モードまたは共有モード）に関係なく、分散トランザクションでは同じ方法を使ってリモート・データベース内のデータにアクセスします。

各ノードでその他の非 Oracle プロセスを実行することも、システムの全体または一部を Oracle 専用とすることもできます。たとえば、5 つのノードからなる構成で、パラレル・サーバーとそのアプリケーションに 3 つのノードを割り当て、残りの 2 つのノードを非 Oracle アプリケーション用に使用することができます。

システム・グローバル領域

パラレル・サーバーの各インスタンスは、それぞれ専用のシステム・グローバル領域（SGA）を持っています。SGA のメモリー構造体を次に示します。

- データ・ブロック用のバッファ・キャッシュ
- データ・ディクショナリ情報用のディクショナリ・キャッシュ
- REDO エントリ用の REDO ログ・バッファ
- 共有 SQL 領域および共有 PL/SQL 領域を含む共有プール
- インスタンス・ロック領域（パラレル・サーバー内のみ）

OPS では、複数の SGA でのデータの共有は、パラレル・キャッシュ管理（PCM）ロックを使用してパラレル・キャッシュ管理によって制御します。

同じデータ・ブロックの複数のコピーが、複数の SGA の中に同時に存在する場合があります。PCM ロックは、すべてのインスタンスについてデータベース・バッファ・キャッシュの一貫性を維持します。したがって、あるインスタンスが実行した変更を、別のインスタンスがそのまま読み取ることができます。

各インスタンスは共有プールを 1 つずつ持っていますが、これは、該当のインスタンスに接続しているユーザー・アプリケーションだけが使用できます。同じインスタンスを使用する異なるアプリケーションが同じ SQL 文を送ると、その文は解析され、そのインスタンスの SGA に 1 回だけ格納されます。他のインスタンスでアプリケーションが同じ SQL 文を送った場合は、そのインスタンスもその文を解析して格納します。

関連項目： 第 9 章「パラレル・キャッシュ管理インスタンス・ロック」

バックグラウンド・プロセス

OPS の各インスタンスは、それぞれ専用のバックグラウンド・プロセスの集合を持っています。これらのプロセスは、排他モードでの単一サーバーのバックグラウンド・プロセスと同じものです。DBWR、LGWR、PMON および SMON プロセスは、すべてのインスタンスに存在します。オプション・プロセスの ARCH、CKPT、D*nnn* および RECO は、初期化パラメータを設定することで使用できます。OPS の各インスタンスは、標準バックグラウンド・プロセスだけでなく、少なくとも 1 つのロック・プロセス、LCK0 を持っています。必要であれば、追加のロック・プロセスを使用できます。

OPS では、IDLM が LMON および LMD0 プロセスも使用します。LMON は、IDLM のインスタンス、障害、および関連回復処理を管理します。特に、LMON はグローバル・ロックに関連した回復処理の部分を担当します。LMD プロセスは、リモート・ロック要求、たとえば他のインスタンスから出された要求などを処理します。LMD は、デッドロックの検出も行います。LCK プロセスは、インスタンスが使用するロックを管理し、それらのロックに対する他のインスタンスからの要求を調整します。

共有モードで、あるインスタンスがダウンした場合は、他のインスタンスの SMON が障害を検出し、失敗したインスタンスに代わって回復します。回復処理を請け負ったインスタンスの LCK プロセスは、ダウンしたインスタンス用の未解決の PCM ロックがあれば、それをクリーン・アップします。

関連項目： 7-6 ページの「LCK プロセス」および 9-12 ページの「GC_* 初期化パラメータ」

フォアグラウンドでのロックの取得

フォアグラウンド・プロセスは、ロック要求を直接にリモート LMD プロセスに送ります。フォアグラウンド・プロセスが送信する要求には、ロック要求の対象リソース名および必要なロック・モードなどの情報が含まれます。

IDLM は要求を非同期式に処理するので、フォアグラウンド・プロセスは要求が完了してから要求をクローズします。

関連項目： これらの要求の処理の詳細は、8-2 ページの「非同期トラップ（AST）によるロック要求状態の通知」を参照してください。

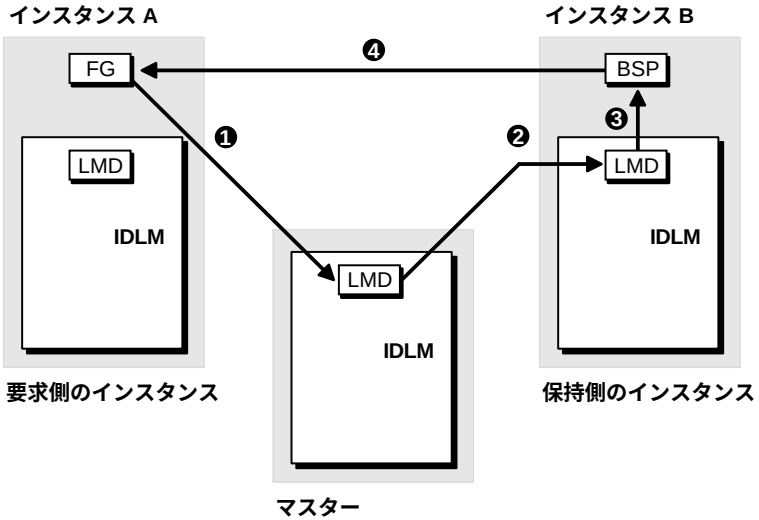
キャッシュ・フュージョン処理およびブロック・サーバー・プロセス

キャッシュ・フュージョンは、インスタンスが排他モードで保持しているブロックをもう 1 つのインスタンスが要求したときに発生する、キャッシュの一貫性の矛盾を解決します。この場合、Oracle は、ロックを保持しているインスタンスのメモリー・キャッシュから要求元インスタンスへ読込み一貫性に基づいたバージョンのブロックを直接送信します。このとき、ディスクへのブロックの書込みは行いません。

キャッシュ・フュージョンは、ブロック・サーバー・プロセス（BSP）を使用してコミットされていないトランザクションをロールバックします。BSP は、読込み一貫性に基づいたブロックを要求元へ直接送信します。ブロックの状態は、要求元インスタンスが要求を出した時点のものと一致しています。図 5-2 に、このプロセスを示します。

この処理は、一貫性読込み、読込みアクセスまたは書込みアクセスを要求した場合にのみ実行されます。この処理によって、ロック・ダウングレードの回数およびインスタンス間の通信量が大幅に減少します。また、OPS に不適切だとみなされていたアプリケーション、たとえば、OLTP およびハイブリッド・アプリケーションなど、数種のアプリケーションの拡張性を改善します。

図 5-2 Consistent Read サーバー処理



1. 要求元の FG（フォアグラウンド）プロセスがマスター・ノードへロック要求メッセージを送信します。マスターとして機能できるノードは、要求元ノード、ロックを保持しているノード、またはまったく異なるノードでもかまいません。
2. マスター・ノードの LMD プロセスは、ロックを保持しているノード、つまり、要求されたブロックに対して排他ロックを保持しているノードの LMD プロセスへロック要求を送信します。
3. ロックを保持しているノードの LMD プロセスは、着信メッセージを処理し、ロックを保持しているインスタンスの BSP に対して、要求されたブロックの一貫性読込みのコピーを作成するように要求します。
4. BSP は要求されたブロックの読込みコピーを作成して要求元の FG プロセスに送信します。

Oracle Parallel Server の構成ガイドライン

OPS を設定するときは、表 5-1 に示すガイドラインに従ってください。

表 5-1 パラレル・サーバー構成のガイドライン

構成事項	ガイドライン
バージョン	すべてのノードの Oracle のバージョンが同じであることを確認してください。
リンク	実行ファイルへの UNIX のソフト・リンクまたはハード・リンク（"alias"）の使用は、OPS ではお薦めできません。単一ノードに実行ファイルを保管すると、そのノードに障害が発生した場合にどのノードも作動不能になります。
初期化パラメータ	初期化パラメータは、1 つのファイルにまとめて入れてください。このパラメータは、OPS のすべてのインスタンスで同じにする必要があります。IFILE オプションを使用して、すべてのインスタンスの個別初期化ファイルに、このファイルを組み込みます。 ROLLBACK SEGMENTS、THREAD INSTANCE などのインスタンス固有のパラメータは、IFILE と一緒にローカル・インスタンス・パラメータ・ファイルに入れておきます。IFILE はさらに大きい共通ファイルを参照し、そのファイルには、同じ値をそのまま維持する必要がある他のすべてのパラメータが入っています。
制御ファイル	すべてのインスタンスからアクセスできることが必要です。
データ・ファイル	すべてのインスタンスからアクセスできることが必要です。
ログ・ファイル	制御ファイルおよびデータ・ファイルと同じディスク集合の中に入っていることが必要です。各インスタンスはそれぞれ固有の REDO ログ・ファイルを持っていますが、回復処理を可能にするには、すべての REDO ログ・ファイルがどのインスタンスからもアクセスできるように設定する必要があります。
NFS	NFS を使用すると、Oracle 実行ファイルへアクセスできますが、データベース・ファイルまたはログ・ファイルへのアクセスはできません。NFS を使用する場合は、元ファイルを持つノードで発生した障害が他のあらゆる機能に影響を与えます。
アーカイブ済み REDO ログ・ファイル	すべてのインスタンスからアクセスできることが必要です。

6

パラレル・サーバー用の Oracle データベース・アーキテクチャ

この章では、OPS の複数インスタンスに関連する Oracle データベース・アーキテクチャについて説明します。

- ファイル構造
- データ・ディクショナリ
- 順序ジェネレータ
- ロールバック・セグメント

ファイル構造

次の各項では、OPS に適用される制御ファイル、データ・ファイルおよび REDO ログ・ファイルの機能について説明します。

- 制御ファイル
- データ・ファイル
- REDO ログ・ファイル

制御ファイル

すべての OPS インスタンスは、同じ制御ファイルにアクセスします。制御ファイルには、GC_FILES_TO_LOCKS などのグローバル定数初期化パラメータの値が入っています。いくかのパラメータは、同時実行するすべてのインスタンスで同じ値を指定する必要があります。Oracle では、各インスタンスの起動時に、共通パラメータ・ファイル（または各インスタンスのパラメータ・ファイル）と制御ファイルに入っているグローバル定数初期化値を比較し、値が異なる場合はメッセージを生成します。

注意： 各クラスタに設置される制御ファイルは 1 つだけです。しかし、複数のインスタンスで制御ファイルをミラー化し、各インスタンスが同じ制御ファイルのコピーにアクセスする場合があります。

関連項目： パラメータの詳細は、18-10 ページの「すべてのインスタンスで同じ値にする必要があるパラメータ」および付録 A の「初期化パラメータ」を参照してください。また、『Oracle8*i* 概要』も参照してください。

データ・ファイル

すべての OPS インスタンスは、同じデータ・ファイルにアクセスします。Oracle では、パラレル・モードでも、排他モードでも、データベース・ファイルは同じです。Oracle を排他モードまたはパラレル・モードのどちらで起動しても、データ・ファイルを変更する必要はありません。

各インスタンスがそれぞれ専用のデータ・ブロックの集合を使用できるようにデータの物理的配置を管理すると、パフォーマンスが向上します。たとえば、空きリストを使用して、挿入用に使用する領域を特定のインスタンスに割り当てることができます。

インスタンスは、起動のたびにすべてのオンライン・データ・ファイルへのアクセスを検証します。最初に起動した OPS インスタンスは、すべてのオンライン・ファイルへのアクセスを検証してメディア回復が必要かどうかを調べなければなりません。その他のインスタンスでは、すべてのオンライン・データ・ファイルにアクセスしなくても処理を実行できますが、検証済みでないファイルを使用する操作はすべて失敗し、メッセージが返されます。

1 つのインスタンスでデータ・ファイルを追加する、またはオフラインのデータ・ファイルをオンラインにすると、すべてのインスタンスがそのファイルへのアクセスを検証します。あるインスタンスが他のインスタンスからアクセスできないディスクに新しいデータ・ファイルを追加すると、検証は失敗しますが、どのインスタンスでも処理はそのまま続けられます。複数のインスタンスが同じデータ・ファイルの異なるコピーにアクセスした場合も、検証は失敗します。

いずれかのインスタンスで検証が失敗した場合は、問題を診断および修正し、その後で ALTER SYSTEM CHECK DATAFILES 文を使用してアクセスを検証します。この文で GLOBAL オプション（デフォルト）を使用すると、すべてのインスタンスでオンライン・データ・ファイルへのアクセスが検証されます。また、LOCAL オプションを指定すると、現行インスタンスでアクセス検証が実行されます。

ALTER SYSTEM CHECK DATAFILES を使用すると、アクセスを検証するインスタンスでオンライン・データ・ファイルが使用可能になります。

回復を実行するインスタンスが必要なすべてのオンライン・データ・ファイルへのアクセスを検証できない限り、Oracle はインスタンス障害またはメディア障害から回復できません。

Oracle は、自動的に絶対ファイル番号を相対ファイル番号にマッピングします。OPS を使用しても、この値に影響はありません。使用しているデータ・ファイルについて、両方の番号を見るときは、V$DATAFILE ビューへの問合せを行います。

関連項目： 詳細は、第 17 章「空きリスト・グループによるデータ分割」、22-4 ページの「インスタンス回復のためのデータ・ファイルへのアクセス」および 15-6 ページの「GC_FILES_TO_LOCKS の設定 : 各データ・ファイルの PCM ロック数」を参照してください。相対ファイル番号の詳細は、『Oracle8*i* 概要』を参照してください。

REDO ログ・ファイル

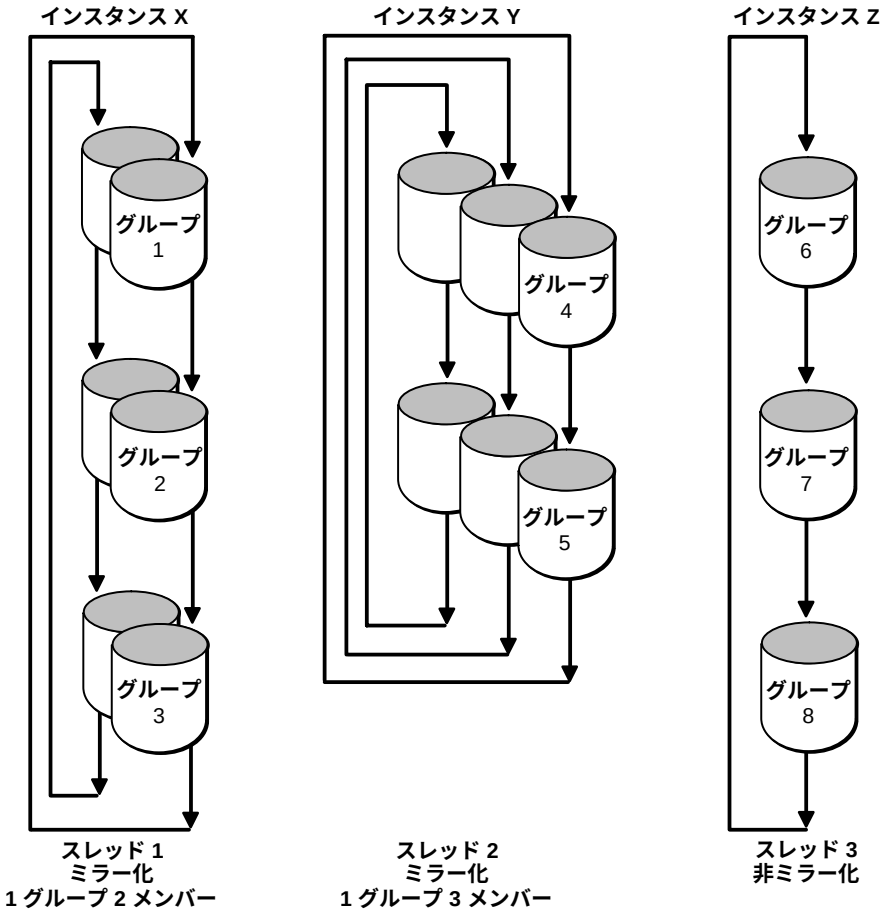
OPS では、各インスタンスが専用のオンライン REDO ログ・ファイル・セットに書き込みます。単一のインスタンスが書き込んだ REDO を、REDO のスレッドと言います。個々のオンライン REDO ログ・ファイルには、それぞれ特定のスレッド番号が関連付けられます。オンライン REDO ログをアーカイブすると、回復時にログを識別するためにそのスレッド番号が記録されます。

プライベート・スレッドは、ALTER DATABASE ADD LOGFILE コマンドに THREAD 句を指定して作成できる REDO ログです。パブリック・スレッドは、ALTER DATABASE ADD LOGFILE コマンドで THREAD 句を指定しなかった場合に作成される REDO ログです。

THREAD 初期化パラメータを指定した場合は、起動したインスタンスは、そのパラメータの値でプライベート・スレッドとして識別されているスレッドを取得します。THREAD がデフォルトの 0 である場合は、インスタンスはパブリック・スレッドを取得します。取得後は、取得したインスタンスが REDO スレッドを排他的に使用します。

オンライン REDO ログ・ファイルは、多重化または " ミラー化 " できます。多重 REDO ログは 2 つ以上のファイル・グループからなり、グループがアクティブであるときに、そのグループのすべてのメンバーへの書込みが同時に行われます。図 6-1 に、3 つの OPS インスタンスの REDO スレッドを示します。

図 6-1 REDO のスレッド



- インスタンス X はスレッド 1 を使用します。スレッド 1 には、グループ 1、2、3 という 3 つのグループのオンライン REDO ログ・ファイルがあります。スレッド 1 は多重化されており、各グループに REDO ログ・ファイルのコピー（メンバー）が 2 つずつあります。
- インスタンス Y はスレッド 2 を使用します。スレッド 2 には、グループ 4 および 5 という 2 つのグループのオンライン REDO ログ・ファイルがあります。スレッド 2 は多重化されており、各グループはそれぞれ 3 つのメンバーで構成されています。

- インスタンス Z はスレッド 3 を使用します。スレッド 3 には、グループ 6、7 および 8 という、多重化されていない 3 つのグループのオンライン REDO ログ・ファイルがあります。

グループ番号は、データベース内で一意であるため、スレッド内でも一意です。しかし、スレッドにグループを割り当てる順序およびインスタンスにスレッドを割り当てる順序は任意です。

たとえば、図 6-1 で、スレッド 1 はグループ 1、2 および 3 からなり、スレッド 2 はグループ 4 および 5 からなります。しかし、スレッド 1 にグループ 2、4 および 5 を割り当て、スレッド 2 にグループ 1 および 3 を割り当ててもかまいません。V$LOGFILE ビューは、各 REDO ログ・ファイルに対応するグループ番号を表示します。

各スレッドを異なるグループ数およびメンバー数で構成することは可能ですが、管理しやすいように、すべてのスレッドを一定の標準に従って構成することをお薦めします。

OPS では、ミラー化の程度またはグループあたりのメンバー数が異なる、さまざまなインスタンス構成が可能です。また、それぞれのインスタンスを異なるグループ数で構成することも可能です。たとえば、第 1 のインスタンスには 1 グループあたり 2 メンバーのグループが 3 つあり、第 2 のインスタンスには多重化されていない 4 つのログ・ファイルがあり、第 3 のインスタンスには 1 グループあたり 4 メンバーのグループが 2 つあるという構成が考えられます。このような構成は管理上は不便なのですが、システムの持つ機能をフルに引き出すためにはこの構成が必要になることもあります。

各インスタンスには、少なくとも 2 つのグループのオンライン REDO ログ・ファイルが必要です。現行グループが満杯になると、インスタンスは次のログ・ファイル・グループに移って書込みを開始します。ログの切替え時に制御ファイルに情報が書き込まれます。この制御ファイルの情報は、アーカイブ後に、満杯のグループおよびそのスレッド番号を識別するために使用できます。

REDO ログ・ファイルに関する情報は、制御ファイルに保持することができますが、REDO ログ・ファイルの数は CREATE DATABASE 文の MAXLOGHISTORY オプションの値によって制限されます。グループあたりに必要なメンバー数は 1 つだけです。OPS で MAXLOGHISTORY の値を設定するときは、単一インスタンスの Oracle で通常設定する値よりも大きい値を使用します。OPS では複数の REDO ログ・ファイルの履歴を追跡しなければならないからです。

注意： MAXLOGHISTORY は、非常に高い可用性が要求されるサイトで使用すると便利です。インスタンスおよびログ・ファイルの数がたいへん多いときに、このオプションを使うと回復を容易に管理できます。

関連項目：『Oracle8*i* 概要』の「回復構造体」（多重 REDO ログ・ファイルについて）、「REDO ログ・ファイルのアーカイブ」、21-2 ページの「チェックポイントおよびログ・スイッチ」および 22-6 ページの「メディア障害の回復」

データ・ディクショナリ

各 OPS インスタンスの SGA には、データ・ディクショナリ情報が入ったディクショナリ・キャッシュ（または行キャッシュ）があります。データ・ディクショナリの構造は、OPS の Oracle インスタンスでも、排他モードのインスタンスでも同じです。インスタンス・ロックは、複数インスタンスのデータ・ディクショナリ・アクティビティを調整します。

順序ジェネレータ

この項では、CREATE SEQUENCE 文およびそのオプションについて説明します。

- CREATE SEQUENCE 文
- CACHE オプション
- ORDER オプション

CREATE SEQUENCE 文

SQL の CREATE SEQUENCE 文でデータベース・オブジェクトを設定し、そのオブジェクトから、複数のユーザーがそれぞれ一意の整数を生成することができます。各ユーザーは、同じ順序番号ジェネレータにアクセスしたトランザクションを他のユーザーがコミットするまで待つ必要がありません。

OPS を使用すると、複数のインスタンスの複数のユーザーが一意の順序番号を生成するときに、インスタンス間での協調動作または競合を最小限に抑えることができます。OPS では、インスタンス・ロックによってインスタンス間の順序が調整されます。

CYCLE オプションを使用した場合を除き、順序番号は常に一意のものです。ただし、次の項で述べるように、CACHE オプションだけを使用し ORDER オプションを使用しなかった場合は、順序番号の割当ての順番が乱れることがあります。

関連項目： CREATE SEQUENCE および CYCLE オプションの詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

CACHE オプション

CREATE SEQUENCE 文の CACHE オプションを使用すると、順序番号が事前に割り当てられて、インスタンスの SGA 内に保管されるため、アクセスが速くなります。キャッシュする順序番号の数を CACHE オプションの引数として指定できます。デフォルト値は 20 です。

順序番号を保持するとパフォーマンスは大幅に向上しますが、順序列の一部の番号が失われることがあります。順序番号の一部が消えても、たとえば順序列を使用して主キー用の一意の番号を生成する場合など、アプリケーションによって重大でないこともあります。

ある順序からの番号を最初に要求した時点で、その順序用のキャッシュに指定個数の順序番号が入ります。キャッシュされた一連の番号のうちの最後の番号が割り当てられてしまうと、指定個数の順序番号が再びキャッシュに再移入されます。

各インスタンスは、それぞれ専用の順序番号キャッシュをメモリー内に保持しています。あるインスタンスがシャット・ダウンすると、保持された順序値のうち、コミット済みの DML 文で使用されていない値が失われることがあります。失われる値の個数は、最大で、CACHE オプションの値にシャット・ダウンしたインスタンスの数を掛けた値になります。インスタンスが正常にシャットダウンした場合でも、保持された順序番号が失われることがあります。

ORDER オプション

CREATE SEQUENCE 文の ORDER オプションを使用すると、要求した順序のとおりに順序番号が生成されます。ORDER オプションは、タイムスタンプおよびその他の順序など、複数のプロセスおよびインスタンスで要求の順序を示す必要があるときに使用します。

順序番号を順番通りに発行する必要がない場合は、CREATE SEQUENCE 文の NOORDER オプションを使用した方が、OPS 環境でのオーバーヘッドが大幅に減少します。

注意： OPS では、データベースがパラレル・モードでマウントされている場合、CREATE SEQUENCE 文に CACHE オプションと ORDER オプションを同時に指定することはできません。Oracle では、各インスタンスに順序値の一部が保持されている場合には正しい順番になるという保証はありません。このため、CACHE オプションおよび ORDER オプションの両方を指定して順序を作成した場合は、順番は確保されますがキャッシュされません。

ロールバック・セグメント

この項では、OPS との関係という観点から、ロールバック・セグメントについて説明します。

- OPS のロールバック・セグメント
- ロールバック・セグメントを制御するパラメータ
- パブリックおよびプライベート・ロールバック・セグメント
- インスタンスがロールバック・セグメントを取得する方法

OPS のロールバック・セグメント

ロールバック・セグメントには、読込み一貫性を確保するために Oracle で必要な情報、およびロールバックまたは異常終了したトランザクションで行った変更を取り消すための情報が入っています。各 OPS インスタンスは、SYSTEM ロールバック・セグメントを共有するとともに、少なくとも 1 つの専用ロールバック・セグメントを必要とします。

プライベートおよびパブリックのどちらのロールバック・セグメントも、インスタンスの起動時に取得でき、取得したインスタンスがそのセグメントを排他的に使用します。このセグメントは、オフラインになるか、またはロールバック・セグメント・パラメータの指定に従って取得したインスタンスがシャットダウンするまで排他的に使用されます。プライベート・ロールバック・セグメントは特定のインスタンスに一意のものであり、他のインスタンスがそれを使用することはできません。パブリック・ロールバック・セグメントは、追加のロールバック・セグメントを要求するインスタンスが起動してロールバック・セグメントを取得し、それをオンラインにするまで、オフラインのままであり、どのインスタンスにも使用されません。パブリック・ロールバック・セグメントがオンラインになると、それを取得したインスタンスが排他モードで使用します。

1 つのロールバック・セグメントに書込みを行うインスタンスは 1 つだけです（SYSTEM ロールバック・セグメントを除く）。ただし、他のインスタンスがそのセグメントから読み込んで、読込み一貫性のあるスナップショットを作成したり、インスタンス回復を実行したりできます。

OPS の必要最小限のロールバック・セグメント数は、同時に動作する最大インスタンス数に SYSTEM ロールバック・セグメント用の 1 を加えた数です。インスタンスを共有モードで起動するには、そのインスタンスが、少なくとも 1 つのロールバック・セグメント（パブリックでもプライベートでもかまいません）に対する排他アクセスを持っていることが必要です。

どの表領域の中にも、新しいロールバック・セグメントを作成できます。ロールバック・セグメントを別個の表領域として分割すると、ロールバック・データと表データの競合を軽減できます。また、これによって、表領域をオフラインにするのも簡単になります。なぜなら、表領域にアクティブ・ロールバック・セグメントが含まれていると、その表領域はオフラインにできないからです。

通常は、ロールバック・セグメントのすべてのエクステントを同じサイズにします。そのためには、記憶領域パラメータ INITIAL および NEXT に同じ値を指定します。

データ・ディクショナリ・ビュー DBA_ROLLBACK_SEGS は、各ロールバック・セグメントの名前およびセグメント ID 番号、所有者（PUBLIC またはその他）を示します。

関連項目： 14-4 ページの「追加のロールバック・セグメントの作成」（データベース作成時に必要なロールバック・セグメントについて）、『Oracle8*i* 管理者ガイド』（ロールバック・セグメントの競合、ロールバック・セグメントを追加した場合のパフォーマンスへの影響について）

ロールバック・セグメントを制御するパラメータ

次の初期化パラメータは、ロールバック・セグメントの使用を制御します。

ROLLBACK_SEGMENTS	インスタンスが起動時に取得するロールバック・セグメントの名前を指定。
GC_ROLLBACK_LOCKS	追加のインスタンス・ロックを予約して、ロールバック・エントリを含むブロックの競合を軽減。具体的には、遅延ロールバック・セグメント用のインスタンス・ロックを予約する。このセグメントには、オフラインにされた表領域内のトランザクション用のロールバック・エントリが入っている。

関連項目： 14-6 ページの「ロールバック・セグメントの監視」および『Oracle8*i* 概要』の「データ・ブロック、エクステント、セグメント」

パブリックおよびプライベート・ロールバック・セグメント

パブリックおよびプライベートのロールバック・セグメントには、パフォーマンスの違いはありません。しかし、インスタンスとロールバック・セグメントのマッチングでは、プライベート・ロールバック・セグメントの制御機能が優れています。この機能を使用すると、ロールバック・セグメントをインスタンスごとにそれぞれのディスクに配置できるので、パフォーマンスが向上します。したがって、高パフォーマンスを実現するシステムでは、ディスク競合を軽減するためにプライベート・ロールバック・セグメントを使用します。

パブリック・ロールバック・セグメントでは、ロールバック・セグメントのプールが形成され、どのインスタンスでも追加ロールバック・セグメントが必要になるとそのプールから取得できます。ただし、インスタンスのシャット・ダウンおよび起動が同時に行われる場合は、パブリック・ロールバック・セグメントを使用すると不利になることがあります。たとえば、インスタンス X がシャット・ダウンし、パブリック・ロールバック・セグメントを解放したとします。インスタンス Y が起動し、その解放されたロールバック・セグメントを取得します。再びインスタンス X が起動したとき、元のロールバック・セグメントを取得できません。TRANSACTIONS および TRANSACTIONS_PER_RBS の設定が適切でない場合は、インスタンスの起動時にパブリック・ロールバック・セグメントを取得することもできます。

パブリック・ロールバック・セグメントを使用することで、領域を有効に利用できます。たとえば、毎月さまざまなインスタンスで実行する長時間実行トランザクションの場合は、大きいパブリック・ロールバック・セグメントを 1 つだけ作成します。これにより、ロールバック・セグメントをオフラインにしてからまたオンラインに戻したり、他のインスタンスに " 移動 " したりできるので、負荷の高いインスタンスの処理が改善されます。

デフォルトのロールバック・セグメントはプライベートであり、パラメータ・ファイル内でそのセグメントを指定しているインスタンスがそれを使用します。プライベート・ロールバック・セグメントは、パラメータ ROLLBACK_SEGMENTS で指定します。

インスタンスが取得したパブリック・ロールバック・セグメントは、そのインスタンスが排他的に使用します。

作成したパブリック・ロールバック・セグメントおよびプライベート・ロールバック・セグメントは、どちらも ALTER ROLLBACK SEGMENT コマンドを使用してオンラインにすることができます。

注意： インスタンスを起動するには、1 つ以上のロールバック・セグメントが必要です。

インスタンスがロールバック・セグメントを取得する方法

起動時に、インスタンスは初期化パラメータ TRANSACTIONS および TRANSACTIONS_PER_ROLLBACK を使用して、取得するロールバック・セグメント数を判断します。必要なロールバック・セグメント数を求める公式は、次のとおりです。

$$\frac{TRANSACTIONS}{TRANSACTIONS_PER_ROLLBACK} = total_rollback_segments_required$$

total_rollback_segments_required は切り上げ後の数値です。

インスタンスは、起動時に次の方法でロールバック・セグメントを取得します。

- 最初に、ROLLBACK_SEGMENTS 初期化パラメータで指定されたプライベート・ロールバック・セグメントを取得する。*total_private_rollback_segments* の数値が *total_rollback_segments_required* よりも大きいときは、新たなロールバック・セグメントの取得は行いません。
- 初期化ファイルでプライベート・ロールバック・セグメントを指定していないときは、パブリック・ロールバック・セグメントの取得を試みる。
- *total_private_rollback_segments* の数値が *total_rollback_segments_required* より小さいときは、パブリック・ロールバック・セグメントを取得することでその差を補完する。
- プライベート・ロールバック・セグメントを 1 つだけ指定し、インスタンスがそれを取得した場合、または、1 つのパブリック・ロールバック・セグメントを取得した場合は、たとえロールバック・セグメント 1 つでは *total_rollback_segments_required* に満たない場合でもインスタンスは起動する（その場合はメッセージが発生する）。
- インスタンスの起動時にプライベート・ロールバック・セグメントをオンラインにできない場合は、起動は失敗し、メッセージが発生する。

関連項目： 詳細は、14-6 ページの「ロールバック・セグメントの監視」および『Oracle8*i* SQL リファレンス』を参照してください。

7

ロック・メカニズムの概要

この章では、Oracle Parallel Server（OPS）の内部ロック・メカニズムの概要を示します。この章では、次の項目について説明します。

- Oracle ロック・メカニズムの区別
- ロックのコスト
- Oracle のロック名
- IDLM によるロック・メカニズムの調整

Oracle ロック・メカニズムの区別

この項では、次の項目について説明します。

- 概要
- ローカル・ロック
- インスタンス・ロック
- LCK プロセス
- LMON および LMD0 プロセス

概要

パラレル処理およびパラレル・データベース機能を有効に利用するには、ロック・メカニズムについて理解している必要があります。初期化パラメータの設定、システムの管理、効率のよいアプリケーション設計は、すべての種類のロックに影響を与えます。ロックを効率よく使用しないと、共有リソースの同期化に時間がかかりすぎてスピードアップとスケールアップを実現できません。パラレル・システムのパフォーマンスが低下することもあります。

OPS では、次の 2 つの主要な目的のためにロックを使用します。

- トランザクションの分離
- キャッシュの一貫性

トランザクション・ロックは、トランザクションの一貫性を確保するために行レベル・ロックをインプリメントします。行レベル・ロックは、標準の単一インスタンス Oracle および OPS の両方でサポートされます。

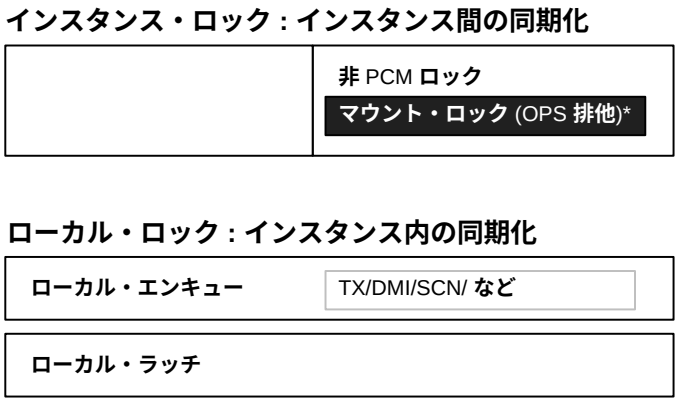
インスタンス・ロック（一般的には分散ロックとも言う）は、キャッシュの一貫性を保証します。インスタンス・ロックは、同じデータベースに属する複数のインスタンスに分散されたデータおよびその他のリソースの一貫性を確保します。インスタンス・ロックには、PCM ロックおよび非 PCM ロックがあります。

> **関連項目：** Oracle のロックの詳細は、第 8 章「統合分散ロック・マネージャ」および『Oracle8*i* 概要』を参照してください。

ローカル・ロック

図 7-1 に、ラッチおよびエンキューを示します。これらは、単一インスタンスの中で同期化するロック・メカニズムです。これらのロック・メカニズムは、パラレル・サーバーの使用可能または使用禁止にかかわらず、標準の単一インスタンス Oracle および OPS で使用されます。

図 7-1 ロック・メカニズム : Oracle および OPS が使用禁止のとき



* マウント・ロックが取得されるのは、ユーザーの Oracle 実行ファイルにパラレル・サーバー・オプションがリンクしている場合です。

ラッチ

ラッチは、SGA のメモリー上のデータ構造体をプロテクトする、単純で低レベルの直列化メカニズムです。ラッチはデータ・ファイルをプロテクトするものではありません。ラッチは完全に自動化されており、排他モードでごく短時間保持されます。内部ロックおよびラッチはノードにとってローカルなものなので、ノード間の同期を確保する機能はありません。

エンキュー

エンキューは、データベース・リソースへのアクセスを直列化する共有メモリー構造体です。このロックは、1 つのインスタンスを対象としてローカルにすることも、データベース全体を対象としてグローバルにすることもできます。エンキューは 1 つのセッションまたはトランザクションに関連付けられ、すべてのモードで使用できます。

- 共有
- 排他
- プロテクト読込み
- プロテクト書込み
- 同時読込み
- 同時書込み
- NULL

エンキューは、ラッチより長く保持され、細分性が高くてモード数も多く、データベース・リソースを幅広くプロテクトします。たとえば、ユーザーが表ロック（または DML ロック）を要求すると、" エンキュー " を受け取ります。

OPS が無効のときは、一定のエンキューが単一インスタンスに対してローカルになります。しかし、OPS が有効な場合は、エンキューをシステム・レベルで管理する必要があります。エンキューは、統合分散ロック・マネージャ（IDLM）によって管理されます。

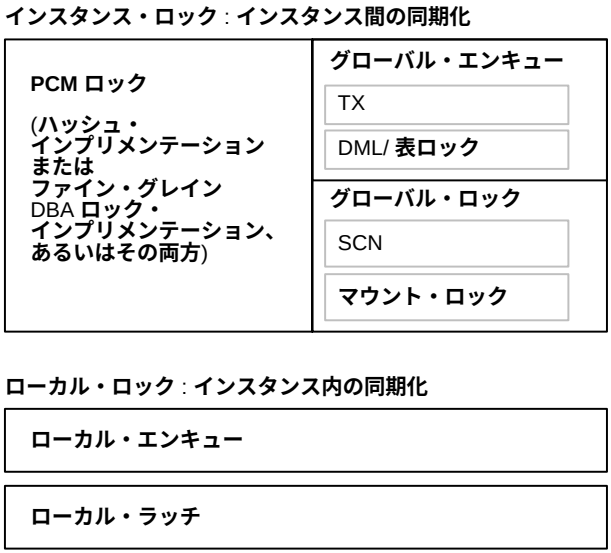
OPS を有効にすると、多くのローカル・エンキューがグローバル・エンキューになります。これを図 7-1 および図 7-2 で説明します。固定表の中では単にエンキューとして示され、ローカル・エンキューとグローバル・エンキューの区別はありません。グローバル・エンキューは分散方式で取り扱われます。

注意： トランザクション・ロックは単にエンキューのサブセットです。

インスタンス・ロック

図 7-2 に、OPS でのインスタンス・ロックの使用を示します。OPS を実装したシステムでは、IDLM がすべての Oracle ロック・メカニズムの状態を追跡および調整します。

図 7-2 ロック・メカニズム : パラレル・サーバーが使用可能なとき



インスタンス・ロック（マウント・ロック以外）は、OPS を有効にして Oracle インスタンスを起動した場合にだけ成立します。インスタンス・ロックは、リソースの現在の状態を複数のインスタンスに伝達することで、OPS のインスタンス間の同期を確保します。

インスタンス・ロックは、トランザクションではなく、インスタンスのバックグラウンド・プロセスによって保持されます。データ・ブロックまたはデータ・ディクショナリ・エントリなどのリソースを SGA に入れると、インスタンスはそれらを保護するインスタンス・ロックを所有します。

キャッシュの一貫性を確保するために IDLM が扱うロックは、1 つ以上の OPS インスタンスによってアクセスされるリソースだけです。IDLM は、インスタンス・ロックの要求およびロックの状態をそれぞれのインスタンスのロック・プロセスに通知します。表 7-1 に、IDLM に対応したビューを示します。

表 7-1 IDLM ビュー

IDLM ビュー名	内容
V$DLM_LOCKS	IDLM によって現在認識されている、ブロックしているロックおよびブロックされているロックに関する情報を表示する。
V$DLM_ALL_LOCKS	すべてのロックとその状態を表示する。
V$DLM_RESS	すべてのリソースとその状態を表示する。

インスタンス・ロックには、パラレル・キャッシュ管理（PCM）ロックおよび非 PCM ロックの 2 つのタイプがあります。

PCM ロック

PCM ロックは、バッファ・キャッシュ内の 1 つ以上のデータ・ブロック（表または索引ブロック）を対象とするインスタンス・ロックです。PCM ロックでは、トランザクションのための行ロックはできません。PCM ロックは、次の 2 つの方法でインプリメントされます。

ファイン・グレイン・ロック	これはデフォルトの方法で、PCM ロックは動的にブロックに割り当てられます。
ハッシュ・ロック	PCM ロックはデータ・ファイル内のブロックに静的に割り当てられます。

ハッシュ・ロックでは、他のインスタンスから要求が出るまで PCM ロックの所有は解除されませんしたがって、リソースに対する競合が比較的少ないシステムでは、インスタンス・ロック・オペレーションのオーバーヘッドが少なくてすみます。ファイン・グレイン・ロックの場合は、ブロックを解放すると、ロックも解放されます。

非 PCM ロック

非 PCM ロックには、さまざまな種類があります。非 PCM ロックは、データ・ファイルと制御ファイルへのアクセスの制御、ライブラリ・キャッシュとディクショナリ・キャッシュの制御、インスタンス間でのさまざまなタイプの通信を実行します。これらのロックには、データ・ファイル・ブロックをプロテクトする機能はありません。この種のロックの例としては、DML エンキュー（表ロック）およびトランザクション・エンキュー、DDL またはディクショナリ・ロックなどがあります。システム変更番号（SCN）およびマウント・ロックはグローバル・ロックであり、エンキューではありません。

注意： OPS 環境では、ほとんどのローカル・エンキューがグローバルになります。これらは、エンキューを表示する固定表および V$LOCK などのビューで見られます。ただし、V$LOCK 表には、SCN ロックおよびマウント・ロック、PCM ロックなどのようなインスタンス・ロックは示されません。

非 PCM ロックより多数の PCM ロック

一般に、PCM ロックの数量は、非 PCM ロックを大幅に上回ります。しかし、非 PCM ロックもかなり多いので、適切な IDLM 容量を慎重に計画する必要があります。通常、ロックの 5% ～ 10% が非 PCM ロックです。非 PCM ロックの量は、PCM ロックほど大きく増加しません。

PCM ロックの数量は、初期化パラメータを設定して必要な数量を割り当てることで、詳細に制御できます。しかし、非 PCM ロックの数量は、制御する方法がほとんどありません。DML_LOCKS = 0 を設定するか、または ALTER TABLE ENABLE / DISABLE TABLE LOCK コマンドを使用すれば、表ロックの必要性をなくすことができますが、それでも他の非 PCM ロックはそのまま存続します。

関連項目： 詳細は、第 16 章「リソースおよびロック用の IDLM 容量の確保」を参照してください。

LCK プロセス

LCK は、OPS でインスタンス間でのロックを実現します。LCK は、インスタンスが使用するほとんどのロックを管理し、他のインスタンスからのロック要求を調整します。LCK は、すべての PCM ロック（ハッシュまたはファイン・グレイン）と、一部の非 PCM ロック（行キャッシュ・ロックまたはライブラリ・キャッシュ・ロックなど）を管理します。LCK は、PCM ロックおよび非 PCM ロックを管理します。

インスタンス・ロックの多くは LCK によって管理されますが、他のバックグラウンド・プロセスまたはシャドウ・フォアグラウンド・プロセスによって直接取得される場合もあります。一般に、LCK のようなバックグラウンド・プロセスがインスタンス・ロックを所有する場合は、そのインスタンス全体がロックの対象となります。フォアグラウンド・プロセスがインスタンス・ロックを所有する場合は、その特定プロセスだけがロックの対象となります。たとえば、ログ・ライター（LGWR）は SCN インスタンス・ロックを取得し、データベース・ライター（DBWR）はメディア回復ロックを取得します。ただし、これらのロックの大部分を LCK プロセスが管理します。

トランザクション・ロックは、LCK ではなく、フォアグラウンド・プロセスが取得します。トランザクション・ロックは、プロセスではなくセッション / トランザクション単位に関連付けられます。

関連項目： LCK の詳細は、『Oracle8*i* 概要』を参照してください。

LMON および LMD0 プロセス

LMON および LMD0 プロセスは、OPS のグローバル・ロック管理サブシステムを実現します。Oracle のシャドウ・プロセスまたは別の Oracle インスタンスの異常終了後に、LMON はロックのクリーン・アップおよびロックの無効化を実行します。また、OPS インスタンスが起動および停止したとき、LMON はグローバル・ロックの再構成および再配分も行います。

LMD0 プロセスは、グローバル・ロックのリモート・ロック要求（現行インスタンスによって所有されたロックに対する他のインスタンスからのロック要求）を処理します。OPS インスタンス宛のすべてのグローバル・ロック・メッセージは、そのインスタンスの LMD0 プロセスによって処理されます。

ロックのコスト

ロックを効率よくインプリメントするには、ロックの相対的コストを慎重に査定します。大まかな目安を次に示します。

- ラッチはコストが低い
- ローカル・エンキューはややコストが高い
- インスタンス・ロックおよびグローバル・エンキューは非常にコストが高い

一般に、インスタンス・ロックおよびグローバル・エンキューは、どちらもパフォーマンスに与える影響は同程度です。OPS が無効のとき、すべてのエンキューはローカルです。OPS が有効なとき、エンキューの多くはグローバルです。

表 7-2 に、ラッチ、エンキューおよびインスタンス・ロックの相対的コストを示します。ロックあたりの所要時間は、システムごとに異なります。次の表の " 実際の所要時間 " で示した値は、一例にすぎません。

表 7-2 ロックの相対コストの比較

ロックのクラス	実際の所要時間	相対所要時間
ラッチ	1 マイクロ秒	1 分
ローカル・エンキュー	1 ミリ秒	1000 分（16 時間）
インスタンス・ロック（またはグローバル・エンキュー）	1/10 秒	100,000 分（69 日）

マイクロ秒、ミリ秒および 1/10 秒などの値は、無視してもいいような時間単位だと考えられがちです。しかし、" 相対所要時間 " で示したように、ロックのコストを極端に大きい値に置き換えてみると、システムおよびアプリケーションでのロックの使用を入念に検討する必要があることがよくわかります。たとえば、大規模な OLTP システムでは、インスタン

ス・ロックを無制限に使用しないでください。1 つのトランザクションの完了までに何時間も何日も待つような状況を想像してみてください。

特定の機能を実行するアプリケーションの場合は、ストアド・プロシージャを使用して、アプリケーションで使用する PCM ロック数を分析できます。初期化パラメータに値を設定してからストアド・プロシージャを起動すると、ロックの観点でコストを見積ることができます。

> **関連項目：** 詳細は、第 15 章「PCM インスタンス・ロックの割当て」および第 16 章「リソースおよびロック用の IDLM 容量の確保」を参照してください。

Oracle のロック名

この項では、次の項目について説明します。

- ロック名の形式
- PCM ロック名
- 非 PCM ロック名

ロック名の形式

Oracle のすべてのエンキューおよびインスタンス・ロックは、次のいずれかの形式を使用して命名されます。

type ID1 ID2

または *type, ID1, ID2*

または *type* (*ID1, ID2*)

項目の内容は、次のとおりです。

type	ロック・タイプを表す 2 文字のタイプ名で、V$LOCK 表の中で記述されています。表 7-3 および表 7-4 に、タイプ名を示してあります。
ID1	IDLM が使用する第 1 のロック識別子。この識別子に関する規則はロック・タイプによって異なります。
ID2	IDLM が使用する第 2 のロック識別子。この識別子に関する規則はロック・タイプによって異なります。

たとえば、領域管理ロックの名前は ST 1 0 のようになります。PCM ロックの名前は BL 1 900 のようになります。

V$LOCK 表は、ローカル・インスタンスによって現在保持または要求された、ローカルおよびグローバル Oracle エンキューを表示します。" ロック名 " は事実上、リソースの名前です。ロックは、リソースに対して取得されます。

PCM ロック名

PCM ロックは、すべてバッファ・キャッシュ管理ロックです。

表 7-3 PCM ロック・タイプおよびロック名

タイプ	ロック名
BL	バッファ・キャッシュ管理

PCM ロック名の構文は、*type ID1 ID2* です。

type PCM ロックはバッファ・ロックであるため、これは常に BL です。

ID1 ブロック・クラスを表します。

ID2 固定ロックの場合、*ID2* はブロック・アドレスをハッシュすることで得られるロック要素（LE）索引番号を表します（V$LOCK_ELEMENT 固定ビューを参照）。解放可能なロックの場合、*ID2* はデータベース・ブロック・アドレスです。

次に、PCM ロック名のサンプルをいくつか示します。

BL (1, 100) これは、ロック要素 100 を伴うデータ・ブロックです。

BL (4, 1000) これは、ロック要素 1000 を伴うセグメント・ヘッダー・ブロックです。

BL (27, 1) ロールバック・セグメント #10 を含むロールバック・セグメント・ヘッダーです。ロールバック・セグメント用の公式は、7 + (10 * 2) です。

非 PCM ロック名

非 PCM ロックには多くの名前があります。

表 7-4 非 PCM ロック・タイプおよびロック名

タイプ	ロック名
CF	制御ファイル・トランザクション
CI	クロスインスタンス・コール起動
DF	データ・ファイル
DL	ダイレクト・ローダー索引作成
DM	データベース・マウント
DX	分散回復

表 7-4 非 PCM ロック・タイプおよびロック名

タイプ	ロック名
FS	ファイル・セット
KK	REDO ログ "Kick"
IN	インスタンス番号
IR	インスタンス回復
IS	インスタンス状態
MM	マウント定義
MR	メディア回復
IV	ライブラリ・キャッシュ無効化
L[A-P]	ライブラリ・キャッシュ・ロック
N[A-Z]	ライブラリ・キャッシュ・ピン
Q[A-Z]	行キャッシュ
PF	パスワード・ファイル
PR	プロセス起動
PS	パラレル・スレーブ同期化
RT	REDO スレッド
SC	システム・コミット番号
SM	SMON
SN	順序番号
SQ	順序番号エンキュー
SV	順序番号値
ST	領域管理トランザクション
TA	トランザクションの回復
TM	DML エンキュー
TS	一時セグメント（および表領域）
TT	一時表
TX	トランザクション
UL	ユーザー定義ロック
UN	ユーザー名

表 7-4 非 PCM ロック・タイプおよびロック名

タイプ	ロック名
WL	REDO ログ書込み開始
XA	インスタンス登録属性ロック
XI	インスタンス登録ロック

関連項目： 非 PCM ロックの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

IDLM によるロック・メカニズムの調整

IDLM コンポーネントは、OPS の内部にある分散リソース・マネージャです。この項では、IDLM が Oracle の内部のロック・メカニズムを調整する方法を説明します。IDLM の機能と特徴の詳細は、第 8 章「統合分散ロック・マネージャ」で説明しています。

この項では、次の項目について説明します。

- IDLM によるロック・モードの追跡
- インスタンスによる IDLM リソースへのデータベース・リソースのマッピング
- IDLM ロックとインスタンス・ロックの相互関係
- IDLM はリソースに対して 1 インスタンスにつき 1 ロックを提供

IDLM によるロック・モードの追跡

OPS を実装したシステムでは、IDLM 機能が、システムのリソースに対して保持されている Oracle インスタンス・ロックおよびグローバル・エンキューのインベントリを維持します。IDLM は、対立するロック要求が生じたときの調停役として働きます。

図 7-3 では、OPS でのリソースと、各リソースに対するロックの現在の状態を IDLM のインベントリ・シートとして表現しています。ロックの状態は、S が共有モード、N が NULL モード、そして X が排他モードです。

図 7-3　Oracle リソースおよびロックの IDLM インベントリ

統合 DLM

リソース	ロック
BL 1, 100	S
BL 1, 101	SSSNNN
BL 4, 3000	X
BL 4, 3001	SSS
BL 6, 100	NNN
BL 6, 101	X
BL 8, 3000	X
BL 8, 3001	N
BL 9, 4000	N

このインベントリにはすべてのインスタンスが含まれています。たとえば、リソース BL 1, 101 では、3 つのインスタンスが共有ロックで、3 つのインスタンスが NULL ロックで実行されています。表は 1 つのリソースについて最大 6 つのロックを示しているので、このシステムでは少なくとも 6 つのインスタンスが実行されていることがわかります。

インスタンスによる IDLM リソースへのデータベース・リソースのマッピング

Oracle データベース・リソースは IDLM リソースにマッピングされますが、必要なマッピングはインスタンスが行います。たとえば、特定のデータ・ブロック・アドレスを持つ Oracle データベース・ブロック（たとえばファイル 2 ブロック 10）に対するハッシュ・ロックは、そのブロックのクラスおよびロック要素番号を持つ BL リソース（たとえば BL 9 1）として解釈されます。データ・ブロック・アドレス（DBA）は、Oracle（データベース）リソース・レベルから IDLM リソース・レベルに変換されます。使用されるハッシュ関数は、GC_* パラメータの設定値によって決まります。IDLM リソース名から物理（データベース）リソースを識別するには、V$LOCK などのビューを参照します。

注意：　DBA ファイン・グレイン・ロックの場合は、ロック要素番号のかわりにデータベース・アドレスが第 2 識別子として使用されます。

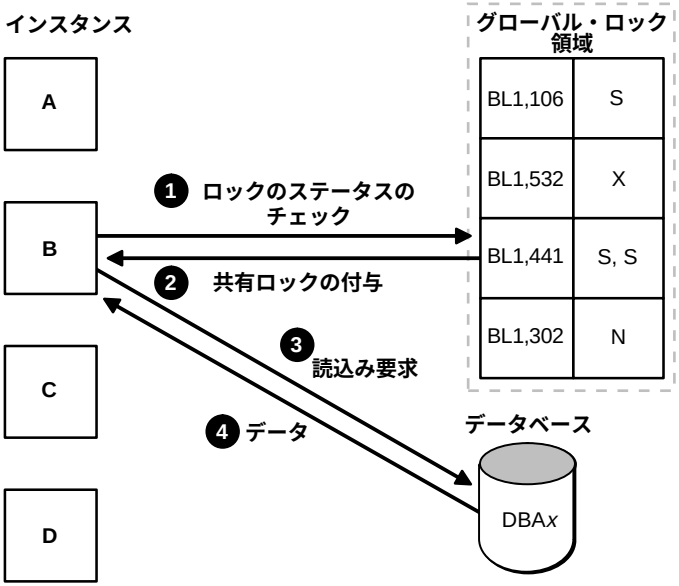
図 7-4 データベース・リソース名と IDLM リソース名の対応付け

IDLM ロックとインスタンス・ロックの相互関係

図 7-5 に、IDLM ロックと PCM ロックの関係を示します。インスタンス B がデータ・ブロック・アドレス *x* のデータ値を読み込むには、最初にそのデータに対するロックをチェックする必要があります。インスタンスは、ブロックのデータベース・リソース名を IDLM リソース名に変換し、データの読込みのために IDLM に共有ロックを要求します。

IDLM は、認可されているキューに対する未解決のロックをチェックし、リソース BL1,441 にすでに 2 つの共有ロックがあることを確認します。共有ロックは読込み専用要求と対立しないので、IDLM はインスタンス B に共有ロックを付与します。次に、そのインスタンスが、データ・ブロック・アドレス *x* のデータを読み込むための問合せをデータベースに出します。データベースはデータを戻します。

図 7-5　IDLM によるロックの状態のチェック



注意： グローバル・ロック・スペースは、すべてのインスタンスの LMD の共同作業により、分散方式で管理されます。

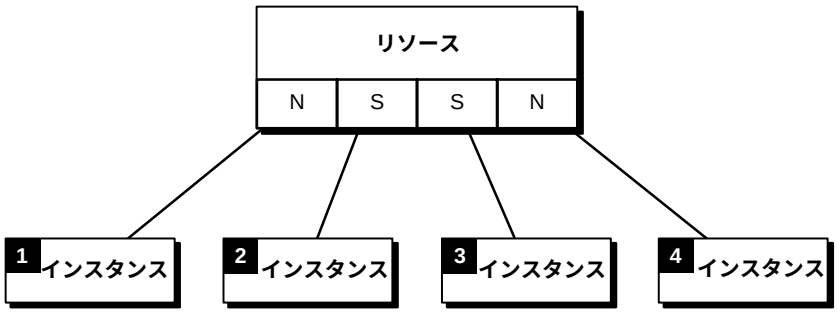
要求したブロックに他のインスタンスによる排他ロックがすでに設定されている場合は、インスタンス B はそのロックが解放されるまで待つ必要があります。IDLM は、インスタンス B からの共有ロック要求を変換キューに入れます。排他ロックが取り除かれた時点で IDLM がインスタンスに通知し、そのインスタンスの共有ロック要求を許可します。

IDLM ロックとは、単に、リソースに対する未解決のロックの追跡および調整を意味する IDLM の用語です。

IDLM はリソースに対して 1 インスタンスにつき 1 ロックを提供

IDLM は、PCM リソースに対して 1 インスタンスにつき 1 ロックを提供します。図 7-6 に示すように、4 つのインスタンスからなるシステムがあり、1 つのリソースに対するバッファ・ロックが必要な場合は、1 インスタンスにつき 1 ロックとして、合計 4 つのロックが実際に生じます。

図 7-6 1 インスタンスにつき 1 ずつロックが要求されたリソース



非 PCM ロックの数はロックのタイプによって異なります。

関連項目： 詳細は、第 10 章「非 PCM インスタンス・ロック」を参照してください。

8

統合分散ロック・マネージャ

この章では、Oracle パラレル・サーバー（OPS）リソースへのアクセス制御での統合分散ロック・マネージャ（IDLM）の役割を説明します。次の項目について説明します。

- 統合分散ロック・マネージャとは
- IDLM によるリソース・ロック要求の許可および調整
- IDLM ロック・モード : リソース・アクセス権
- IDLM 機能

統合分散ロック・マネージャとは

Oracle8*i* の IDLM コンポーネントは、システム・リソースのリストを維持し、Oracle（のデータベース）リソースの割当ておよび変更を制御するロック・メカニズムを提供します。IDLM リソースは複数の論理的概念、データ構造です。IDLM は、データベース本体に保管された表またはオブジェクトへのアクセスを制御するものではありません。IDLM によってプロテクトされたデータベース・リソースを必要とするプロセスは、そのリソースに対するロックをオープンする必要があります。

OPS では IDLM を使用してデータ・ブロックおよびロールバック・セグメントなどのリソースへの複数のインスタンスでの同時アクセスを調整します。

IDLM によるリソース・ロック要求の許可および調整

この項では、IDLM がどのような方法でリソース・ロック要求を調整するのかを説明します。次の項目について説明します。

- ロック要求の待機
- 非同期トラップ（AST）によるロック要求状態の通知
- ロック要求の変換および許可

IDLM はロック要求を調整して、リソースへのアクセス権を確保します。このプロセスでは、IDLM はすべてのロック要求を追跡します。要求されたリソースが使用可能であればアクセスを許可し、アクセス権を追跡します。要求されたリソースが現在使用可能でない場合はリソースに対する要求を追跡し、リソースが使用可能になった時点でアクセス権を付与します。IDLM はこれらのロック要求のインベントリを維持し、状態を関連するユーザーおよびプロセスに知らせます。

ロック要求の待機

IDLM は、ロック要求の 2 つのキューを維持します。

変換キュー	ロック要求をすぐに許可できないときは、ロック要求を変換キューに入れます。変換キューでは、待機中のロック要求が追跡されます。
許可キュー	許可されたロック要求は、許可キューで追跡されます。

非同期トラップ（AST）によるロック要求状態の通知

IDLM は、2 種類の非同期トラップ（AST）または割込みを使用して、ロック要求の状態を関連するユーザーおよびプロセスに知らせます。

獲得 AST	要求したモードでロックが取得されると、リクエスタ（ロックの要求を出したプロセス）に獲得 AST（" ウェイクアップ・コール "）を送信してロックが許可されたことを知らせます。
ブロッキング AST	プロセスがリソースに対して特定モードのロックを要求すると、IDLM はブロッキング AST を送信して、現在そのリソースに対する非互換モードのロックを所有しているプロセスに通知します。（たとえば、共有モードと排他モードには互換性がありません。）ロックの所有プロセスは、通知を受けると、ロックを放棄して、リクエスタにアクセスを許可することができます。

ロック要求の変換および許可

次の図は、IDLM がロック要求をどのように扱うかを示しています。

図 8-1 では、リソースの共有ロック要求 1 がプロセス 1 に対して許可され、共有ロック要求 2 がプロセス 2 に対して許可されています。これらのロックは、許可キュー内で追跡されます。プロセス 2 が排他ロックを要求した場合、プロセス 2 は変換キュー内で待機する必要があります。

図 8-1 IDLM の許可キューおよび変換キュー

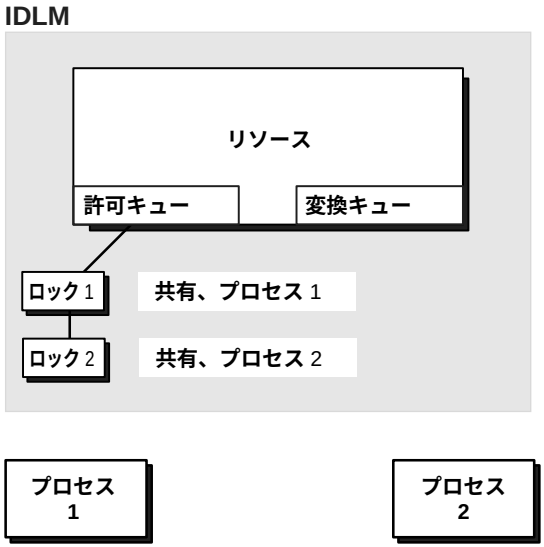
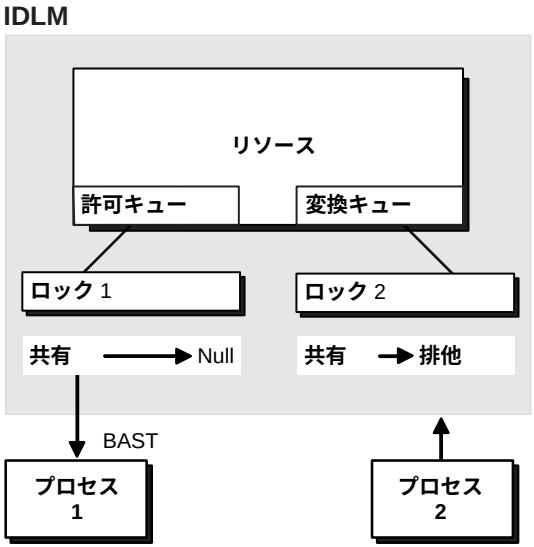


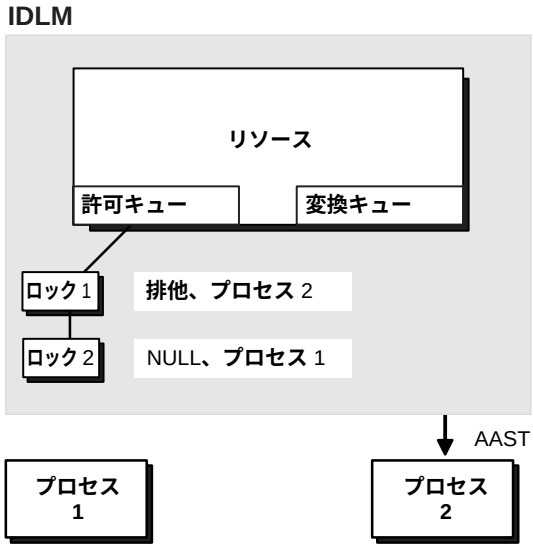
図 8-2 に示すように、IDLM は共有ロックの所有者であるプロセス 1 にブロッキング AST を送って、排他ロックの要求が待機中であることを通知します。プロセス 1 が共有ロックを手放すと、そのロックは NULL モード・ロックに変換または解放されます。

図 8-2 ブロッキング AST



次に、獲得 AST が、排他ロックのリクエスタである待機中のプロセス 2 に送られ、活動を再開します。排他ロックが許可されると、許可キューに変換されます。図 8-3 に獲得 AST を示します。

図 8-3 獲得 AST


IDLM
リソース
許可キュー
変換キュー
ロック 1
排他、プロセス 2
ロック 2
NULL、プロセス 1
AAST
プロセス 1
プロセス 2

IDLM ロック・モード : リソース・アクセス権

インスタンスはロックを使用してリソースに対するさまざまな権利を取得します。最初は、アクセス権が何も付与されていないロックが、リソースに対して作成されます。その後、プロセスがロックを変換して新しいアクセス権を取得します。

図 8-4 に、IDLM で使用できるアクセス権のレベルまたは " ロック・モード " を示します。

図 8-4 IDLM ロック・モード : アクセスのレベル

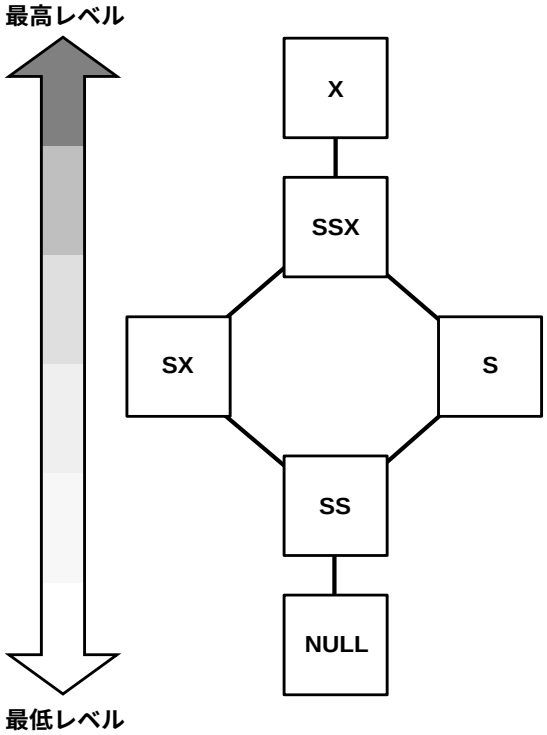


表 8-1 ロック・モード名

Oracle モード	概要	説明
NULL	NULL モード。リソースはロックされていない	このレベルのロックには、アクセス権は何も付与しません。通常、このレベルのロックは、リソースを必要としているプロセスがあることを示します。または、プレース・ホルダとして使用されます。 NULL ロックの作成により、リクエスタがいつでもそのリソースをロックできることが保証されます。アクセス権が必要になるたびに、IDLM がロックを作成したり破棄したりする必要はありません。
SS	半共有モード（同時読込み）。読込みアクセス。書込みおよび他の読込みアクセスも許可される	ロックがこのレベルで設定されているときは、プロテクトなしで対応リソースの読込みができます。つまり、他のプロセスにも対応リソースの読込みと書込みが許可されます。
SX	共有排他モード（同時書込み）。書込みアクセス。他の読込みや書込みアクセスも許可される	ロックがこのレベルで設定されているときは、プロテクトなしで対応リソースの読込みまたは書込みができます。つまり、他のプロセスにもリソースの読込みと書込みが許可されます。
S	共有モード（プロテクト読込み）。 読込みアクセス。書込みアクセスは許可されない	ロックがこのレベルで設定されているときは、対応リソースの書込みはできません。このリソースの読込みは、複数のプロセスに許可されます。これが従来の一般的な共有ロックです。 共有モードでは、ユーザーが何人でも同時に、リソースへの読込みアクセスができます。共有アクセスは、読取り操作専用です。
SSX	半共有排他モード（プロテクト書込み）。1 つの書込みアクセスのみ。他の読込みアクセスは許可される	このレベルでロックを設定できるのは 1 つのプロセスだけです。このロックが設定されたプロセスはリソースを変更することができ、他のプロセスは同時にそのリソースを変更できません。他のプロセスは、プロテクトなしの読込みができます。一般的な更新ロックです。
X	排他モード。 書込みアクセス。他のアクセスはいっさい許可されない	ロックがこのレベルで保持されているときは、ロックを保持しているプロセスにリソースへの排他アクセスが許可されます。他のプロセスは、そのリソースの読込みも書込みもできません。これは従来の排他ロックに相当します。

IDLM 機能

この項では次の IDLM 機能について説明します。

- 分散アーキテクチャ
- フォールト・トレラント
- ロックのマスター化
- デッドロックの検出
- Lamport SCN 生成
- グループ・ベースのロック
- 持続リソース
- メモリー要件
- MTS および XA に対するサポート
- IDLM 統計を監視するビュー

分散アーキテクチャ

IDLM は、リソースのデータベースとリソースに対するロックをそれぞれ別のモードで維持します。このロック・データベースは揮発性メモリーに常駐し、分散しています。

IDLM は分散アーキテクチャを使用しています。分散アーキテクチャでは、クラスタ内の各ノード（または、Oracle データベースの各 OPS インスタンス）がグローバル・ロックの管理に参加し、グローバル・ロック・データベースの一部を管理します。ロック・データベースは、関与するすべてのノードに分散されます。このような分散ロック管理スキームは、フォールト・トレラント性を提供し、実行時パフォーマンスを強化します。

フォールト・トレラント

IDLM はフォールト・トレラントです。つまり、ノードおよび OPS インスタンスで複数の障害が発生しても、サービスを継続させ、ロック・データベースの整合性を維持します。回復後に、データベースでアクティブな OPS インスタンスが 1 つでも存在する限り、そのデータベースにアクセスできます。

また、フォールト・トレラントであることにより、OPS インスタンスをいつでも、任意の順番で起動および停止できます。ただし、インスタンスの再構成によって多少の遅延が生じることがあります。

ロックのマスター化

IDLM は、特定のリソースへのアクセスが必要なすべてのノードのロック情報を維持します。IDLM は通常、あるリソースとそのロックについてのすべての関連情報を管理するノードを 1 つ指名します。このノードを（そのリソースの）" マスター・ノード " と言います。

OPS では静的ハッシング・ロック・マスター・スキームを使用します。マスター化プロセスでは、あるリソース名が OPS インスタンスの 1 つにハッシュされ、そのインスタンスがそのリソースのマスターとして機能します。このスキームにより、リソースはすべての使用可能ノードに均等、公平に分散します。すべてのリソースはそれぞれ、マスター・ノードに対応付けられています。

IDLM は、状況に応じて使用するロック・マスター化の方法を最適化します。使用するロック・マスター化の方法により、インスタンスの起動時だけでなく、通常の実行時アクティビティのシステム・パフォーマンスにも影響があります。リソースがローカルでマスター化されていれば、パフォーマンスは最も高くなります。

リソースがリモートでマスター化された場合、そのリソースへの競合アクセスが生じるたびに、そのリソースのマスター・ノードへメッセージが転送されます。これにより、ノード間でのメッセージ・トラフィックが増加し、システム・パフォーマンスが低下します。

デッドロックの検出

IDLM はデッドロックの影響を受けるすべてのロックおよびリソースに対してデッドロック検出を実行します。

Lamport SCN 生成

OPS が使用する Lamport SCN 生成スキームは、SCN を全インスタンスに同時に生成でき、高速かつ拡張性のあるスキームです。

関連項目： 4-8 ページの「Lamport SCN 生成」

グループ・ベースのロック

グループ・ベースのロックでは、所有権は動的なものとなります。つまり、同じグループに属する複数のプロセスが 1 つのロックを共有できます。同じグループに属するプロセスは、新しいロックのオープンまたは別のロックへの変換を行うことなく、ロックの共有または操作（あるいはその両方）を行うことができます。

関連項目： 8-10 ページの「MTS および XA に対するサポート」

持続リソース

IDLM は持続的なリソースを提供します。したがって、そのリソースにロックを要求したすべてのプロセスまたはグループが異常終了した場合でも、リソースはその状態を維持します。

関連項目： 8-3 ページの「ロック要求の変換および許可」

メモリー要件

ユーザー・レベルの IDLM では通常、ユーザーが要求するだけリソースを割り当てることができます。ただし、その数が多いほどプロセス・サイズが大きくなります。ロックおよびリソースが常駐する共有メモリーをアドレス領域にマッピングするからです。その結果、プロセス・アドレス領域が非常に大きくなることがあります。

アプリケーションで必要なすべてのリソースをサポートできる IDLM 構成であるかを確認してください。

MTS および XA に対するサポート

OPS では、2 つの形式のロック所有権を使用します。

プロセス単位の所有権	通常は、ロックを所有するのはプロセスです。つまり、1 つのプロセスが排他的にロックを所有している場合は、他のプロセスはそのロックを操作することはできません。
グループ・ベースの所有権	グループ・ベースのロックでは、所有権は動的なものとなります。つまり、同じグループに属する複数のプロセスで 1 つのロックを交換できます。同じグループ内のプロセスが、IDLM 許可キューおよび変換キューに移らなくても、ロックを交換またはロックを操作すること（あるいはその両方）ができます。

グループ・ベースのロックは、Oracle マルチスレッド・サーバー（MTS）および XA ライブラリ機能にとって重要な IDLM 機能です。

MTS	Oracle MTS 構成には、グループ・ベースのロックを使用します。グループ・ベースのロックがない場合は、共有サーバー・プロセス間でセッションを移すことができません。しかも、特に長時間実行のトランザクションでは作業ロード・バランシングに支障が生じることがあります。

XA ライブラリ	Oracle XA ライブラリでは、トランザクション上で複数のセッションまたはプロセスが稼働できます。したがって、排他モードの場合でも、それらのセッションまたはプロセスで、同じロックを交換する必要があります。グループ・ベースのロック所有権では、プロセスは排他リソースへのアクセスを交換することができます。

IDLM 統計を監視するビュー

表 8-2 に、IDLM 統計の監視に使用できる 6 つの動的パフォーマンス・ビューを示します。

表 8-2 IDLM 統計を監視するビュー

ビュー	説明
V$DLM_CONVERT_LOCAL	ローカル・ロック変換操作のための変換時間を表示します。EVENT 29700 により使用可能です。
V$DLM_CONVERT_REMOTE	リモート・ロック変換操作のための変換時間を表示します。EVENT 29700 により使用可能です。
V$DLM_LOCKS	ブロックされているロックまたは他のロックをブロックしているロックについて、現在 IDLM に認識されているすべてのロックに関するデバッグ情報を表示します。
V$DLM_ALL_LOCKS	現在 IDLM に認識されているすべてのロックに関するデバッグ情報を表示します。
V$DLM_RESS	すべてのロックで使用されているリソースに関する統計を表示します。
V$DLM_MISC	各種の IDLM 統計を表示します。

関連項目： 動的パフォーマンス・ビューの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

9

パラレル・キャッシュ管理インスタンス・ロック

PCM ロックの計画と割当ては、Oracle Parallel Server（OPS）のデータベース管理者が経験する最も複雑な作業の 1 つです。この章では、PCM ロックの概念を要約して示します。次の項目について説明します。

- PCM ロックおよびその機能
- 初期化パラメータによるブロックおよび PCM ロックの制御方法
- PCM ロックの 2 つの方法 : 固定および解放可能なロック
- Oracle によるブロックへのロックの割当て
- 例 : PCM ロックへのブロックのマッピング

関連項目： 第 15 章「PCM インスタンス・ロックの割当て」（PCM ロックの計画と割当てについて）および第 8 章「統合分散ロック・マネージャ」（IDLM 機能について）

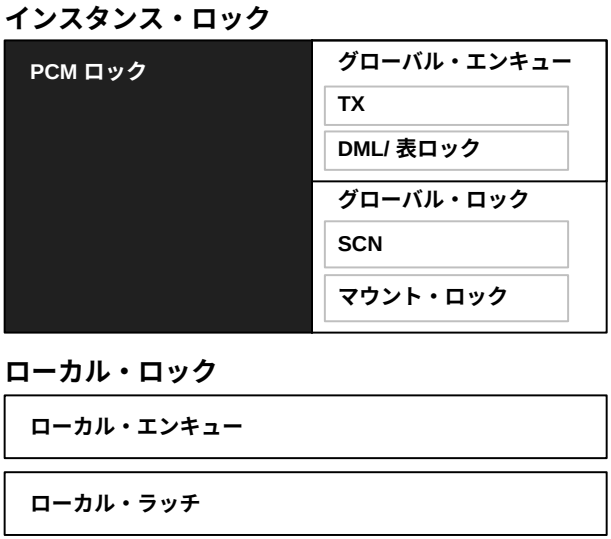
PCM ロックおよびその機能

この項では、次の項目について説明します。

- PCM ロック
- PCM ロックの割当ておよび解放
- PCM ロックの動き
- PCM ロックあたりのブロック数
- ping: 更新の必要性の通知
- ロック・モードおよびバッファ状態

図 9-1 では、PCM ロックと Oracle で使用する他のロックとの関連を示しています。

図 9-1　Oracle ロック・メカニズム : PCM ロック



PCM ロック

パラレル・キャッシュ管理ロック（PCM ロック）は、データ・ファイル内のブロックのロックを管理する、インスタンス・ロックです。PCM ロックは、あらゆるクラスのブロック、たとえば、データ・ブロック、索引ブロック、取消しブロック、セグメント・ヘッダーなどを 1 つ以上受け持つことができます。インスタンス・ロックはいくつかの種類に分けられますが、それらはすべて同じ目的で機能します。

OPS ではインスタンス・ロックを使用して共有リソースへのアクセスを調整し、IDLM がインスタンス・ロックのステータスを管理します。

PCM ロックではキャッシュの一貫性を確保するために、インスタンスがデータベース・ブロックの変更または読込みを行うときは、その前に、要求を出したインスタンスが、ロックを保持しているインスタンスから、ロックを取得することを強制します。PCM ロックを使用すると、同一時点でブロックを変更できるインスタンスは 1 つだけです。インスタンスがブロックを変更した場合は、変更されたブロックをディスクに書き込んでからでなければ、他のインスタンスはそのブロックの PCM ロックを取得することも、変更することもできません。

PCM ロックは、最小限の通信量でキャッシュの一貫性を確保します。インスタンス間の相互アクティビティ量と OPS のパフォーマンスの対応は、ping 数に換算して評価されます。

あるインスタンスが（変更された）ブロックをディスクに書き込まなければ、他のインスタンスはブロックを読み込むことができないので、そのたびに ping が発生します。

負荷の高いシステムでは大量のロック・アクティビティが発生しますが、必ずしも ping が発生しているとは限りません。データが適切に分割されていれば、ロックは各ノードに対してローカルになるため、ping は発生しません。

PCM ロックの割当ておよび解放

最適なパフォーマンスを実現するために、OPS 管理者は PCM ロックをデータ・ファイルに割り当てる必要があります。それには、データベースの起動時に読み込む初期化パラメータの値を指定します。この手順の詳細は、第 15 章「PCM インスタンス・ロックの割当て」で説明しています。

1 つまたは一群のデータ・ファイル内のデータ・ブロックを対象とする（ハッシュ）PCM ロックの数を指定するには、初期化パラメータ GC_FILES_TO_LOCKS を使用します。最小の細分性は、1 データ・ブロックあたり 1 つの PCM ロックです。一般に、PCM ロックは、OPS のインスタンス・ロックの中で最大の割合を占めます。

4 つのタイプの PCM ロックのいずれかを割り当てることが出来ます。これらは、割当て方法、およびロックが解放されるかどうかという点で異なります。

解放可能なファイン・グレイン・ロックの割当て

ファイン・グレイン PCM ロックは、必要に応じて獲得され解放されます。このロックは必要な時にだけ割り当てられるので、ハッシュ・ロックがある場合に比べて、インスタンスは速く起動できます。ユーザーが実際にブロックを要求した場合に限り、IDLM リソースが作成され、IDLM ロックが取得されます。作成されたファイン・グレイン・ロックは、個々のインスタンスの必要に応じて、各種のモードに変換できます。

ファイン・グレイン・ロックは解放可能です。つまり、インスタンスは、通常の操作中に、あるリソース名へのすべての参照をやめることができます。IDLM リソースを別のブロック用に再使用する必要がある場合は、そのリソースが解放されます。したがって、あるリソースに対してどのインスタンスもロックを所有していないこともあります。

固定ハッシュ・ロックの割当て

ハッシュ・ロックは事前に割り当てられ、起動時に静的にブロックにハッシュされます。最初に起動するインスタンスは、個々のハッシュ PCM ロックについて、IDLM リソースを作成し、その IDLM リソースに対する IDLM ロック（NULL モード）を作成します。最初のインスタンスは、各ロックを初期化します。次に、そのインスタンスは、必要に応じて IDLM ロックを他のモードに変換します。2 番目のインスタンスが特定の IDLM ロックを必要とする場合は、インスタンスはそのロックが使用可能になるまで待ってから、ロックを必要なモードに変換します。

デフォルトでは、ハッシュ PCM ロックは解放されることがありません。各ロックは、最後に要求されたときのモードのままになっています。他のインスタンスがそのロックを必要と

する場合は、ロックを NULL モードに変換します。このロックの割当てが解除されるのは、インスタンスのシャット・ダウン時だけです。

解放可能なハッシュ・ロックの割当て

GC_FILES_TO_LOCKS パラメータに R オプションを付けることで、解放可能なハッシュ PCM ロックを指定できます。解放可能なハッシュ PCM ロックは、GC_RELEASABLE_LOCKS プールから取得されます。

固定ファイン・グレイン・ロックの割当て

固定ロックをファイン・グレインの方法で割り当てることもできます。たとえば、特定の 50,000 データベース・ブロックを持つファイルに 50,000 PCM ロックを設定し、各ブロックごとに 1 つの固定ロックを提供することができます。

関連項目： 15-6 ページの「GC_FILES_TO_LOCKS の構文」（GC_FILES_TO_LOCKS パラメータの設定方法）

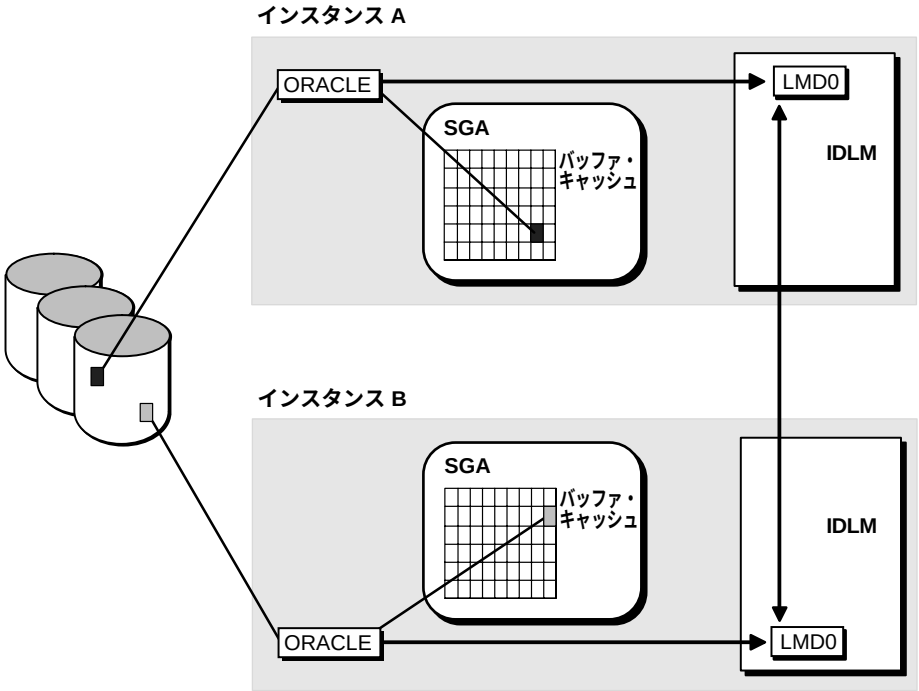
PCM ロックの動き

固定（ハッシュ）PCM ロックは、最初は NULL モードで獲得されます。指定したハッシュ・ロックはすべてインスタンスの起動時に割り当てられ、インスタンスのシャット・ダウン時に割当てが解除されます。したがって、ファイン・グレイン・ロックに比べて、ハッシュ・ロックの方がオーバーヘッドが大きく起動時間も長くなります。しかし、ハッシュ PCM ロックには、頻繁に獲得と解放を繰り返す必要がないという利点があります。

解放可能（ファイン・グレインもしくは、ハッシュ）PCM ロックは、固定ハッシュ・ロックより動的です。たとえば、GC_RELEASABLE_LOCKS を 10000 に設定した場合、最大 10000 のファイン・グレイン PCM ロックを取得できます。ただし、ロックは IDLM が必要とするときだけ割り当てられます。Oracle では、起動時にロック要素を割り当てます。ロック要素は、要求したモード（通常は共有モードまたは排他モード）で直接取得されます。

図 9-2 に、PCM ロックの動きを示します。インスタンス A は、変更のために黒のブロックを読み込むときに、ブロックの PCM ロックを取得します。陰ブロックとインスタンス B についても、これと同じことが行われます。インスタンス B が黒ブロックを必要とする場合は、インスタンス A が黒ブロックを変更したので、そのブロックをディスクに書き込む必要があります。Oracle プロセスは、LMD プロセスと通信して IDLM からインスタンス・ロックを取得します。

図 9-2　PCM ロックの動き



PCM ロックはインスタンス LCK プロセスが所有

各インスタンスには、LCK バックグラウンド・プロセスが少なくとも 1 つずつあります。同じインスタンスの中に複数の LCK プロセスが存在する場合は、PCM ロックはそれらの LCK プロセス間で分割されます。これは、個々の LCK プロセスが、それぞれ PCM ロックのサブセットを受け持てばよいことを意味します。

ロック・モードの変換

ある PCM ロック（共有モードまたは排他モード）の対象となっているブロックが、あるインスタンスに属するバッファ・キャッシュに入ると、そのインスタンスがその PCM ロックを " 所有 " または制御します。

複数のインスタンスで同じロックを所有

あるインスタンスが共有モードで PCM ロックを所有している場合に、別のインスタンスが同じ共有モードの PCM ロックを要求しても、PCM ロックを所有しているインスタンスはそのロックを手放しません。つまり、PCM ロックのコピーが共有されることになるため、2 つ

のインスタンスが、それぞれのバッファ・キャッシュ内に同じデータ・ブロックを持つことになります（書込みは行われません）。同じハッシュ PCM ロックが受け持つ異なるデータ・ブロックが、異なるインスタンスのバッファ・キャッシュに入る場合もあります。これが起こるのは、異なるインスタンスが共有モードで PCM ロックを要求した場合です。

PCM ロックあたりのブロック数

データ・ファイルに割り当てられる PCM ロックの数、およびデータ・ファイル内のデータ・ブロックの数によって、1 つの PCM ロックが受け持つデータ・ブロックの数が決まります。

- ファイルに対して GC_FILES_TO_LOCKS を設定していない場合は、各ブロックに 1 つずつ解放可能ロックが使用されます。
- ファイルに対して GC_FILES_TO_LOCKS を設定してある場合は、1 つのハッシュ PCM ロックあたりのブロック数は、ファイル別に次のように計算できます。この例では、GC_FILES_TO_LOCKS = 1:300,2:200,3-5:100 という値を想定しています。

$$\text{ファイル 1:}\ \frac{\text{ファイル 1 のブロック数}}{\text{300 ロック}}$$

$$\text{ファイル 2:}\ \frac{\text{ファイル 2 のブロック数}}{\text{200 ロック}}$$

$$\text{ファイル 3:}\ \frac{\text{合計数（ファイル 3、ファイル 4、ファイル 5 のブロック数)}}{\text{100 ロック}}$$

ファイルのサイズ（ブロック単位）が、そのファイルに割り当てられたハッシュ PCM ロック数の倍数である場合は、各ハッシュ PCM ロックが受け持つデータ・ブロック数は、前述の式で正確な値が得られます。

ファイル・サイズがハッシュ PCM ロック数の倍数でない場合は、そのデータ・ファイルのハッシュ PCM ロックあたりのデータ・ブロック数に、ブロック 1 個分の誤差が生じる場合があります。たとえば、データ・ブロック数が 2,500 であるデータ・ファイルに 400 個のハッシュ PCM ロックを割り当てると、そのうち 100 個のハッシュ PCM ロックはそれぞれ 7 つずつデータ・ブロックを受け持ち、300 個のハッシュ PCM ロックは 6 つずつブロックを受け持ちます。GC_FILES_TO_LOCKS 初期化パラメータに指定してないデータ・ファイルは、すべて（解放可能）PCM ロックを使用します。

n 個のファイルがいくつかの同じハッシュ PCM ロックを共有している場合は、ロックあたりのブロック数に最大 *n* 個の差が生じます。個別の GC_FILES_TO_LOCKS 句を使用するか

またはキーワード EACH を使用して、ファイル単位でロックを割り当てた場合は、ロックあたりのブロック数に 2 以上の差が生じることはありません。

一群のデータ・ファイルに集合的にハッシュ PCM ロックを割り当てた場合は、通常、各ロックが各ファイル内の 1 つ以上のブロックを受け持つことになります。ただし、連続ブロックを指定した場合（"!*blocks*" オプションを使用）、または、ファイルのブロック数がファイルの集合に割り当てたロック数よりも少ない場合は、例外が生じることがあります。

例

次の例は、ハッシュ PCM ロックが異なるファイル内の複数のブロックをどのように受け持つかを示しています。図 9-3 では、合計で 44 ブロックをもった 2 つのファイルに対して 44 個の PCM ロックが割り当てられています。GC_FILES_TO_LOCKS は A,B:44 に設定されています。

ファイルのブロック 1 は、必ずしもロック 1 から始まっているとは限りません。ファイルがどのロックから始まるかは、ハッシュ関数によって決まります。24 個のブロックからなるファイル A では、ブロック 1 はロック 32 にハッシュします。20 個のブロックからなるファイル B では、ブロック 1 はロック 28 にハッシュします。

図 9-3 複数のファイル内のブロックを受け持つハッシュ PCM ロック

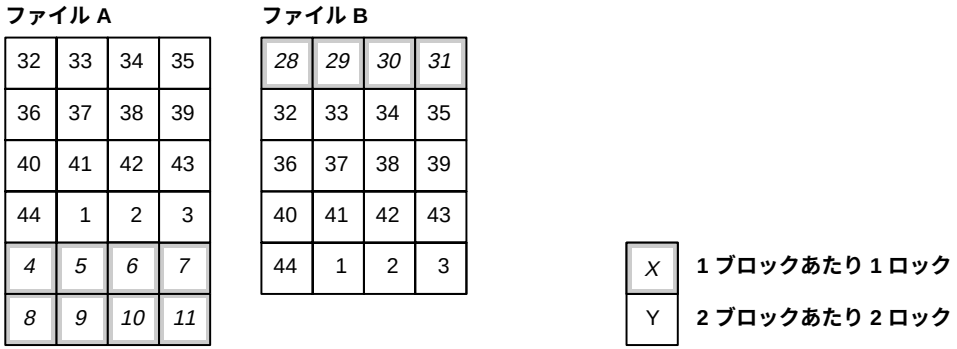


図 9-3 では、ロック 32 ～ 44 および 1 ～ 3 でそれぞれ 2 ブロックずつ受け持っています。ロック 4 ～ 11 および 28 ～ 31 は、それぞれ 1 ブロックずつ受け持っています。ロック 12 ～ 27 は 1 つもブロックを受け持っていません。

最悪のケースとして考えられるのは、2 つのファイルが開始ブロックを同じロックにハッシュしている場合で、その場合はすべての共通ロックがどれも 2 つずつブロックを受け持つことになります。ファイルの規模が大きく、1 ロックあたり複数のブロックがある場合は（おおむね 1 ロックあたり 100 ブロック程度）、これは重大な問題ではありません。

ハッシュ PCM ロックの周期性

PCM ロックの周期性も考慮する必要があります。図 9-4 では、30 ブロックのファイルを 6 個の PCM ロックで受け持っています。このファイルは、ロック番号 5 から始まるようにハッシュされています。PCM ロック番号 4 が受け持つ影付きブロックからわかるように、各ロックが割り当てられた結果、ファイルのブロックに一定のパターンが生じています。

図 9-4 ハッシュ PCM ロックの周期性

5	6	1	2	3
4	5	6	1	2
3	4	5	6	1
2	3	4	5	6
1	2	3	4	5
6	1	2	3	4

ping: 更新の必要性の通知

OPS では、一度に 1 つのインスタンスだけが特定のデータ・ブロックを変更できます。インスタンスが変更したデータ・ブロックを他のインスタンスが要求した場合は、要求元インスタンスがブロックに要求したロック・タイプに応じて、ping が生じる場合とそうでない場合があります。

要求元インスタンスがブロックを変更する場合、要求元インスタンスは、ロックを保持しているインスタンスがデータ・ブロックに対してかけているロックを必要に応じて変換する必要があります。ロックを保持しているインスタンスがそのブロックをディスクに書き込んでからでなければ、要求元インスタンスはそのブロックを読み込むことができません。これをブロックの ping と言います。

BSP（ブロック・サーバー・プロセス）は、2 つのインスタンス間で IDLM 機能を使用して必要なブロックを通知しあいます。インスタンスが CR（一貫性読込み）モードでブロックを要求しているだけであれば、ブロックを保持しているインスタンスの BSP がインターコネクトを介してブロックの CR バージョンを要求元インスタンスへ送ります。この例では、ping は必要ありません。

データ・ブロックが ping されるのは、インスタンスのバッファ・キャッシュ内に排他現行（XCUR）状態で保持されているブロックがあり、他のインスタンスが変更のためにそのブロックを必要としている場合だけです。インスタンスが共有（SHARE）モードでブロックを保持している場合、ブロックが ping されるのは、別のインスタンスがそのブロックを XCUR 状態にする必要があるときです。したがって、データ・ブロックを受け持つ PCM ロックの数が、ブロックでの ping の発生を左右することはほとんどありません。

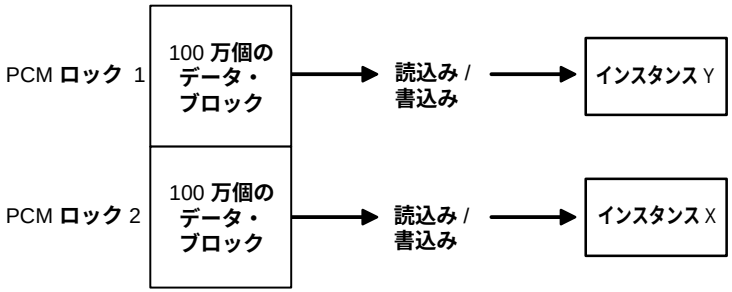
インスタンスは、ブロックの排他ロックを放棄した後も、そのブロックの行ロックを引き続き保持している場合があります。ping の発生は、コミットには関係ありません。ユーザーはブロックを変更できますが、そのブロックが ping されるかどうかは、コミットをしたかどうかには関係ありません。

データの分割による ping の回避

インスタンス間でデータを分割して更新を実行する場合は、異なるインスタンスにそれぞれ 100 万個のブロックを置くことができます。この場合は、全て（100 万個）のブロックを 1 つの PCM ロックで受け持ったとしても、強制読込みまたは強制書込みはいっさい起きません。

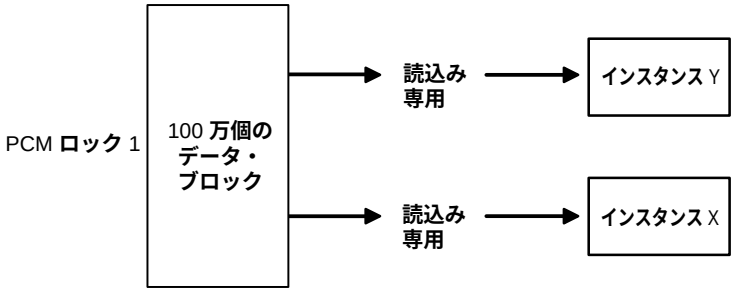
図 9-5 に示すように、表の中に 100 万個のデータ・ブロックがあり、それを 1 つの PCM ロックで受け持っています。この表のブロックは、インスタンス X の SGA から読込み、または（同 SGA へ）書込みを行います。もう 1 つの表にも 100 万個のデータ・ブロックがあり、1 つの PCM ロックで受け持っています。この表では、インスタンス Y の SGA から読込みまたは書込みを行います。この図のように分割すれば、更新が何回発生しても、インスタンス X とインスタンス Y 間でデータ・ブロックの強制読込みまたは強制書込みはいっさい起きません。

図 9-5 データの分割による ping の回避



読込み専用データの場合は、インスタンス X およびインスタンス Y の両方が共有モードで PCM ロックを保持できるので、ping は発生しません。これを図で示すと、図 9-6 のようになります。

図 9-6 ping が発生しない読込み専用データ



ロック・モードおよびバッファ状態

バッファ・キャッシュ内のブロックの状態は、そのブロックに設定されているロックのモードに直接関係しています。たとえば、あるインスタンスのバッファが排他現行モード（XCUR）状態の場合、そのインスタンスは排他モードで PCM ロックを所有します。複数のインスタンスのバッファ・キャッシュ内にある同じブロックについては XCUR バージョンは 1 つしかありませんが、SCUR バージョンは 2 つ以上存在することがあります。そのブロックの変更を行うには、プロセスは XCUR モードでブロックを取得する必要があります。

バッファの状態の確認

バッファの状態を知るには、V$BH 動的パフォーマンス表の STATUS 列をチェックします。この表は各バッファ・ヘッダーについての情報を提供します。

表 9-1 PCM ロック・モードおよびバッファ状態

PCM ロック・モード	バッファ状態名	説明
X	XCUR	インスタンスはこのバッファについて排他ロックを所有している。
S	SCUR	インスタンスはこのバッファについて共有ロックを所有している。
N	CR	インスタンスはこのバッファについて NULL ロックを所有している。

バッファ状態およびロック・モードの変化

図 9-7 に、インスタンスが特定のバッファでさまざまな操作を実行すると、バッファ状態およびロック・モードがどのように変化するかを示します。ロック・モードはカッコに入れて示してあります。

図 9-7 バッファの状態およびロック・モードの変化

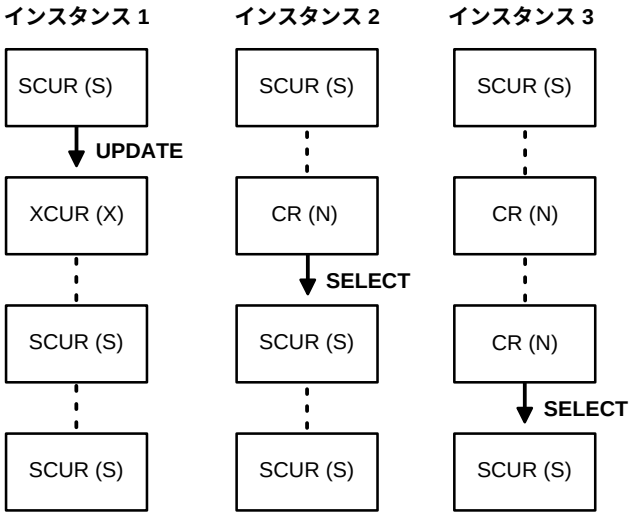


図 9-7 では、3 つのインスタンスがそれぞれ共有現行モード（SCUR）のブロックと共有ロック（S）で開始しています。インスタンス 1 がブロックに対する更新を行うときは、そのブロックに対するロック・モードは排他モード（X）に変化します。インスタンス 2 およびインスタンス 3 が所有していた共有ロックは、NULL モード（N）に変換されます。同時に、インスタンス 1 でのブロック状態は排他現行モード（XCUR）となり、インスタンス 2 およびインスタンス 3 でのブロック状態は一貫性読込みモード（CR）となります。これらのロック・モードは両立することができます。インスタンス 2 がブロックに対して SELECT 操作をするとき、およびインスタンス 3 がブロックに対して SELECT 操作をするときも、これと同様のロック・モードおよびブロック状態の変換が行われます。

ロック・モードの互換性と非互換性

一方のプロセスが特定のモードでロックを所有しているときに、他方のプロセスが特定のモードでロックを要求した場合、それが成功するか失敗するかは、表 9-2 に示すとおりです。

表 9-2 ロック・モードの互換性

要求されたロック / 所有されたロック	NULL	SS	SX	S	SSX	X
NULL	成功	成功	成功	成功	成功	成功
SS	成功	成功	成功	成功	成功	失敗
SX	成功	成功	成功	失敗	失敗	失敗
S	成功	成功	失敗	成功	失敗	失敗
SSX	成功	成功	失敗	失敗	失敗	失敗
X	成功	失敗	失敗	失敗	失敗	失敗

初期化パラメータによるブロックおよび PCM ロックの制御方法

この項では、初期化パラメータによってブロックおよび PCM ロックを制御する方法について説明します。

- GC_* 初期化パラメータ
- データ・ブロックの取扱い

GC_* 初期化パラメータ

表 9-3 に、PCM ロックを制御する初期化パラメータを示します。すべてのインスタンスですべてのパラメータを必ず設定してください。

表 9-3 PCM ロックを制御するパラメータ

パラメータ	説明	値
GC_FILES_TO_LOCKS	各データ・ファイル内で、各ブロックに対して固定ハッシュ・ロックと解放可能ハッシュ・ロックをマップします。 このパラメータの意味が変わりました。以前は、このパラメータで指定されていない ファイル（または、あとから追加されたファイル）には残りのハッシュ・ロックを割り当てていました。このパラメータで指定されていないファイルは、解放可能ファイン・グレイン・ロックを使用します。現在は、GC_FILES_TO_LOCKS の複数のエントリを使用できるようになりました。	GC_FILES_TO_LOCKS 用の構成文字列には、ロック数として 0 の値を指定できます。これは、ブロックがファイン・グレイン・ロックによりプロテクトされていることを示します。 この値は、すべてのインスタンスで同じにする必要があります。
GC_RELEASABLE_LOCKS	解放可能（ファイン・グレインもしくはハッシュもしくはロールバック）ロックのために使用するロックの数を設定します。	デフォルトは DB_BLOCK_BUFFERS の値です。通常はこの値を変更する必要はありません。 旧バージョンでは、このパラメータを DB_BLOCK_BUFFERS よりも小さい値に設定するのは無効でした。設定した場合は、自動的にデフォルト値に戻されました。Oracle8*i* では、低い値に設定できます。旧バージョンから移行している場合は、パフォーマンスに否定的な影響がないようにこのパラメータの設定を調べてください。
GC_ROLLBACK_LOCKS	各ロールバック・セグメントでロールバック・セグメント・ブロックの同時変更に利用できるインスタンス・ロック数を指定します。	デフォルトでは、各ロールバック・セグメントは、解放可能ロックを使用します。

関連項目：『Oracle8*i* リファレンス・マニュアル』（パラメータの詳細）および第 15 章「PCM インスタンス・ロックの割当て」（パラメータの設定方法）

データ・ブロックの取扱い

次に示すものだけを含むファイルには、PCM ロックを割り当てないでください。これらのファイルでは、クラス 1 のブロックは使用されません。

- 内部ソート用の一時表。これはクラス 2 のブロックです。
- ロールバック・セグメント。これは GC_ROLLBACK_LOCKS によって保護されます。

PCM ロックの 2 つの方法 : 固定および解放可能なロック

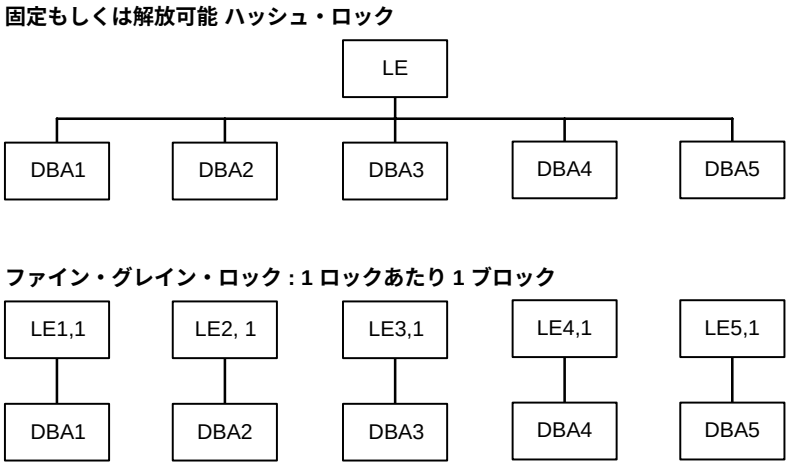
この項では、PCM ロックの 2 つの方法、つまり固定（ハッシュ）ロックと解放可能ロックを比較します。どちらか一方または両方の PCM ロックを使用して、データ・ファイルのブロックをプロテクトできます。

- IDLM ロック要素および PCM ロック
- PCM ロックあたりのブロック数
- ファイン・グレイン・ロックの動き
- ロック要素の動き
- 解放可能ロックのパフォーマンスへの影響
- 異なるファイルへのファイン・グレイン・ロックおよびハッシュ・ロックの適用

IDLM ロック要素および PCM ロック

図 9-8 に、ハッシュ・ロックおよびファイン・グレイン・ロックでのロック要素とブロックの対応を示します。ロック要素（LE）は、IDLM ロックを表す Oracle 固有のデータ構造体です。IDLM では、ロック要素と PCM ロックとの間に一対一の対応関係があります。

図 9-8 ハッシュ・ロックおよびファイン・グレイン・ロック



固定ハッシュ PCM ロックの場合のロック要素

固定ハッシュ PCM ロックおよび解放可能ハッシュ・ロックのどちらの場合も、1 つのロック要素に複数のブロックを指定できます。ただし、固定ハッシュ PCM ロックは解放可能でないため、ロック要素名が " 固定 " しているという違いがあります。

ロック要素が ping されると、そのロック要素が所有している他の変更済みブロックも、必要なブロックといっしょに書き込まれます。たとえば、図 9-8 で、ブロック DBA2 が必要になり、LE が ping されると、ブロック DBA1、DBA3、DBA4 および DBA5 もすべてディスクに書き込まれます（ブロックが変更されている場合）。

ファイン・グレイン PCM ロックの場合のロック要素

ファイン・グレイン・ロックでは、ロック要素の名前は IDLM の内部でのリソースの名前です。

決まった数のロック要素が潜在的に数百万のブロックを受け持ちますが、ロック要素名は、要求された特定のブロックに関連付けられるたびに、何度も変更されます。ロック要素名（たとえば LE7,1）は、その要素が受け持つブロックのデータベース・ブロック・アドレス（7）およびクラス（1）を示します。ロック要素を再使用するには、IDLM ロックを解放する必要があります。その上で、そのロック要素を改名および再使用して、必要があれば IDLM 内に新しいリソースを作成することができます。

ファイン・グレイン・ロックを使用すると、さらに多くのロックを扱うシステムを設定できます。これは、それらの名前を同時に保持する必要がないからです。

DBA ファイン・グレイン PCM ロックも、解放可能ロックです。DBA ファイン・グレイン・ロックでは、ロック要素とブロックの間に 1 対 1 の関係が設定されます。図 9-8 で示してい

るこのような関係を "DBA ロック " とも言います。この設定では、LE2,1 が ping された場合に、ブロック DBA2 だけがディスクに書き込まれます。

PCM ロックあたりのブロック数

この項では、ハッシュ・ロックおよびファイン・グレイン・ロックで、ロックの細分性がどのように違うかについて説明します。

複数ブロックを受け持つ固定ハッシュ・ロック

固定ハッシュ PCM ロックは、複数の Oracle データベース・ブロックをプロテクトできます。データベース内での PCM ロックとブロックとのマッピングは、最初の OPS インスタンスの起動時に指定した初期化パラメータを使用して、ファイル別に決定されます。初期化パラメータには、PCM ロックがファイル内の一定範囲内の連続したブロックをプロテクトするように指定できます。

ハッシュ・ロックは、次のような状況で使用すると役立ちます。

表 9-4 ハッシュ PCM ロックの使用が適している状況

状況	理由
データのほとんどが読込み専用であるとき	少数のハッシュ・ロックが多数のブロックを受け持つことができるので、頻繁なロック・オペレーションを必要としません。ハッシュ・ロックが解放されるのは、他のインスタンスがデータの変更を必要とするときだけです。パラレル問合せオプションを使用した場合の読込み専用データの処理では、ハッシュ・ロックの方がファイン・グレイン・ロックの場合より最高 100% まで高速化できます。 データのすべてが読込み専用である場合は、表領域自体を読込み専用として指定することを検討してください。これにより、表領域では PCM ロックがまったく不要になります。
データを変更する可能性の高いインスタンスに従ってデータを分割できるとき	このような分割に適合するように定義されたハッシュ・ロックを使用すると、インスタンスが分割された IDLM ロックの集合を保持できるので、必要な IDLM 操作が減少します。
比較的少数のインスタンスの集合が大量のデータを変更するとき	ハッシュ・ロックでは、要求元のインスタンスがすでにロックを保持している場合に、IDLM アクティビティがなくても新しいデータベース・ブロックへのアクセスを続けることができます。

ハッシュ・ロックを使用すると、異なるデータベース・ブロックを変更するインスタンス間に衝突が起こり、余分なクロス・インスタンス・ロック・アクティビティが生じることがあります。このような偽の衝突（"false ping"）を解決するには、現在ロックを所有しているイ

ンスタンスのキャッシュから、ブロックをいくつか書き込むことが必要になる場合があります。

解放可能ロック : 1 つ以上のブロックの場合のロック

解放可能ロック（ハッシュまたはファイン・グレイン）は、1 つ以上の Oracle データベース・ブロックをプロテクトできます。PCM ロックとデータブロックとの間に 1 対 1 の関係を設定した場合、競合が起こるのは、複数のインスタンスが同じブロックからのデータを必要とする場合だけです。このレベルの解放可能ファイン・グレイン・ロックを DBA ロックと呼びます。（*DBA* は、単一ブロックのデータのデータ・ブロック・アドレスです。）1 ロックにつき複数のブロックを割り当てた場合は、解放可能ハッシュ・ロックと呼び、固定ハッシュ・ロックの場合と同様に競合が起こります。

ほとんどのシステムでは、1 つのインスタンスがデータベースの各ブロックについてロックを設定することはできません。これは、SGA メモリーまたは IDLM ロックの能力を超過してしまうからです。したがって、インスタンスは、必要なときにファイン・グレイン・ロックを取得したり解放したりします。解放可能ロック、ロック要素およびリソースは、IDLM 内で改名され再使用されるので、システムに設定する数は少なくて済みます。DB_BLOCK_BUFFERS の値は、割り当てる解放可能ロック数としてお薦めする最小の数量です。

DBA ロックは、複数のインスタンスが同じデータベース・オブジェクトを頻繁に更新するような場合に便利です。これには次のような利点があります。

- 2 つのインスタンスが同じブロックを必要とする場合以外は、衝突が起こらない。
- 排他モードで PCM ロックを所有しているインスタンスが、必要なブロックだけをディスクに書き込む。

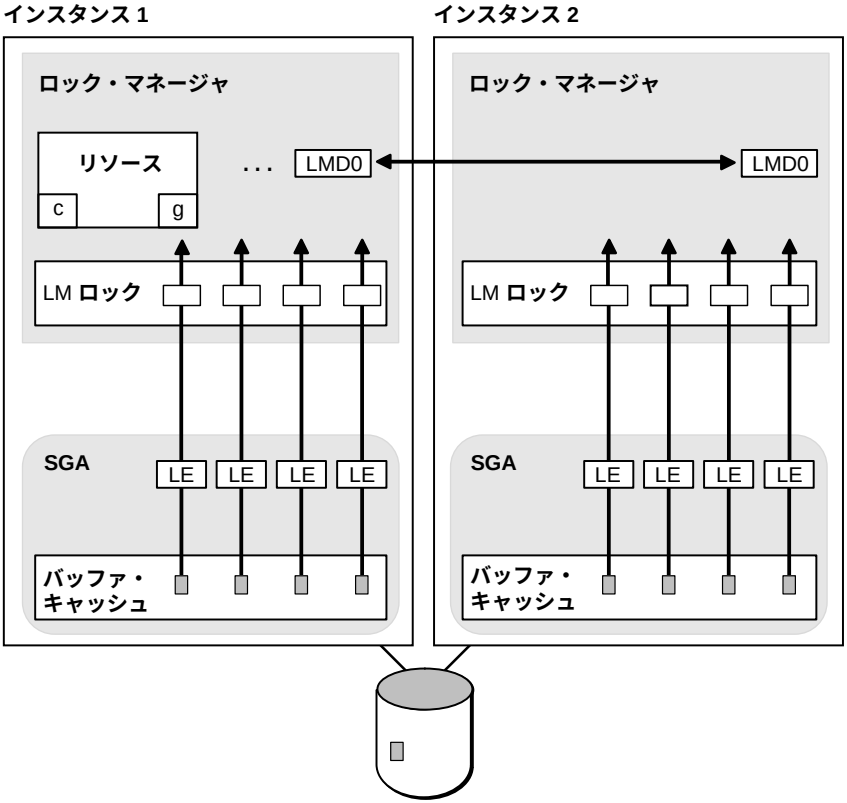
DBA ロックの欠点は、各ブロックの読込みのたびにオーバーヘッドが生じ、そのためにパフォーマンスが低下するということです。（そのつど新しいロックの獲得と解放を繰り返すと、固定ハッシュ・ロックの場合のように既存のロックを NULL モードから排他モードに（またはその逆に）変換するのに比べて、IDLM を介したオーバーヘッドが増加します。）

関連項目： 15-8 ページの「解放可能ロックの例」

ファイン・グレイン・ロックの動き

図 9-9 に、ファイン・グレイン・ロック（DBA ロック）が機能するしくみを示します。

図 9-9 ロック要素によるブロックの調整（ファイン・グレイン・ロック）



フォアグラウンド・プロセスは SGA をチェックして、インスタンスがブロックに対するロックを所有しているかどうかを調べます。

- 所有されていないロックが存在しない場合は、フォアグラウンド・プロセスが他のロックを解放してロックを 1 つ作成します。
- インスタンスがロックを所有していて、そのモードが正しくない場合は、フォアグラウンド・プロセスはそのロックを変換します（たとえば共有モードから排他モードへ変換する）。

ロック要素の動き

ロック要素は、固定または解放可能の 2 つのモードのいずれかで作成されます。

- 固定ロック（1 つ以上のブロックを受け持つハッシュ・ロック）は、固定モードでロック要素を作成します（このモードは常に有効です）。これは静的モードです。つまり、いったんこのモードを割り当てたロック要素は同じ状態を維持します。
- 解放可能ロック（ファイン・グレイン・ロック（DBA ロック）またはハッシュ・ロック）は、非固定モードでロック要素を作成します。これらのロック要素名は変化することがあり、また、対象となるブロックも変化することがあります。非固定モードのロック要素は、有効、古いまたは空きのいずれかになります。有効ビットが設定されている場合は、IDLM 内のリソースに対するロックが所有されます。これが設定されていない場合は、ロックはありません。空きの場合は、ロックは存在しますが、バッファはすでにそのロック要素からリンク解除されているので、そのロック要素は、空きロック要素の最低使用頻度（LRU）リストに入っています。

注意： 有効なロック要素は、IDLM 内にロックを持っています。無効なロック要素はロックを持っていません。空きロック要素は、IDLM の中に、現在このバッファにリンクされていないロックがあることを示します。このロックは、LRU リスト上で待機しています。ロック要素が古い場合は、古い名前についての有効ロック・ハンドルが存在します。Oracle がこのロック要素を使用できるようにするには、新しい名前を与える必要があります。

V$LOCK_ELEMENT ビューは、ロック要素のステータスを示します。

解放可能ロックのパフォーマンスへの影響

解放可能ロックは、OPS のパフォーマンスに影響を与えることがあります。解放可能ロックは比較的不経済なので（バッファ取得時のロック解放およびロック取得を伴うため）、このモードで実行しているときは、一部の操作でパフォーマンスが低下することがあります。しかし、データベースへのその他のタイプの操作では、解放可能なファイン・グレイン・ロック（DBA ロック）を使用することでパフォーマンスが向上します。ファイン・グレイン・ロックに伴う結果には次のものがあります。

- 表の読込み専用走査では、余分なロック・オペレーションが必要になることがあります。この様な場合は、ハッシュ・ロックを使用することができます。
- 競合度の高いブロックまたはオブジェクトについては、ハッシュ・ロックを使用時に発生していた偽の衝突が減少する場合があります。この場合は、ファイン・グレイン・ロックを選択するほうがよいでしょう。
- 同時処理の頻度の低いデータ・ブロックの場合は、システムのロック・オペレーションは不経済になり、偽の衝突率は低下します。この場合は、何を優先するかを調べた上で、アプリケーションにとってどのロックが有利かどうかを判断します。

- 解放可能ハッシュ・ロックによるブロックのグループ化は、表の走査に効果的な場合があります。

異なるファイルへのファイン・グレイン・ロックおよびハッシュ・ロックの適用

各データ・ファイルについて、2 つのロック方法のどちらかを使用できます。最善の結果を得るには、一部のデータ・ファイルについてはハッシュ・ロックを使用し、その他のデータ・ファイルについてはファイン・グレイン・ロック（DBA ロック）を使用することが必要な場合があります。

ハッシュ・ロックおよびファイン・グレイン・ロックを、ファイル単位で選択して適用できます。たとえば、一群のファイルに対して次のようにロックを適用できます。

GC_FILES_TO_LOCKS = "1=100:2=0:3=1000:4-5=0EACH"
GC_RELEASABLE_LOCKS=10000

表 9-5 ハッシュ・ロックおよびファイン・グレイン・ロックの選択的適用

ファイル番号	ロック・モード	GC_FILES_TO_LOCKS の値
1	ハッシュ	100
2	ファイン・グレイン	0
3	ハッシュ	1000
4	ファイン・グレイン	0
5	ファイン・グレイン	0

Oracle によるブロックへのロックの割当て

この項では、固定ハッシュ・ロックおよび解放可能ハッシュ・ロックがどのようにブロックに割り当てられるかについて説明します。（もちろん、DBA ロックはブロックに対して 1 対 1 の関係を持ちます。）

- ファイル対ロックのマッピング
- ブロック・クラスあたりのロック数
- ロック要素番号

ファイル対ロックのマッピング

ファイル対ロックのマッピングは、SGA の中の 2 つのデータ構造体が制御します。第 1 の構造体は、各ファイル（DB_FILES）を第 2 の構造体の中のバケット（索引）にマップします。この構造体には、このバケットに割り当てられているロックの数、基本ロック番号、ブロックのグループ化因数についての情報が入っています。表領域のロック数を検出するには、表

領域の各ファイルをプロテクトしている実固定ロックの合計件数を割り出す必要があります。複数のファイルがロックを共有している場合は、ロックを 1 回だけ数えます。

1. 表領域のロック数を検出するには、最初に FILE_LOCK データ・ディクショナリ表からの選択を実行します。

   ```
   SELECT * FROM FILE_LOCK ORDER BY FILE_ID;
   ```

   たとえば、GC_FILES_TO_LOCKS="1=500:5=200" のように設定すると、次のような値が返されます。

   ```
   FILE_ID FILE_NAME       TS_NAME          START_LK     NLOCKS   BLOCKING
   ------- --------------- -------------- ---------- ---------- ----------
         1 ¥¥.¥OPS_SYS01   SYSTEM                100       1500          1
         2 ¥¥.¥OPS_USR01   USER_DATA            1600       3000          1
         3 ¥¥.¥OPS_RBS01   ROLLBACK_DATA           0        100          1
         4 ¥¥.¥OPS_TMP01   TEMPORARY_DATA          0        100          1
         5 ¥¥.¥OPS_USR03   TRAVEL_DEMO          4600       4000          1
         6 ¥¥.¥PROBLEM_REP PROBLEM_REP             0        100          1

   6 rows selected.
   ```

2. ロック数（NLOCKS 列の値）を合計することで、表領域内のロックの件数を算出します。ただし、START_LCK 列の値が異なる行だけを合計します。

   次の例では、ファイル 1 およびファイル 5 に異なる START_LCK 値が設定されています。したがって、これらのファイルの NLOCKS 値を合計すると、700 個のロックを得ることができます。

   GC_FILES_TO_LOCKS="1-2=500:5=200" のように設定すると、次のような結果になります。

   ```
   FILE_ID  FILE_NAME  TABLESPACE_NAME  START_LK  NLOCKS BLOCKING
   1        file1      system             1         500    1
   1        file2      system             1         500    1
   1        file3      system             0
   1        file4      system             0
   1        file5      system           501         200    1
   ```

   今度は、ファイル 1 およびファイル 2 の START_LCK 値が同じです。つまり、これらのファイルは、問題のロックを共有しています。ファイル 5 には、異なる START_LCK 値が設定されています。したがって、ファイル 1 およびファイル 2 が共有する 500 個のロックを 1 件として数えて、その値にファイル 5 の 200 個のロックを加算すると、合計で 700 個のロックを得ることができます。

ブロック・クラスあたりのロック数

考慮する必要があるのは、データのブロック数および取消し（ロールバック）ブロック・クラスだけです。データ・ブロック（クラス 1）には、索引または表からのデータが入ります。システム取消し（ロールバック）ヘッダー・ブロック（クラス 10）は、ロールバック・セグメント・ヘッダーまたはトランザクション・テーブルとも呼ばれます。システム取消しブロック（クラス 11）はロールバック・セグメントの一部で、取消し（ロールバック）レコード用の記憶領域として使用されます。

ユーザー取消し（ロールバック）セグメント *n* ヘッダーのブロック数は、クラス 10 + (*n**2) として識別され、*n* はロールバック・セグメント番号を表します。値が *n* = 0 の場合は、システム・ロールバック・セグメントであることを示し、値が *n* > 0 の場合は、非システム・ロールバック・セグメントであることを示します。同様に、ユーザー取消しセグメント *n* ヘッダーのブロック数は、クラス 10 + ((n*2) + 1) として識別されます。

次の問合せは、1 クラスに割り当てられているロックの数を表示します。

```
SELECT CLASS, COUNT(*)
  FROM V$LOCK_ELEMENT
  GROUP BY CLASS
  ORDER BY CLASS;
```

次の問合せは、固定（非解放可能）PCM ロックの数を表示します。

```
SELECT COUNT(*)
  FROM V$LOCK_ELEMENT
  WHERE bitand(flag, 4)!=0;
```

次の問合せは、解放可能 PCM ロックの数を表示します。

```
SELECT COUNT(*)
  FROM V$LOCK_ELEMENT
  WHERE bitand(flag, 4)=0;
```

ロック要素番号

データ・クラスのブロックの場合は、データ・ブロック・アドレス（DBA）からファイル番号が判別されます。バケットは、X$KCLFI 動的パフォーマンス表から見つけ出されます。データ・クラスのブロックは、次のようにロック要素番号にハッシュされます。

$$\left(\frac{\text{DBA}}{\text{グループ化因数}}\right) \text{ モジュロ (ロック数) + (開始番号)}$$

その他のブロック・クラスは、次のようにロック要素番号にハッシュされます。

(*DBA*) モジュロ (クラスのロック数)

例 : PCM ロックへのブロックのマッピング

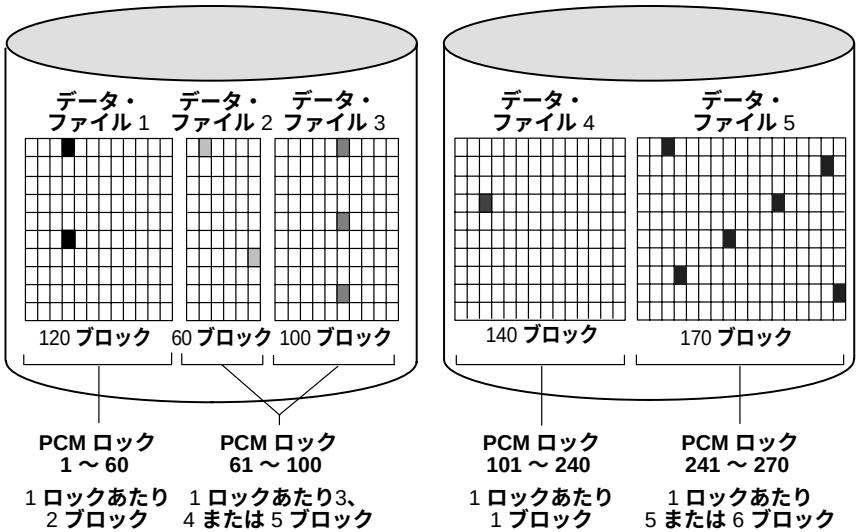
- GC_FILES_TO_LOCKS のハッシュ・ロック設定
- GC_FILES_TO_LOCKS のハッシュ・ロック設定のその他の例
- GC_FILES_TO_LOCKS の DBA ロック設定の例

GC_FILES_TO_LOCKS のハッシュ・ロック設定

次の例は、ブロックをハッシュ PCM ロックにマッピングするためのいくつかの方法、および複数のデータ・ファイルに対して同じロックがどのように使用されるかを示しています。

注意 : これらの例では、重要な概念を示すためにきわめて小さいサンプル・ファイルを使用しています。ユーザーが実際に管理するファイルは、これよりはるかに大きくなります。

図 9-10 データ・ブロックへの PCM ロックのマッピング



例 1 図 9-10 に、パラメータ値が GC_FILES_TO_LOCKS = "1=60:2-3=40:4=140:5=30" である場合の、ブロックと PCM ロックの対応例を示します。

図 9-10 のデータ・ファイル 1 では、60 個の PCM ロックが、60 の倍数である 120 個のブロックにマップされます。したがって、各 PCM ロックはそれぞれ 2 つのデータ・ブロックを受け持ちます。

データ・ファイル 2 および 3 では、40 個の PCM ロックが合計 160 個のブロックにマップされます。1 つの PCM ロックが受け持つことができるのは、データ・ファイル 2 では 1 個または 2 個のデータ・ブロック、そしてデータ・ファイル 3 では 2 個または 3 個のデータ・ブロックです。したがって、両方のデータ・ファイルを合算した場合、1 つの PCM ロックが受け持つデータ・ブロックの数は、3 個、4 個、5 個のいずれかです。

データ・ファイル 4 では、PCM ロックとデータ・ブロックの数が同じなので、各 PCM ロックはそれぞれ 1 つのデータ・ブロックにマップされます。

データ・ファイル 5 では、30 個の PCM ロックが 170 個のブロックにマップされますが、これは 30 の倍数ではありません。したがって、各 PCM ロックは、5 個または 6 個のデータ・ブロックを受け持ちます。

図 9-10 に示した PCM ロックはすべて、読込みロック・モードまたは読込み排他モードのどちらでも保持できます。

例 2 次のパラメータ値は、500 個の PCM ロックをデータ・ファイル 1 に、それぞれ 400 個の PCM ロックをファイル 2、3、4、10、11、12 に、150 個の PCM ロックをファイル 5 に、250 個の PCM ロックをファイル 6 に、300 個の PCM ロックを集合的にファイル 7 ～ 9 に割り当てます。

```
GC_FILES_TO_LOCKS = "1=500:2-4,10-12=400EACH:5=150:6=250:7-9=300"
```

この例では、合計 (500 + (6*400) + 150 + 250 + 300) = 3600 個の PCM ロックを割り当てます。後でさらにデータ・ファイルを追加することを予定している場合は、この PCM ロック数を超える値を指定できます。

例 3 例 2 では、"7-9=300" という句で、300 個の PCM ロックをデータ・ファイル 7、8、9 に集合的に割り当てました。キーワード EACH は省略されています。これらのデータ・ファイルにはそれぞれ 900 個ずつのデータ・ブロックがあり、合計データ・ブロック数が 2700 個であるとすれば、各 PCM ロックがそれぞれ 9 個のデータ・ブロックを受け持つことになります。データ・ファイルの総数が 300 の倍数なので、各 PCM ロックが受け持つ 9 個のデータ・ブロックは 3 つのデータ・ファイルに分散し、1 つの PCM ロックが各データ・ファイル内の 3 個ずつのデータ・ブロックを受け持つことになります。

例 4 次のパラメータ値は、200 個の PCM ロックをそれぞれデータ・ファイル 1 ～ 3 に、50 個の PCM ロックをデータ・ファイル 4 に、100 個の PCM ロックを集合的にデータ・ファイル 5、6、7、9 に、そして、連続した 50 ブロックのグループ内の 20 個のデータ・ロックを、データ・ファイル 8 および 10 の組合せに割り当てます。

```
GC_FILES_TO_LOCKS = "1-3=200EACH 4=50:5-7,9=100:8,10=20!50"
```

この例では、個々のデータ・ファイルのデータ・ブロック数が 100 より少ない場合を除き、データ・ファイル 5、6、7、9 の組合せに割り当てられた各 PCM ロックが、それぞれ各デー

タ・ファイル内の 1 つ以上のデータ・ブロックを受け持ちます。データ・ファイル 5 ～ 7 にはそれぞれ 500 個ずつのデータ・ブロックがあり、データ・ファイル 9 には 100 個のデータ・ブロックがあるとすると、各 PCM ロックはそれぞれ 16 個のデータ・ブロックを受け持ちます。つまり、データ・ファイル 9 の中の 1 個のブロックと、その他のデータ・ファイルの中の 5 個ずつのブロックを受け持ちます。これに対して、データ・ファイル 9 にデータ・ブロックが 50 個しかないとすると、PCM ロックの半分はそれぞれ 16 個のデータ・ブロック（そのうち 1 つはデータファイル 9 のブロック）を受け持ちますが、残りの半分はそれぞれ 15 個ずつしか受け持たないことになります（データ・ファイル 9 のブロックは対象となりません）。

データ・ファイル 8 および 10 に集合的に割り当てられた 20 個の PCM ロックは、50 個のデータ・ブロックからなる連続したグループを受け持ちます。これらのデータ・ファイルに含まれるデータ・ブロックの数が 50 の倍数で、合計ブロック数が 50 の 20 倍（つまり 1000）以下である場合は、各 PCM ロックは、それぞれ、データ・ファイル 8 またはデータ・ファイル 10 の両方ではなく、どちらか一方だけのデータ・ブロックを受け持ちます。これは、各 PCM ロックが、それぞれ連続した 50 個のデータ・ブロックを受け持つことになるからです。データ・ファイル 8 のデータ・ブロック数が 50 の倍数ではない場合は、1 つの PCM ロックが両方のファイル内のデータ・ブロックを受け持つ必要があります。データ・ファイル 8 および 10 のデータ・ブロック数が 1000 を超えている場合は、一部の PCM ロックが、50 データ・ブロックからなる複数のグループを受け持つことが必要になります。その場合、各グループがそれぞれ異なるファイルに入っていることもあります。

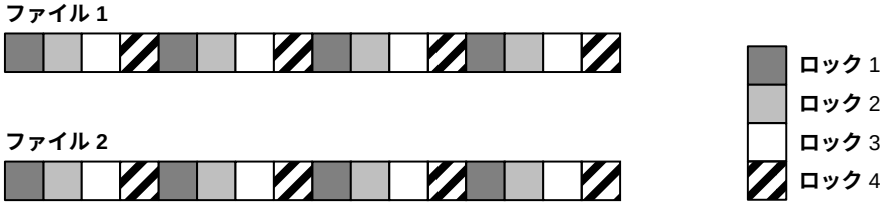
GC_FILES_TO_LOCKS のハッシュ・ロック設定のその他の例

例 5、例 6 および例 7 は、GC_FILES_TO_LOCKS に各種の値を指定したときの結果を示しています。これらの例では、ファイル 1 および 2 にはそれぞれデータ・ブロックが 16 個ずつあります。

例 5 GC_FILES_TO_LOCKS="1-2=4"

この例では、ファイル 1 および 2 について 4 つのロックを指定しています。したがって、各ロックは、それぞれ 8 個（(16+16) / 4）のブロックを受け持ちます。ブロックは連続していません。

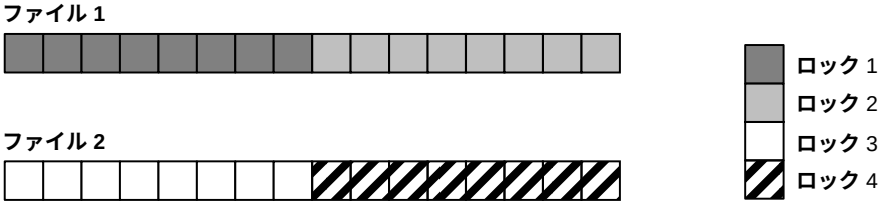
図 9-11　GC_FILES_TO_LOCKS 例 5



例 6 GC_FILES_TO_LOCKS="1-2=4!8"

この例では、ファイル 1 および 2 について 4 つのロックを指定しています。しかし、これらのロックは 8 個の連続したブロックを受け持つ必要があります。

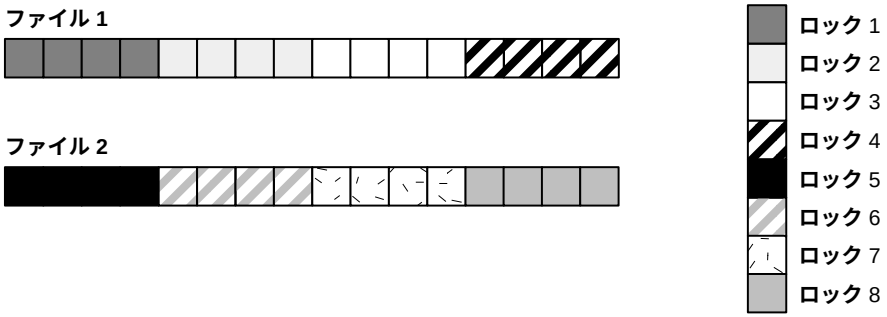
図 9-12　GC_FILES_TO_LOCKS 例 6



例 7 GC_FILES_TO_LOCKS="1-2=4!4EACH"

この例では、ファイル 1 および 2 について 4 つのロックを指定しています。これらのロックは 4 個の連続したブロックを受け持つ必要があります。

図 9-13　GC_FILES_TO_LOCKS 例 7



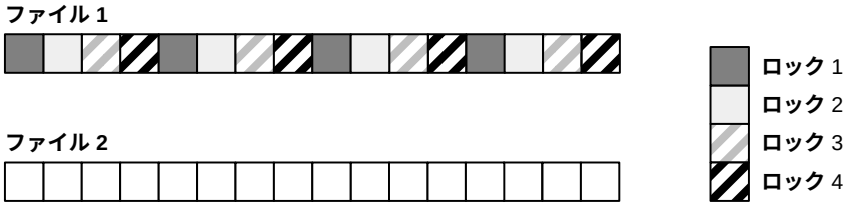
GC_FILES_TO_LOCKS の DBA ロック設定の例

次の例は、DBA ロックとハッシュ・ロックの組合せを示しています。

例 8 GC_FILES_TO_LOCKS="1=4:2=0"

ファイル 1 には、4 つのロックからなるハッシュ PCM ロックがあります。ファイル 2 では、要求に応じて DBA ロックを割り当てます。初期の状態では、ロックは 1 つも割り当てられていません。

図 9-14　GC_FILES_TO_LOCKS 例 8



10

非 PCM インスタンス・ロック

この章では、最も一般的な非 PCM インスタンス・ロックについて説明します。次の項目について説明します。

- 概要
- トランザクション・ロック（TX）
- 表ロック（TM）
- システム変更番号（SC）
- ライブラリ・キャッシュ・ロック（N[A-Z]）
- ディクショナリ・キャッシュ・ロック（Q[A-Z]）
- データベース・マウント・ロック（DM）

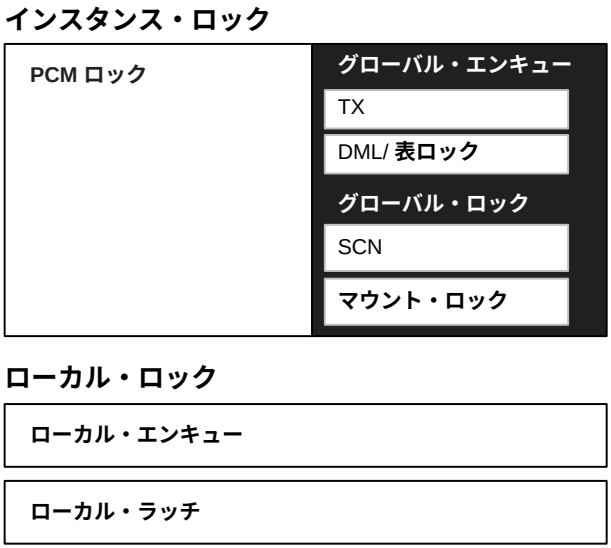
関連項目： 第 16 章「リソースおよびロック用の IDLM 容量の確保」（統合分散ロック・マネージャ（IDLM）で構成する非 PCM リソースおよびロック数の計算方法について）

概要

この章では、Oracle 環境内でのトランザクション、表およびその他のエンティティのロックを管理するために、Oracle で非 PCM ロックがどのように使用されているかを説明します。トランザクション・ロックの "TX" および表ロックの "TM" などの接頭辞は、各ロック・タイプを識別するために Oracle で使用するネーミング・スキームを参照します。

図 10-1 では、非 PCM ロックと Oracle で使用する他のロックとの関連を示しています。

図 10-1　Oracle ロック・メカニズム : 非 PCM ロック



PCM ロックは静的なもの（アプリケーションの設計時に割り当てるもの）ですが、非 PCM ロックはきわめて動的なものです。非 PCM ロックの数およびそれに対応する領域要件は、システムの初期化パラメータの値の変化に伴って異なります。

関連項目：『Oracle8*i* リファレンス・マニュアル』（すべての非 PCM ロックの説明）

トランザクション・ロック（TX）

行ロックとは、選択した行を保護するロックです。次のいずれかの文によって行が変更されると、トランザクションは個々の行ごとにグローバル・エンキューおよび排他ロックを取得します。

- INSERT
- UPDATE
- DELETE
- FOR UPDATE 句付きの SELECT

これらのロックは、ブロックに保管され、グローバル・トランザクション・エンキューを参照します。

排他モードのトランザクション・ロックが獲得されるのは、トランザクションが最初の変更を開始する時点です。このロックは、トランザクションがコミットまたはロールバックされるまで設定されます。また、SMON がトランザクションの回復（取消し）を行うときも、排他モードのトランザクション・ロックを取得します。トランザクション・ロックは、実行中のトランザクションによりロックされたオブジェクトの解放を待つプロセスのためのキュー・メカニズムとして働きます。

表ロック（TM）

表ロックは、表全体を保護する DML ロックです。トランザクションは、INSERT、UPDATE、DELETE、FOR UPDATE 句を伴う SELECT、LOCK TABLE のいずれかの文で変更される個々の表について、それぞれ表ロックを取得します。表ロックのモードとしては、NULL（N）、行共有（RS）、行排他（RX）、共有ロック（S）、共有行排他（SRX）および排他（X）のどれでも使用できます。

あるインスタンスがデータベースをマウントしようとすると、表ロックを使用して、関与するすべてのインスタンスが、DML_LOCKS = 0 または DML_LOCKS != 0 のどちらかに設定されます。この条件が満たされていない場合は、エラー ORA-61 が戻され、マウント試行は失敗します。実行中のトランザクションで DML 文で表を参照すると、表ロックが獲得されます。これは、トランザクションの実行中にオブジェクトが削除されたり変更されたりしないようにするためです。ロックが獲得されるのは、DML_LOCKS パラメータが 0 以外である場合だけです。

次の文を使用すると、特定の表について、選択的に表ロックをオンまたはオフにすることもできます。

```
ALTER TABLE tablename DISABLE|ENABLE TABLE LOCK
```

DML_LOCKS をゼロに設定すると、DDL 操作はいっさいできません。これは、表ロックが使用禁止に設定された表についても同じです。

関連項目： 16-6 ページの「表ロック数の最小化によるパフォーマンスの最適化」（パフォーマンス向上のための表ロックの使用禁止について）

システム変更番号（SC）

システム変更番号（SCN）は、1 つのインスタンス内または全インスタンスを通じて、イベントを順序付けるために Oracle が使用する論理タイムスタンプです。Oracle が SCN の生成に使用するスキームの 1 つに、ロック・スキームがあります。

SCN ロック・スキームでは、グローバル SCN を SCN ロック値ブロック内に保持します。グローバル SCN 値は、多くのデータベース・イベント（特に COMMIT WORK）に応じて増加します。グローバル SCN 増加プロセスは、排他モードで SCN ロックを取得し、SCN を増加させ、ロック値ブロックを書き込み、そしてロックをダウングレードします。SCN ロック値へのアクセスはバッチ化されています。Oracle はグローバル SCN のキャッシュ・コピーをメモリー内に保持します。他のプロセスによってフェッチされた SCN を読み込むことで、いっさいの通信オーバーヘッドを生じさせずに SCN を取得できます。

SCN のインプリメント方式はプラットフォームごとに異なります。MAX_COMMIT_PROPAGATION_DELAY 初期化パラメータの方がプラットフォーム固有のしきい値（通常は 7）よりも小さいとき、通常 Oracle は SCN ロック・スキームを使用します。反対に、MAX_COMMIT_PROPAGATION_DELAY の方がしきい値よりも大きいときは、Oracle は Lamport SCN スキームを使用します。インスタンスの起動後にアラート・ログを調べれば、採用された SCN 生成スキームがわかります。

関連項目： システム固有の Oracle マニュアル（SCN のインプリメントの詳細）

ライブラリ・キャッシュ・ロック（N[A-Z]）

SQL (DML / DDL) または PL / SQL 文の解析またはコンパイル中に、データベース・オブジェクト（表、ビュー、プロシージャ、関数、パッケージ、パッケージ本体、トリガー、索引、クラスタ、シノニム）が参照されると、その文を解析またはコンパイルしているプロセスは、正しいモードのライブラリ・キャッシュ・ロックを取得します。Oracle8 では、ロックが設定されるのは、解析またはコンパイルが完了するまで（つまり解析コールの存続期間中だけ）です。

ディクショナリ・キャッシュ・ロック（Q[A-Z]）

データ・ディクショナリ・キャッシュには、データ・ディクショナリ（メタデータ記憶領域）からの情報が入っています。このキャッシュを利用することで、データ・ディクショナリに効率的にアクセスできます。

たとえば新しい表を作成すると、その表のメタデータがデータ・ディクショナリにキャッシュされます。表を削除する場合は、データ・ディクショナリ・キャッシュからメタデータを削除する必要があります。データ・ディクショナリ・キャッシュへのアクセスを同期化するために、ラッチが排他モードおよび単一共有モードで使用されます。インスタンス・ロックは、複数共有（パラレル）モードで使用されます。

Oracle Parallel Server（OPS）では、1 つのインスタンスで削除された表のメタデータが、すべてのノード上のデータ・ディクショナリ・キャッシュに含まれている場合があります。この表のメタデータは、各インスタンスのデータ・ディクショナリ・キャッシュからフラッシュする必要があります。これは、インスタンス・ロックが実行し同期化します。

データベース・マウント・ロック（DM）

マウント・ロックは、インスタンスのどれかが特定のデータベースをマウントしたかどうかを示します。このロックは、OPS 構成でのみ使用できます。これは、OPS が排他モードで使用する唯一の複数インスタンス・ロックで、他のインスタンスが共有モードでデータベースをマウントするのを防止するためのものです。

OPS の単一共有モードでは、このロックは共有モードで設定されます。他のインスタンスも、共有モードで同じデータベースをマウントできます。ただし、OPS 排他モードでは、他のインスタンスはロックを取得できません。

11

領域管理および空きリスト・グループ

Thus would I double my life's fading space;
For he that runs it well, runs twice his race.

Abraham Cowley, *Discourse xi, Of Myself*

この章では、領域管理の概念について説明します。

- Oracle での空き領域の取扱い方法
- 空き領域の管理のための SQL オプション
- 複数インスタンスでの空き領域の管理
- インスタンス、ユーザーおよびロックに関連付けられた空きリスト
- エクステントの割当ての制御

関連項目： 第 17 章「空きリスト・グループによるデータ分割」（領域管理の手順について）

Oracle での空き領域の取扱い方法

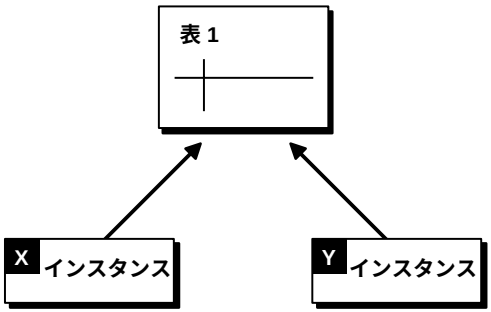
この項では、Oracle が空き領域をどのように取り扱うかについて示します。この項では、次の項目について説明します。

- 概要
- データベース記憶構造
- 空き領域管理のための構造体
- 例 : 空きリスト・グループ

概要

Oracle Parallel Server（OPS）では、別々のインスタンスで実行中の複数のトランザクションが、新規レコードのための空き領域の確保で競合せずに、同じ表にデータを同時に挿入および更新ができます。

図 11-1 同じ表に同時に挿入するインスタンス



この機能を利用するには、この章で定義するいくつかの構造体を使用して、データベース内の空き領域を積極的に管理することが必要です。

表、クラスタまたは索引などの各データベース・オブジェクトでは、挿入または更新によって行が元のブロックの使用可能領域を超える可能性があるので、Oracle は挿入または更新のために使用できる領域からなるブロックを管理します。空き領域を必要とするユーザー・プロセスは、空き領域のあるブロックのマスター空きリストから、領域を見つけることができます。マスター空きリストの中に、そのユーザー・プロセスを収容できる領域を備えたブロックがない場合は、Oracle は新しいエクステントを割り当てます。

新しいエクステントが自動的に表に割り当てられると、そのエクステントのブロックがマスター空きリストに追加されます。自動的に割り当てられたエクステントに含まれている空き領域は、どのグループの空きリストにも再割当てできないので、パラレル・サーバー上の複数のインスタンス間で、空き領域を獲得するための競合が生じることがあります。エクステントをインスタンスに明確に割り当てれば、ユーザーによる空き領域の管理がさらに強化できます。これによって、空き領域についての競合を最小限に抑えることができます。

データベース記憶構造

この項では、データベース記憶域の基本構造について説明します。

- セグメントおよびエクステント
- 高水位標

セグメントおよびエクステント

セグメントは、論理データベース記憶域の単位です。Oracle は、エクステントと呼ばれるさらに小さい単位で、セグメント用の領域を割り当てます。エクステントは、特定のタイプの情報を格納するために割り当てられた、特定数の連続したデータ・ブロックです。

したがって、セグメントは、特定タイプのデータ構造体用に割り当てられたエクステントの集合で構成されています。たとえば、個々の表のデータはそれぞれ専用のデータ・セグメントに格納され、個々の索引のデータはそれぞれ専用の索引セグメントに格納されます。

同じセグメントのエクステントは、同じ表領域に格納されます。ただし、それらのエクステントはディスク上で連続している場合と、連続していない場合があります。セグメントは複数のファイルにまたがることがありますが、1 つのエクステントが複数ファイルにまたがることはできません。

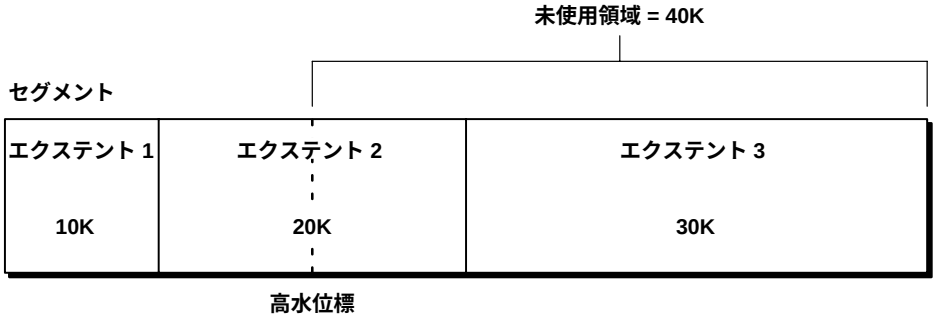
ユーザーは追加のエクステントを割り当てることができますが、ブロック自体は別個に割り当てられます。特定のインスタンスにエクステントを割り当てると、ブロックがただちに空きリストに割り当てられます。特定のインスタンスにエクステントを割り当てなかった場合、ブロック自体が割り当てられるのは高水位標が移動したときだけです。

高水位標

高水位標は、1 つのセグメントの使用済み領域と未使用領域との境界です。新しい空きブロックを求める要求（既存の空きリストでは満たすことができない要求）が出ると、高水位標が指しているブロックが使用済みブロックとなり、高水位標はその次のブロックに移ります。つまり、セグメントの中の高水位標の左側の領域が使用済みで、右側の領域が未使用ということになります。

図 11-2 に、それぞれ 10K、20K、30K の領域を占める 3 つのエクステントからなるセグメントを示します。高水位標は、2 番目のエクステントの中央にあります。したがって、このセグメントは、高水位標の左側にある 20K の使用済み領域と、高水位標の右側にある 40K の未使用領域からなります。

図 11-2 高水位標



関連項目：『Oracle8*i* 概要』（セグメントおよびエクステントの詳細）

空き領域管理のための構造体

Oracle は、次の構造体を使用して空き領域を管理します。

- トランザクション空きリスト
- プロセス空きリスト
- 空きリスト・グループ
- マスター空きリスト

プロセス空きリストは、インスタンス内で生じる各プロセス間の空き領域のための競合を緩和します。これは、複数のインスタンスを単一の空きリスト・グループにハッシュできる場合でも緩和されます。空きリスト・グループは、インスタンス間での強制的な読込み / 書込みを緩和します。プロセス空きリストおよび空きリスト・グループは、すべてのデータベース・オブジェクト、つまり、表、索引およびクラスタに対して一様にサポートされます。

トランザクション空きリスト

トランザクション空きリストは、コミットされていないトランザクションによって解放されたブロックのリストです。これらはデフォルトで存在します。トランザクションがコミットされると、解放されたブロックは最終的にマスター空きリストに加えられます。詳細は、「マスター空きリスト」の項を参照してください。

プロセス空きリスト

プロセス空きリストは、単に " 空きリスト " とも呼ばれており、セグメント内のいくつかの異なるエクステントから抽出できる空きデータ・ブロックのリストです。

空きリスト内のブロックには、PCTFREE より大きい空き領域が含まれています。PCTFREE は、既存の行への更新のために確保される空き領域がそのブロックで占める割合です。一般に、データベース・オブジェクトのプロセス空きリスト中のブロックは、PCTFREE および PCTUSED の制約を満たしている必要があります。これらの制約については、『Oracle8*i* 概要』の「データ・ブロック、エクステント、セグメント」の章を参照してください。

プロセス空きリストは、ユーザーが明示的に使用可能にします。プロセス空きリストの必要数を指定するには、表または索引、クラスタを作成するときに FREELISTS パラメータを設定します。FREELISTS パラメータの最大値は、システム上の Oracle ブロック・サイズによって決まります。さらに、各空きリストで、オーバーヘッドを処理するためにブロック内に一定数のバイトを格納することが必要です。

注意： 1 つの空きリスト用の予約領域および必要バイト数は、プラットフォームによって異なります。詳細は、使用しているシステム固有の Oracle マニュアルを参照してください。

空きリスト・グループ

空きリスト・グループを指定すると、1 つ以上の特定インスタンスで空きリストの集合を使用できます。各空きリスト・グループは、表およびクラスタに対する挿入または更新を収容する空きデータ・ブロックを提供するもので、起動時にインスタンスに関連付けられます。

パラレル・サーバーには複数のインスタンスがあるので、プロセス空きリストだけでは競合問題を解決できません。空きリスト・グループがあれば、インスタンス間の ping を効果的に緩和することができます。

使用可能になった空きリスト・グループは、空きリストの集合をサブセットに分割します。プロセス空きリストの記述が、空きリスト・グループごとに異なるブロックに格納されます。すべての空きリスト・グループのブロックは、同じ空きリストを指していますが、各インスタンスはそれぞれ専用の空きリストを取得します。（ただし、インスタンスが空きリスト・グループの数より多い場合は、複数のインスタンスが同じ空きリスト・グループにハッシュすることになります。）これにより、複数のインスタンスが同じブロックのために競い合うことはありません。

注意： OPS で空きリスト・グループを使用するときは、常にプロセス空きリストと一緒に使用してください。

マスター空きリスト

マスター空きリストは、表の中のエクステントから抽出された使用可能な領域を含むブロックのリポジトリです。デフォルトではこのマスター空きリストが存在し、その内容は次のとおりです。

- コミット済みのトランザクションが解放したブロック。空きブロックが必要になると、これらのブロックがマスター空きリストに加えられます。
- 特定の空きリスト・グループに関連付けられていない後続の領域割当て。高水位標が移動すると、各ブロックがマスター空きリストに加えられます。

空きリスト・グループが存在する場合は、各グループには専用のマスター空きリストがあります。さらに、ほとんどがパラレル操作のために使用される、中心的マスター空きリストも存在します。

セグメント・ヘッダーおよびマスター空きリストの競合を防止する

同時実行性の高い環境では、マスター空きリストが入っているセグメント・ヘッダーに対する競合が生じることが考えられます。

- 空きリスト・グループが存在する場合は、セグメント・ヘッダーは中心的なマスター空きリストのみを指します。この他に、すべての空きリスト・グループのブロックには、専用のマスター空きリストおよびトランザクション空きリスト、プロセス空きリストを指しているポインタがあります。
- 空きリスト・グループが存在しない場合は、セグメント・ヘッダーにはマスター空きリスト、トランザクション空きリストおよびプロセス空きリストを指しているポインタがあります。

単一インスタンス環境では、複数のプロセス空きリストの使用によってセグメント・ヘッダー・ブロックの競合が緩和されるため、多数のユーザーが空きデータ・ブロックを探す場合に生じる問題の解決に役立ちます。

図 11-3 に示すように、複数インスタンスの環境では、プロセス空きリストが使用可能なエクステントの空きデータ・ブロックをいくつかの異なるインスタンスに与えます。複数の空きリストを分割して、エクステントが特定のデータベース・インスタンスに割り当てられるようにすることができます。各インスタンスは 1 つ以上の空きリスト・グループにハッシュされ、また各グループのヘッダー・ブロックはプロセス空きリストを指します。

しかし、空きリスト・グループが割り当てられていない場合は、ファイルのセグメント・ヘッダー・ブロックがプロセス空きリストを指します。空きリスト・グループがない場合は、すべてのインスタンスが、空きリストにアクセスするためにセグメント・ヘッダー・ブロックを読み込む必要があります。

図 11-3 セグメント・ヘッダーに対する競合

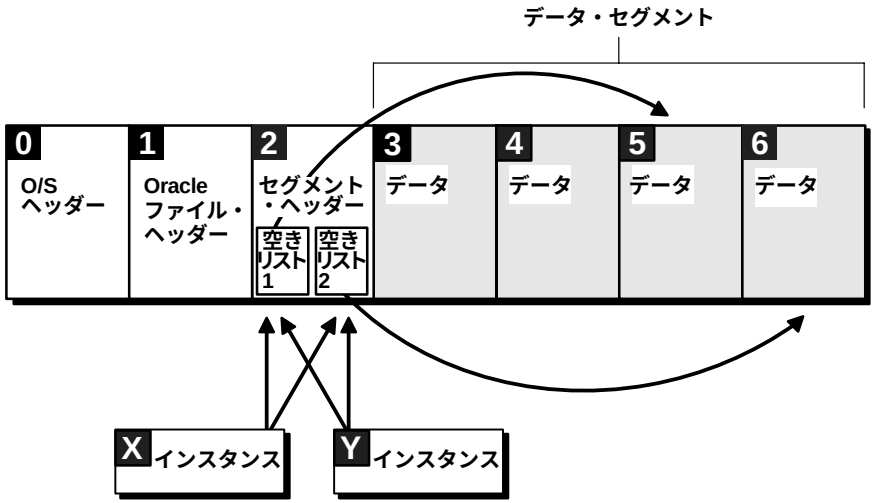
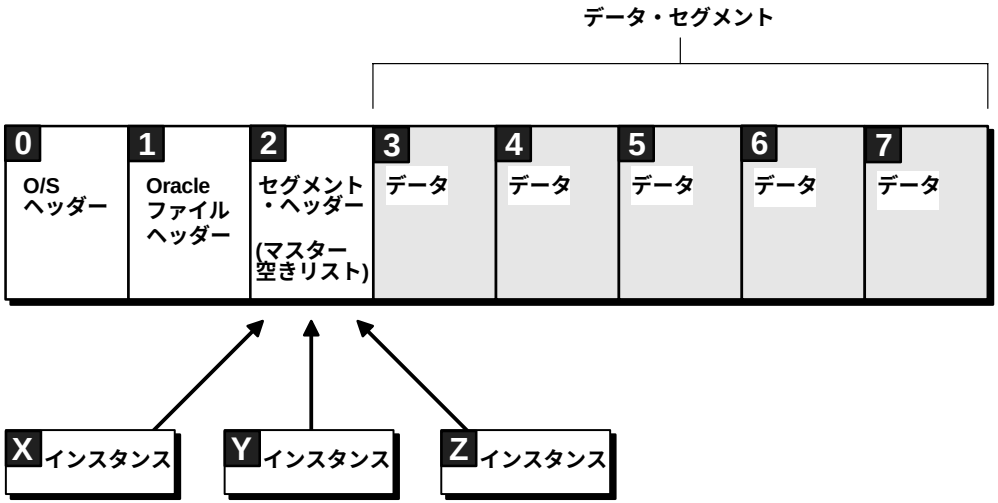


図 11-4 に、セグメント・ヘッダー・ブロックにマスター空きリストが格納されているファイルのブロックを示します。空き領域を取得するために、3 つのインスタンスでブロックの強制読込みが発生しています。空きリストは 1 つだけしかないので、挿入点も 1 つだけです。プロセス空きリストを使用すれば、この挿入点が複数のブロックに拡散し、各ブロックへのアクセス頻度が低下するので、競合は緩和されます。

図 11-4 マスター空きリストに対する競合



例 : 空きリスト・グループ

簡単な例

図 11-5 で、表の空き領域は、マスター空きリストと、空きリストを 3 つずつ含んだ 2 つの空きリスト・グループに分割されています。これは巧みに分割されたアプリケーションについての例であり、このアプリケーションでは削除が発生します。この例に示されたマスター空きリストは、この特定の空きリスト・グループ用のマスター空きリストです。

この表の作成時の初期エクステントは 1 つで、その後、エクステント 2 および 5 がインスタンス X に割り当てられ、エクステント 3 および 4 がインスタンス Y に割り当てられ、エクステント 6 が特定のインスタンスに対してではなく、自動的に割り当てられています。次の点に注意してください。

- 初期割当ておよびエクステント 6 の中の暗い影のブロックは、空きブロックのマスター空きリストを表しています。
- 薄い灰色のブロックは、空きリスト・グループ X の中の使用可能な空き領域を表しています。
- 濃い灰色のブロックは、空きリスト・グループ Y の中の使用可能な空き領域を表しています。
- エクステント 5 は新しく割り当てられたものであり、したがって、このエクステントのすべてのブロックは空きリスト・グループ X の中にあります。

- 黒のブロックは、削除によって解放された領域を表し、そのブロックは、空きリスト・グループ X および Y に戻ります。
- 影なしのブロックには、挿入用の十分な空き領域がありません。

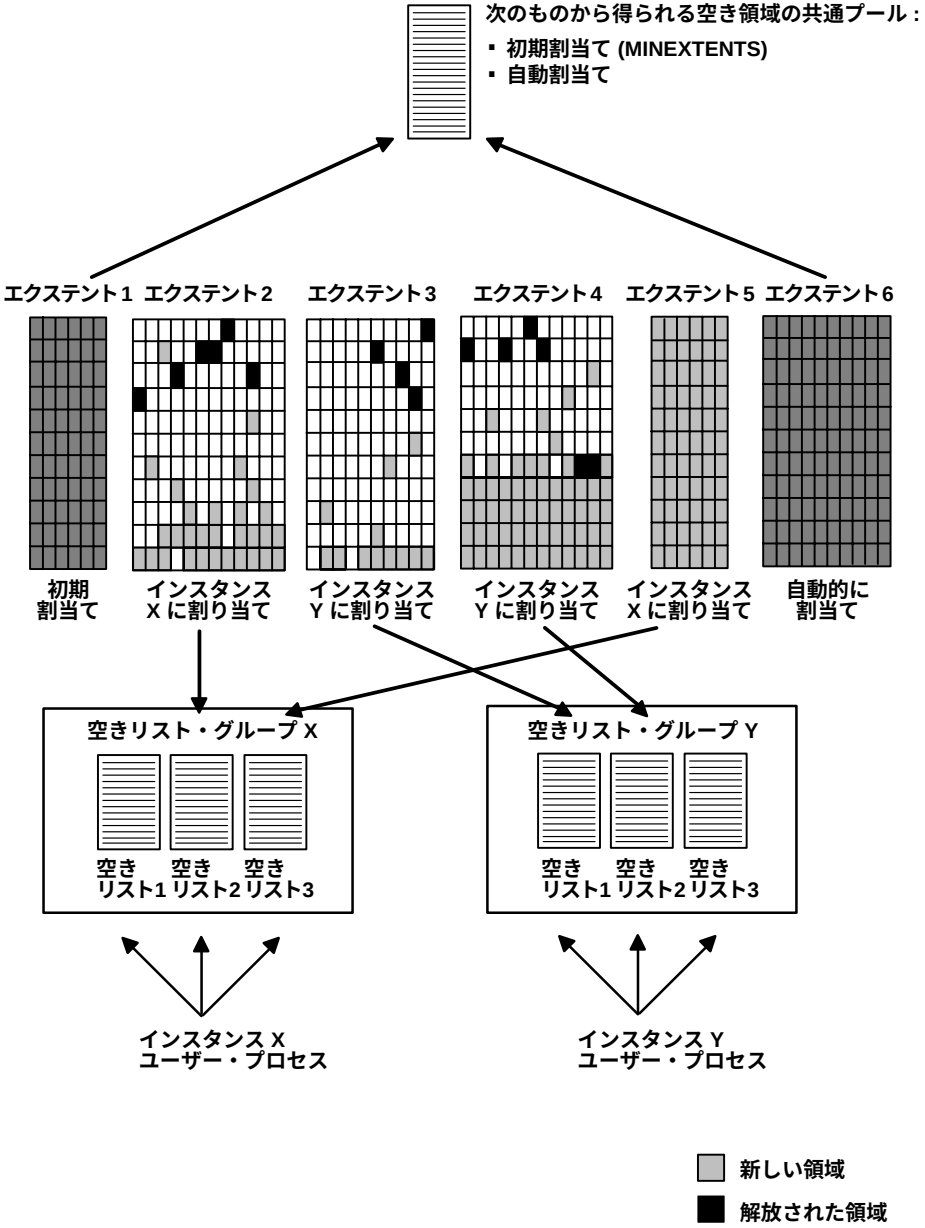
インスタンス X で実行する各ユーザー・プロセスはグループ X の空きリストの 1 つを使用し、インスタンス Y で実行する各ユーザー・プロセスはグループ Y の空きリストの 1 つを使用します。さらに多くのインスタンスが起動すると、それらのインスタンスのユーザー・プロセスは、インスタンス X または Y と空きリストを共有することになります。

複雑な例

図 11-5 では簡単な例を示していますが、インスタンスに割り当てられたエクステントは永続的なものではなく、また、1 つのインスタンスに割り当てられた領域は別のインスタンスによって使用できないことなどを考慮に入れると、複雑さが増してきます。各空きリスト・グループには、それぞれ専用のマスター空きリストがあります。ブロックを割り当てると、その一部はグループのマスター空きリストに加えられ、一部はプロセス空きリストに加えられます。また、どの空きリストにも加わらないブロックもあります。アプリケーションが全面的に分割された場合には、あるインスタンスに割り当てられたブロックは、そのインスタンスに割り当てられた状態のままになります。ただし、アプリケーションが全面的に分割されていなければ、ブロックは別のインスタンスに移動することができます。

次のような状況について考えてみます。たとえば、インスタンス Y のブロックが一杯になったために空きリストからそのブロックを取り去った後に、インスタンス X がブロックを解放したとします。このとき、インスタンス X、つまりブロックを解放したインスタンスの空きリストにブロックが移ります。インスタンス Y には領域が必要ですが、インスタンス Y はこのブロックを再利用できません。インスタンス Y は、自身の空きリスト・グループからのみ空き領域を取得できます。

図 11-5 表のための空きリストのグループ



空き領域の管理のための SQL オプション

SQL のオプションには、表およびクラスタ、索引用のプロセス空きリストおよび空きリスト・グループを割り当てるために使用できるものがいくつかあります。ユーザーは、オブジェクト用の新しい領域を特定のデータ・ファイルから取るように、明示的に指定できます。また、空き領域を特定の空きリスト・グループに関連付け、さらにそれらの空きリスト・グループを特定のインスタンスに関連付けることもできます。

SQL 文には次のものがあります。

```
CREATE TABLE [CLUSTER, INDEX]
  STORAGE
  FREELISTS
  FREELIST GROUPS
ALTER TABLE [CLUSTER, INDEX]
  ALLOCATE EXTENT
  SIZE
  DATAFILE
  INSTANCE
```

これらの SQL オプションを初期化パラメータ INSTANCE_NUMBER といっしょに使用することにより、データ・ブロックをインスタンスに関連付けることができます。

関連項目：『Oracle8*i* SQL リファレンス』（上記の文の構文について）

複数インスタンスでの空き領域の管理

この項では、次の項目について説明します。

- 空き領域の分割による複数の空きリストの作成
- 空きリスト・グループによるデータの分割
- インスタンスへの空きリストおよび空きリスト・グループの割当て方法

空き領域の分割による複数の空きリストの作成

個々の表、クラスタ（ハッシュ・クラスタ以外）および索引の空き領域を分割して複数のプロセス空きリストを作成できます。複数の空きリストがあると、プロセスは、領域が必要になったときに特定のブロック・プールを検索できるので、空き領域を求めるユーザー間の競合が緩和されます。1 つのインスタンスの中で空きリストを使用すると、複数のプロセスが同じ表への挿入を行うときに発生する競合を緩和することができます。

個々の表は、それぞれ使用可能領域のあるブロックのマスター空きリストを持つことができ、さらに複数の空きリストを持つことができます。ユーザー・プロセスは、マスター空きリストを調べる前に、適切な空きリストを走査して、十分な領域のあるブロックを見つけようとします。

空きリスト・グループによるデータの分割

空き領域をいくつかのグループに分割すると、OPS で複数のインスタンスが同時に挿入を実行したときに生じる空きデータ・ブロックのための競合が緩和され、パフォーマンスが向上します。OPS でプロセス空きリストのグループをいくつか作成し、各グループに表、索引またはクラスタ用の複数の空きリストを入れておくことができます。空きリスト・グループを使用すると、エクステントを特定のインスタンスに割り当てて、データを分割できます。

一般に、すべての表が同じ数の空きリスト・グループを持つことが必要ですが、グループ内の空きリストの数は、各表のアクティビティのタイプおよび量に応じて違っていてもかまいません。

空き領域の分割によって特にパフォーマンスが向上するのは、複数のインスタンスからの大量の同時挿入（または新しい領域を必要とする更新）を実行するアプリケーションです。もちろん、パフォーマンスの向上は、オペレーティング・システム、ハードウェア、データ・ブロック・サイズなどによっても左右されます。

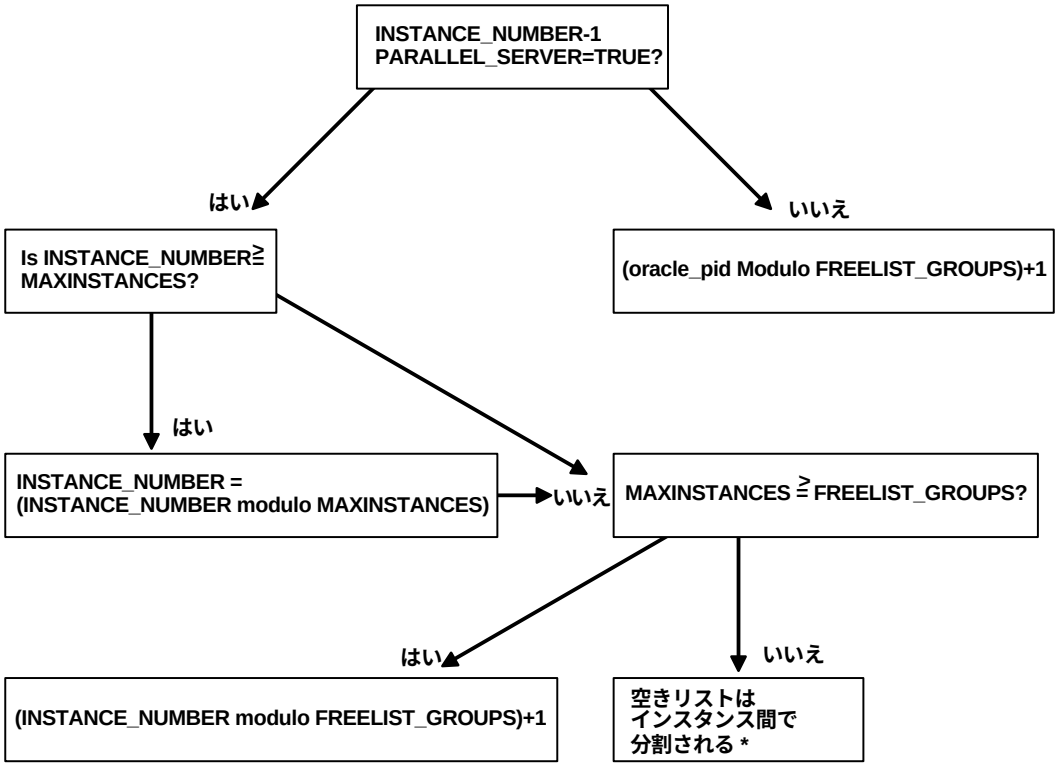
複数インスタンス環境では、複数の空きリストおよび空きリスト・グループについての情報はインポート時に保存されません。エクスポートおよびインポートを使用してデータのバックアップおよび復元を行う場合、再度分割されるような形でデータをインポートすることは困難です。

関連項目：「Import ユーティリティおよび Export ユーティリティでの空きリスト」および第 12 章「アプリケーションの分析」（データ分割の詳細について）。また、『Oracle8*i* チューニング』も参照してください。

インスタンスへの空きリストおよび空きリスト・グループの割当て方法

図 11-6 に、インスタンスへの空きリストおよび空きリスト・グループの割当て方法を示します。

図 11-6 空きリストおよび空きリスト・グループの割当て方法



ALTER SESSION INSTANCE_NUMBER 文を使用すると、インスタンス番号を MAXINSTANCES 値より大きくすることができます。図 11-6 では、この可能性がどのように処理されるかを示しています。つまり、空きリスト・グループは内部計算によって割り当てられますが、その内部計算では、インスタンス番号が MAXINSTANCES 値を超えないように処理されます。

* 空きリストは次のように分割されます。インスタンスの数が 3 で空きリスト・グループの数が 35 であるとすれば、インスタンス 1 が最初の 12 個の空きリスト・グループ、インスタンス 2 が次の 12 個のグループ、そしてインスタンス 3 が残りの 11 個のグループを、それぞ

れ使用します。実際の空きリスト・グループ・ブロックは、空きリスト・グループの数で *oracle_pid* をハッシュすることによって決定されます。

インスタンス、ユーザーおよびロックに関連付けられた空きリスト

この項では、次の項目について説明します。

- インスタンスと空きリストの関連付け
- ユーザー・プロセスと空きリストの関連付け
- PCM ロックと空きリストの関連付け

インスタンスと空きリストの関連付け

1 つの表に、それぞれが特定のインスタンスに割り当てられた複数のプロセス空きリストのグループが存在できます。空きリストの各グループをそれぞれ 1 つのインスタンスに関連付けることも、いくつかのインスタンスが 1 つの空きリスト・グループを共有することもできます。また、すべてのインスタンスが、使用可能な領域のマスターの空きリストにもアクセスできます。

空きリストのグループを使用すると、新しい領域を必要とする同時挿入および同時更新の場合に、個々のインスタンスをそれぞれ異なるデータ・ブロックの集合に関連付けることができます。これによって、空きブロックのマスター空きリストについての情報が入ったセグメント・ヘッダー・ブロック獲得のための競合が減少します。複数の空きリスト・グループを持たない表の場合、セグメント・ヘッダーには、ユーザー・プロセス用の空きリストの情報も入っています。空きリスト・グループを使用すると、インスタンスが頻繁に挿入およびアクセスするデータを、そのインスタンスに割り当てたエクステントに入れておくことができます。

データの分割によって、データ・ブロックに対する競合を緩和することができます。一般に、1 つの空きリスト・グループのブロックを受け持つ PCM は、主にその空きリスト・グループを使用するインスタンスによって保持されることが多くなります。データを変更するインスタンスは、一般に他のインスタンスよりそのデータを再使用することが多いからです。ただし、複数のインスタンスが同じエクステントから空き領域を取得し、その後各インスタンスが挿入したデータを変更する場合は、そのエクステント内のブロックを求める競合の発生度が高くなります。

既存の空きリスト・グループへの新しいインスタンスの割当て

表またはクラスタの中の空きリスト・グループの数より MAXINSTANCES の方が大きい場合は、インスタンス番号は、次のように、関連付けられた空きリスト・グループにマップされます。

instance_number modulo *number_of_free_list_groups*

" モジュロ "（または " 剰余 " を表す "rem"）は、剰余の値を計算して、どの空きリスト・グループを使用するかを決定する公式です。たとえば、空きリスト・グループの数が 2 で、インスタンスの数が 10 であるとします。インスタンス 6 がどの空きリスト・グループを使用するかを判別するための公式は、6 モジュロ 2 = 0 となります。つまり、6 を 2 で割ると商は 3 で剰余は 0 なので、インスタンス 6 は空きリスト・グループ 0 を使用することになります。同様に、5 モジュロ 2 = 1 なので、インスタンス 5 は空きリスト・グループ 1 を使用します。5 を 2 で割ると剰余は 1 だからです。

MAXINSTANCES より空きリスト・グループの数の方が多い場合は、別のハッシュ機構が使用されます。

複数のインスタンスが 1 つの空きリスト・グループを共有している場合、それらのインスタンスに明確に割り当てられているすべてのエクステントへのアクセスも、共有することになります。

FREELIST GROUPS および MAXINSTANCES

クラスタ化されたシステムなどの比較的ノード数の少ないシステムでは、通常、表を対象とした FREELIST GROUPS オプションには、CREATE DATABASE の MAXINSTANCES オプションと同じ値を指定します。これにより、データベースに同時にアクセスできるインスタンスの数を制限します。

ただし、大規模なパラレル・システムでは、MAXINSTANCES が FREELIST GROUPS の何倍にもなると、複数のインスタンスが 1 つの空きリスト・グループを共有することになります。

関連項目： 17-9 ページの「インスタンス、ユーザーおよびロックと空きリスト・グループとの関連付け」

ユーザー・プロセスと空きリストの関連付け

ユーザー・プロセスは、それぞれの Oracle プロセス ID に基づいてプロセス空きリストに関連付けられます。個々のユーザー・プロセスは、そのプロセスを実行しているインスタンス用の空きリスト・グループ内の 1 つの空きリストだけにアクセスできます。また、各ユーザー・プロセスは、空きブロックのマスター空きリストにもアクセスできます。

ある表に、複数の空きリストがあり複数の空きリスト・グループがない場合、または空きリスト・グループの数がインスタンスの数より少ない場合は、各空きリストは、異なるインスタンスのユーザー・プロセスで共有されます。

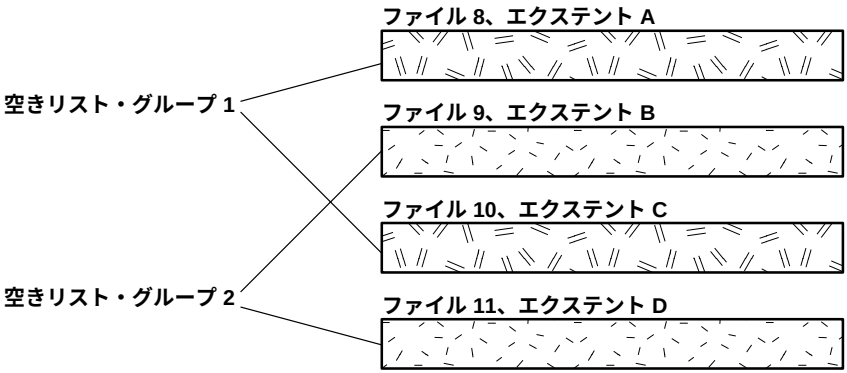
PCM ロックと空きリストの関連付け

表の各エクステントが別個のデータ・ファイルに入っている場合は、GC_FILES_TO_LOCKS パラメータを使用して、特定範囲の PCM ロックを各エクステントに割り当て、PCM ロックの各セットを 1 つの空きリスト・グループだけに関連付けることができます。

図 11-7 に、複数のエクステントが別々のファイル内に入っている場合を示します。GC_FILES_TO_LOCKS パラメータは、ファイル 8 および 10 に 10 個のロックを、そしてファイル 9 および 11 に 10 個のロックを割り当てています。エクステント A および C は同じ空きリスト・グループ内にあり、エクステント B および D は別の空きリスト・グループ内にあります。ファイル 8 および 10 には PCM ロックの集合の 1 つが割り当てられ、ファイル 9 および 11 には別の集合が割り当てられます。同じ空きリスト・グループに属するファイル、たとえば、ファイル 8 および 10、またはファイル 9 および 11 については、ロックを別々に分ける必要はありません。

図 11-7 エクステントおよび空きリスト・グループ

GC_FILES_TO_LOCKS = 8, 10:10; 9, 11:10



この例では、読込みおよび書込みのすべての分割を想定しています。複数のインスタンスがブロックの更新を行う場合にも、強制的な読込みおよび書込みを最小限に抑えるためには、各ファイルごとに複数のロックを持つことをお薦めします。なぜなら、共有ロックを使用した場合にも、ロックされたブロックのうちの 1 つを別のインスタンスが読込みを行おうとしただけで、ロックによって保持されたすべてのブロックの強制読込みが行われるからです。

関連項目： 15-6 ページの「GC_FILES_TO_LOCKS の設定 : 各データ・ファイルの PCM ロック数」

エクステントの割当ての制御

この項では、次の項目について説明します。

- 新しいエクステントの自動割当て
- 新しいエクステントの事前割当て
- ロック境界でのブロックの動的割当て

表に行を挿入するときに、新しいエクステントの割当てが必要になると、所定の数の連続したブロック（GC_FILES_TO_LOCKS パラメータで指定した !*blocks*）が、インスタンスに関連付けられた空きリスト・グループに割り当てられます。表またはクラスタの初期作成時に割り当ててあるエクステントのブロック、および自動割当てされた新しいエクステントのブロックが、マスター空きリストまたは高水位標よりも上の領域に追加されます。

新しいエクステントの自動割当て

インスタンスを指定しないで明示的にエクステントを割り当てた場合、またはシステムが領域を使い果たしてしまった（つまり高水位標が先へ進まなくなった）ためにエクステントが自動的にセグメントに割り当てられた場合は、その新しいエクステントは未使用領域の一部となります。そのエクステントはエクステント・マップの末尾に置かれます。したがって、現在の高水位標は、新しいエクステントの " 左側 " のエクステント内にあります。つまり、新しいエクステントは高水位標の " 上位 " に追加されることになります。

新しいエクステントの事前割当て

新しいエクステントの割当てを制御するための 2 つのオプションがあります。

- 空きリスト・グループへのエクステントの事前割当て
- ロック境界でのブロックの動的割当て

空きリスト・グループへのエクステントの事前割当て

エクステントの事前割当ては、Oracle によるエクステントの自動割当てを抑止するための、静的な解決方法です。ユーザーは、空きリスト・グループを持つ表に、エクステントを事前に割り当てることができます。その結果、すべての空きブロックが空きリストにフォーマットされます。これらの空きリストは、エクステントを事前割当てしようとしているインスタンスの空きリスト・グループ内に常駐することになります。この方法が便利なのは、挿入時の ping を大幅に削減するためにデータを分割する必要がある場合、または、増大が予測されるオブジェクトを収容する必要がある場合です。

注意： false ping を完全に排除することはできません。

関連項目： 17-10 ページの「エクステントの事前割当て（オプション）」

ロック境界でのブロックの動的割当て

成長分を収容することが主な要件である場合は、空きリスト・グループへのブロックの動的割当ての方が、エクステントの事前割当てより効率的です。GC_FILES_TO_LOCKS の !*blocks* オプションを使用すれば、ロック境界内の高水位標からブロックを空きリストに動的に割り当てることができます。この方法は、セグメント・ヘッダーに対するすべての ping を排除するものではありません。むしろ、この方法は、必要に応じてブロックを割り当てるので、エクステントを事前に割り当てる必要はありません。

ロックはインスタンスが所有するものであることに注意してください。ブロックはインスタンス単位で割り当てられるものであり、ブロックが空きリスト・グループに割り当てられる理由もそこにあります。1 つのインスタンスの中では、個々のブロックを別々の空きリストに割り当てることができます。

この方法を使用する場合は、!*blocks* の値を明示的に割り当てることも、新しいブロック数のバランス調整をそのまま既存の PCM ロックに任せることもできます。後者を選択する場合は、他のインスタンスへの割当てに伴って、既存の PCM ロックに対する競合が依然として生じる可能性があるので注意してください。PCM ロックが複数のブロック・グループを受け持っている場合は、ロックによって保持されたすべてのブロックの強制読込みおよび書込みが不必要に行われることがあります。

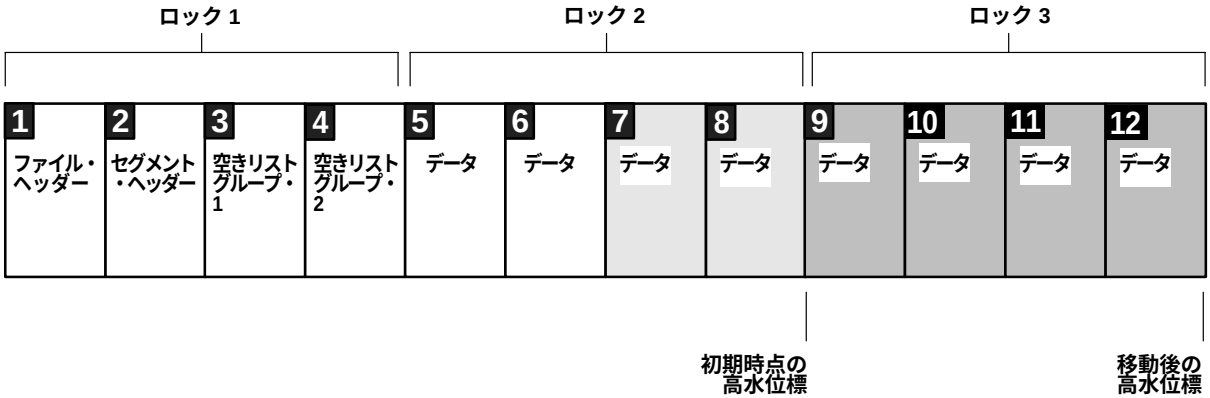
関連項目： 17-14 ページの「エクステントの動的割当て」

セグメントの高水位標の移動

セグメントの高水位標は、そのセグメントに割当て済みのブロック数についての現在の限界値です。エクステントを動的に割り当てる場合は、高水位標はロック境界でもあります。ロック境界は、1 つのエクステント内で同時に割り当てられるブロック数に一致している必要があります。この値はすべてのインスタンスで同じにする必要があります。

たとえば、次の例では、1 ロックにつき 4 つのブロックがあります（!4）。ロックは、ブロックの内容を入力する前に割り当てられています。ここで、ロック 2 が保持しているデータ・ブロック D2 の内容を入力してから、別の一連の 4 ブロックを割り当てたとすると、実際に割り当てられるのはロック境界の範囲内に収まる数のブロックだけです。この例では、それはブロック 7 および 8 です。これらのブロックは、どちらも現行ロックによってプロテクトされます。高水位標が 8 にあるときに、インスタンス 2 が一連のブロックを割り当てると、ブロック 9 ～ 12 の 4 ブロックすべてが、ロック 3 の受持ちに割り当てられます。次回インスタンス 1 がブロックを割り当てると、ブロック 13 ～ 16 がロック 4 の受持ちに割り当てられます。

図 11-8 ブロックの割当てに伴って高水位標が移動するファイル



例 この項に示す例では、どちらのインスタンスについても、GC_FILES_TO_LOCKS の設定が次のとおりであるものとします。

```
GC_FILES_TO_LOCKS = "1000!5"
```

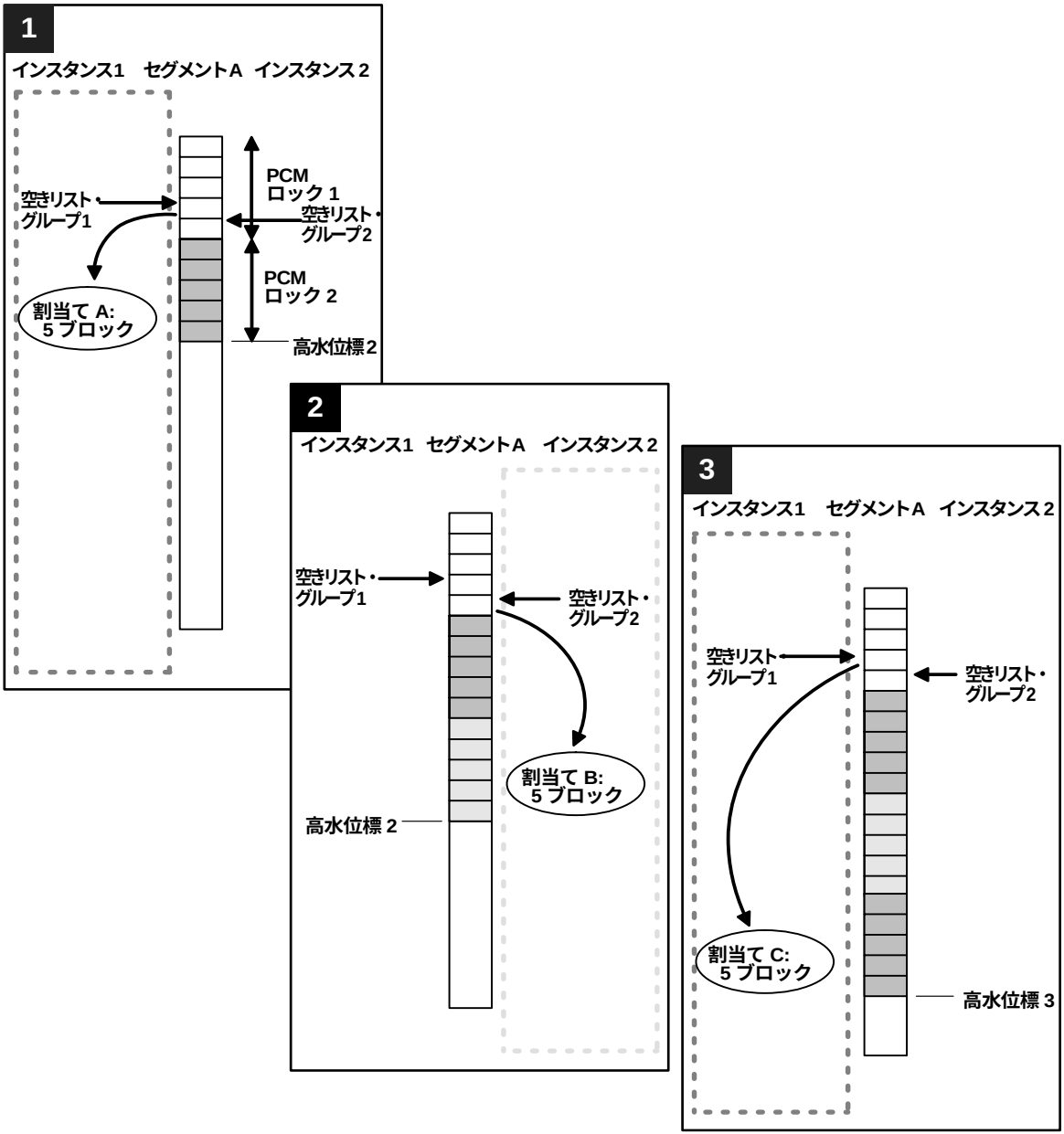
EACH オプションが指定されていると、*file_list* 内の各ファイルに、*#locks* の数の PCM ロックが割り当てられます。各ファイルの中では、*!blocks* によって、各ロックが受け持つ連続データ・ブロックの数を指定します。

図 11-9 に、増分によるセグメントの成長のしかたを示します。

- ステージ 1 に示すエクステントには、インスタンス 1 が 5 個のデータ・ブロックを割り当て、それらのデータ・ブロックをロック 2 がプロテクトする。
- ステージ 2 では、インスタンス 2 がさらに 5 個のデータ・ブロックを割り当て、それをロック 3 がプロテクトする。
- ステージ 3 では、インスタンス 1 がさらに 5 個のデータ・ブロックを割り当て、ロック 4 がプロテクトする。

この例で、インスタンス 1 のユーザー A がブロック 10 を操作している場合、どのインスタンスのどのユーザーも、ロック 2 の対象範囲内のブロックを使用できません。ロック 2 には、ブロック 6 ～ 10 が含まれます。

図 11-9 エクステント内のブロックの割当て



12

アプリケーションの分析

この章では、Oracle Parallel Server(OPS）アプリケーション設計を最適化するための概念的なフレームワークについて説明します。この章には次の項があります。

- 分析に必要な詳細度
- アプリケーション・プロファイル
- 分割のガイドライン

関連項目：『Oracle8*i* チューニング』（パフォーマンス調整の原理および方法について）

分析に必要な詳細度

アプリケーションをどの程度まで詳細に分析する必要があるかは、OPS の使用目的に応じて異なります。OPS によってデータベース全体のスループットを改善する必要がある場合は、データベース設計およびアプリケーション作業負荷プロファイルの詳細な分析が必要になります。詳細な分析により、OPS の各ノードが提供する追加の CPU パワーが、アプリケーション処理のために完全に使用されることが確実になります。主として高い可用性を得るために OPS を使用する場合にも、十分な分析を行うことにより、どのぐらいのリソースが必要となるかを見積ることができます。

多くのベンチマーク・アプリケーションおよび実際のアプリケーションでの実績によると、OPS システムでパフォーマンスを最適化するには、パラレル・キャッシュ管理に使用するコンピューティング・リソースを最小限に抑える必要があります。これは、インスタンス・ロックの操作回数を最小限に抑えることを意味します。最適な OPS システムでは、各ノードで行うインスタンス・ロックの操作回数がきわめて少ないため、マシン間の高速インターコネクトの通信量が、クラスタの設計上の制限内に収まります。

データベース存続期間における最終的な微調整フェーズの時点では、PCM ロック・オペレーション数の最小化を効果的に行うことができません。したがって、データベース設計プロセスの初期段階でアプリケーションの分析を計画することが必要です。

関連項目： 第 13 章「パラレル・サーバー用のデータベース設計」（OPS を十分に活用できるようなアプリケーションの設計方法を示す事例）

アプリケーション・プロファイル

アプリケーション・プロファイルを理解するには、アプリケーションの機能およびアクセス・パターンに従って表を分類する必要があります。この項では、次の項目について説明します。

- アプリケーションの機能および表アクセス・パターンの分析
- 読込み専用表
- ランダム SELECT および UPDATE 表
- INSERT、UPDATE または DELETE 表
- 実装の計画

以降の内容は、クラスタ化された表およびクラスタ化されていない表のどちらにもあてはまります。

アプリケーションの機能および表アクセス・パターンの分析

OPS のデータベース設計者は、アプリケーションおよびデータについての通常の分析フェーズに加えて、過剰なロック変換率をもたらすようなトランザクションまたは業務機能のタイプを予測する必要があります。主要なアプリケーション表および各表のアクセス・パターンと、アプリケーションの機能との間のクロス・リファレンスを行う必要があります。

関連項目： 第 13 章「パラレル・サーバー用のデータベース設計」（表アクセス・パターンの分析に使用できるワークシートについて）

読込み専用表

主に読込み専用として使用する表の場合、すべての OPS ノードが速やかに PCM ロックを共有モードに初期化するので、ロック・アクティビティが生じることはほとんどありません。読込み専用表およびそれぞれに関連する索引構造体が必要とする PCM ロックの割当て量はわずかです。この種の表を OPS で使用すると、優れたパフォーマンスおよび拡張性を期待できます。

また、SQL の ALTER TABLESPACE READ ONLY 文を使用して、表を読込み専用表領域に入れることもできます。これにより、特定の表領域へのアクセスおよび回復全般でスピードアップが達成され、PCM ロックは不要になり、表領域のバックアップは表を読込み専用にした後で 1 度だけ作成すればよいので、読込み専用表にはいくつかの利点があります。

OPS でのパラレル問合せの拡張性は、ノード間のインターコネクトの速度に左右されます。プロセッサをビジー状態にしておくためだけに、高度の並列性を維持することが必要な場合

があります。並列度がノード数（またはプロセッサ数）の 3 倍に達することも、珍しくはありません。

アプリケーションが読込み専用であっても、これらのファイルには、GC_FILES_TO_LOCKS パラメータで指定した専用の PCM ロックが必要です。ソート / マージ結合を使用したり、または GROUP-BY および ORDER-BY によって大きいソートを行う問合せなどでは、SYSTEM 表領域内のデータ・ディクショナリを更新できます。

関連項目： 2-1 ページの「4 つのレベルの拡張性」および『Oracle8*i* チューニング』の第 26 章「ステップ 2: 並列度の設定とマルチユーザー問合せ調整の使用可能化」

ランダム SELECT および UPDATE 表

ランダム SELECT および UPDATE 表、もしくは、非パーティション化表には、表の中のあらゆる行を読み込んで更新するトランザクションが含まれています。このアクセス・タイプには、多数のロック変換が必要になります。最初に、このトランザクションを実行するインスタンスは、データ・ブロックに対する共有 PCM ロックを取得する必要があります。このロックが原因で、他のノードでロック・ダウングレード操作が生じることがあります。トランザクションを実行するインスタンスは、UPDATE を実際に実行する時点で、最終的には排他モード PCM ロックを取得する必要があります。

それぞれ異なるノードで実行している複数のユーザー・トランザクションが、同じ PCM ロックによってロックされた複数のデータ・ブロックを同時に変更しようとした場合、パフォーマンスが大幅に低下します。追加のハッシュ PCM ロックを作成することで、競合が緩和される場合もあります。ただし、大きい表では、ハードウェアの制約および実際的な制約によって、有効に使用できるハッシュ PCM ロックの数が制限されることがあります。たとえば、偽の競合を削減するには、何百万ものハッシュ PCM ロックが必要となるでしょう。しかし、メモリーの制約および起動時間の関係でそれは不可能です。この場合は、ファイン・グレインまたは解放可能なハッシュ・ロックを選択するとよい結果を得ることができます。

この種の表において、表の索引キーが 1 つも更新されない場合、索引の PCM ロックは共有モードに変換されるだけなので、必要な PCM ロックの数は少なくてすみます。

関連項目： 詳細は、13-16 ページの「ハッシュ・ロックとファイン・グレイン・ロックの実装」を参照してください。

INSERT、UPDATE または DELETE 表

ランダム INSERT、UPDATE および DELETE 表についてのトランザクションでは、多数のデータ・ブロックを読み込み、そしてそれらのデータ・ブロックの一部またはすべてを変更することが必要になります。指定した各データ・ブロックを対象とするこのプロセスでも、PCM ロックを共有モードに変換してから、ブロックの変更時にそれを排他モードに変換することが必要です。このプロセスについても、ランダム SELECT および UPDATE 表の場合と同じパフォーマンス上の問題点があります。

この種の表には、2 つの主な理由によって、多くのパフォーマンス問題が起こります。それは、索引データ・ブロックが変更されることと、表の空きリストでのデータ・ブロック獲得のための競合が生じることです。

索引キーを変更する INSERT、DELETE および UPDATE トランザクションでは、ユーザーが表の索引をメンテナンスする必要があります。このプロセスは追加の索引ブロックの変更を必要とするので、発生するロック変換の数が増加する可能性があります。さらに、他のノードのユーザーが索引を使用して他のデータにアクセスすることになるため、索引ブロックではさらに多くのロック変換が必要となる可能性があります。これは、特に、ブロックの分割が行われる索引の初期ルート・コンポーネントについて言えることです。結果として、クラスタ内のすべてのノードにおいて、NULL から排他への変換、およびその逆の変換が増加します。

INSERT 操作および DELETE 操作が長時間実行トランザクションに含まれている場合は、他のインスタンスでトランザクションの完了のために読込み一貫性情報が必要になる可能性が大きくなります。また、ロールバック・セグメント・データ・ブロックがディスクにフラッシュされ、他のインスタンスで使用可能にされると、必要なロック変換がさらに増加します。

OPS を実装したシステムにおいてパフォーマンスが重要な要素である場合は、高いロック変換率をもたらす索引ブロック競合はどんな犠牲を払っても回避する必要があります。

複数の OPS ノードから、順序番号ジェネレータを使用して 1 つの表にそれぞれ一意のキーを生成する場合は、索引ブロックの競合はさらに深刻になります。一意のキーを生成する場合は、主キーの一部にインスタンス番号を使用して、各インスタンスがそれぞれ索引の異なる部分へ INSERT するようにしてください。INSERT の負荷を索引の全域に分散させることで、単一インスタンスおよび複数インスタンスの両方のパフォーマンスが向上します。それには、逆キー索引を使用します。

INSERT 操作では、エクステント内に空き領域を割り当てる場合も、ロック変換率が高まることがあります。これは、複数のインスタンスが、同じデータ・ブロックまたは近接するデータ・ブロックに、新しい行を挿入しようとすることがあるからです。これらのデータ・ブロックを同じ PCM ロックが管理していると、競合が生じます。この事態を避けるには、表および索引を分割して複数のインスタンスが使用できるようにするか、または複数の表を作成して複数の空きリストおよび空きリスト・グループを使用できるようにします。

関連項目： 第 17 章「空きリスト・グループによるデータ分割」および『Oracle8*i* 概要』

実装の計画

アプリケーションの作業負荷を分析したら、次に、アプリケーションの OPS 実装の計画を立てることができます。アクセス・プロファイルを使用すると、複数のノードで実行すべきトランザクションか、単一のノードで実行すべきトランザクションかを判別できます。多くの場合、アプリケーションについて必要なパフォーマンスを確保するには、折衷案やトレードオフが必要になります。

注意： ノード間でのロード・バランシングは、主要な目標ではありません。ロード・バランシングはベンチマークの状況下では有用ですが、実際のアプリケーションでは実用的でないことがあります。OPS でのパフォーマンスの主要要素は分割です。

分割のガイドライン

この項では、次の項目について説明します。

- 概要
- アプリケーションの分割
- データの分割

概要

データベース設計者は、アプリケーションの分割によって生じるシステム・パフォーマンスへの影響と設計との釣合いについて明確に理解している必要があります。目標は同期の最少化であることを、常に念頭においてください。そうすることで、最適なパフォーマンスを得ることができます。

前述のとおり、ロック変換を最小限に抑えれば、OPS のパフォーマンスの向上を予測でき、スケーラブルなシステムを実現できます。アプリケーションまたはデータ（あるいはその両方）を分割することで、クラスタの特定ノードから見たデータベース・データのキャッシュの親和性を確保し、維持できます。分割したアプリケーションでは、ロック変換は最小限に抑えられるので、データ・ブロックの ping および統合分散ロック・マネージャ（IDLM）アクティビティはきわめて少なくなります。分割したアプリケーションで過度の IDLM ロック・アクティビティが発生する場合は、分割の方針が適正でないか、またはデータベースの作成および調整プロセスに誤りがあったと考えられます。

アプリケーションの分割

高いシステム・パフォーマンスを得るための、さまざまな分割方法があります。データベース上の負荷を分散つまり分割するための最も簡単な方法の 1 つは、同じデータベースにアクセスする複数のアプリケーションを、それぞれ、クラスタの別々のノードで実行することです。たとえば、一方のアプリケーションはあるデータ・ファイルの集合に常駐する同じ表の集合だけを参照し、もう 1 つのアプリケーションは、異なるデータ・ファイルの集合に常駐する異なる表の集合を参照するものとします。データ・ファイルにそれぞれ異なる PCM ロックを割り当てれば、この 2 つのアプリケーションをクラスタの別々のノードで実行することで、優れたパフォーマンスを得ることができます。この 2 つのアプリケーションでは、（データがそれぞれ異なるファイル内に存在するので）同じデータベース・オブジェクトに対する競合が生じません。したがって、同じデータベース・ブロックに対する競合が生じることもありません。

この手法が特に効果的になるのは、日中に多数のユーザーおよび負荷の高い OLTP をサポートし、夜間には負荷の高いバッチ処理および意思決定支援を実行するアプリケーションです。この場合は、アプリケーションをクラスタのノード間で分割することによって、日中の効率のよい OLTP パフォーマンスを確保できます。

この事例は、同時にアクセスする表をまとめて格納しておく分散データベース・モデルに似ています。夜間に、OLTP の効率化のために分割された表へのアクセスが必要になったときも、すべてのデータが 1 つのデータベース内に効率よく格納されているので、単一データベースの利点を十分に活用できます。これにより、バッチおよび意思決定支援の処理効率の改善、問合せパフォーマンスの向上、ネットワーク通信量の削減、データ・レプリケーション問題の減少が達成されます。

この手法では、各アプリケーションの表および索引を格納するときに、両方のアプリケーションが使用するデータ・ブロックを同じ PCM ロックが受け持つことがないようにする必要があります。このようにしないと、分割の目的は失われます。正しい状態を確保するには、各アプリケーションの表データおよび索引データを別々のデータ・ファイルに格納します。

SQL 文の集合を複数のアプリケーションで共有する場合は、それらのアプリケーションを同一インスタンスで実行すると、パフォーマンスが向上します。共有 SQL 領域は複数のインスタンス間で共有されるものではないので、1 つのインスタンスで SQL 文の類似した集合を実行することによって、メモリーの使用効率を向上させ、解析の量を削減できます。

データの分割

場合によっては、ノード間でアプリケーションを分割ができないことがあります。そのときの代替手段として、データベース・オブジェクト自体を分割することができます。これを効果的に行うには、アプリケーション・プロファイルを徹底的に分析する必要があります。1 つの表を複数の表に分割する必要がある場合とその必要がない場合があります。OPS での分割プロセスでは、事前定義済みのキー範囲間での表の水平分割が必要になることがあります。

データベース・オブジェクトを分割するだけでなく、各ユーザー・トランザクションが正しい OPS インスタンスで実行されることも確保する必要があります。トランザクションを実行するための正しいノードは、トランザクションで実際に使用するデータ値によって決まります。このプロセスは一般に、データ依存ルーティングと呼ばれています。

パラレル・サーバーのパフォーマンスを強化するための表の分割のプロセスには、開発および管理上のさまざまな留意事項があります。

開発の観点から見ると、表の分割が完了すると同時に、アプリケーション・コードの量および複雑度が増します。さらに、表を分割することで、バッチおよび意思決定支援問合せなどのような他のアプリケーション機能のパフォーマンスが低下することがあります。

データ依存ルーティングを達成するには、どの方法を使う場合でも次のことを考慮する必要があります。表の分割が現在使用されている分割パターンにうまく収まる場合、つまり、表を状況またはコール・センター別に分割してあり、これと同じ方法でユーザーも分割可能な場合は、手動式のルーティングによってユーザーを適切なインスタンスと接続させることができます。そうでない場合は、データ依存ルーティングの管理が複雑になり、アプリケーション・コードの追加が必要になります。

このプロセスを簡単にするには、アプリケーションでトランザクション処理モニター（TPM）または RPC メカニズムを使用します。入力 RPC 引数に基づいて、データ依存ルーティング方針を TPM の構成としてコーディングできます。同様に、どのインスタンスがトランザクションを実行すべきかを決定する CASE 文を使用して、プロシージャ・コードとしてこのプロセスをコーディングすることもできます。

関連項目： 1-20 ページの「クライアント / サーバー・システム」および『Oracle8*i* チューニング』（分割の詳細について）

第 III 部

Oracle Parallel Server の開発手順

13

パラレル・サーバー用のデータベース設計

この章では、Oracle Parallel Server（OPS）に最適なシステムを設計するための一般的な方法について説明します。

- 概要
- ケース・スタディ : 初期のデータベース設計から OPS へ
- 表へのアクセスの分析
- ユーザー別のトランザクション量の分析
- ユーザーおよびデータの分割
- 索引の分割
- ハッシュ・ロックとファイン・グレイン・ロックの実装
- 設計のインプリメントおよびチューニング

概要

この章では、OPS で使用する新規アプリケーションの設計テクニックを示します。これらの分析テクニックは、既存のアプリケーションを評価し、それらのアプリケーションがパラレル・サーバーへの移行にどの程度適しているかを調べるためにも使用できます。

注意： パフォーマンスの最適化を達成するために、常に同期の最小化をめざしてください。

この章では、初期のデータベース設計がすでに完了していることが前提になっています。OPS に最適な設計を実現するには、次の方法に従います。

1. 初期のデータベース設計を作成する。
2. 表へのアクセスを分析する。
3. トランザクション量を分析する。
4. ユーザーおよびデータの分割方法を決定する。
5. 必要な場合、索引の分割方法を決定する。
6. ハッシュ・ロックまたはファイン・グレイン・ロックを選択する。
7. 設計をインプリメントしチューニングする。

ケース・スタディ : 初期のデータベース設計から OPS へ

この章では、ケース・スタディを使用して具体的な分析テクニックを示します。アプリケーションごとの違いはありますが、この例は必要なプロセスを理解するのに役立ちます。

- "Eddie Bean" カタログ販売
- 表
- ユーザー
- アプリケーション・プロファイル

"Eddie Bean" カタログ販売

このケース・スタディでは、"Eddie Bean" カタログ販売会社についてのものです。この会社には、さまざまな製品の電話注文を受ける多数の注文入力オペレータがいます。発送処理オペレータが注文書を作成し、売掛処理オペレータが請求書を取り扱います。買掛処理オペレータは、社内で必要な物品やサービスについての発注業務を処理します。セールス・マネージャと財務アナリストは、データについてのレポートを実行します。この会社の財務アプリケーションには、単一のデータベースで稼働する次の 3 つの業務プロセスがあります。

- 注文入力
- 買掛金
- 売掛金

表

Eddie Bean データベースには次の表が含まれています。

表 13-1 "Eddie Bean" 社のサンプル表

表	内容
ORDER_HEADER	注文番号、顧客の名前と住所
ORDER_ITEMS	注文した製品、量、価格
ORGANIZATIONS	顧客および供給業者の名前、住所、電話番号
ACCOUNTS_PAYABLE	物品とサービスについて、社内の発注および支払いを追跡
BUDGET	会社の収支表
FORECASTS	将来の販売予測と現在の実績の記録

ユーザー

さまざまなアプリケーション・ユーザーが、異なる機能を実行するためにデータベースにアクセスします。

- 注文入力オペレータ
- 買掛処理オペレータ
- 売掛処理オペレータ
- 発送処理オペレータ
- セールス・マネージャ
- 財務アナリスト

アプリケーション・プロファイル

Eddie Bean アプリケーションの操作は 1 日中、ほぼ一貫したものです。つまり、終日一貫して注文入力、注文処理および発送を行います。これらの処理は、たとえば 1 時間単位で分離されているものではありません。

入力する注文の数は 1 日あたり約 500 件です。各注文のヘッダーは、その存続期間内に約 4 回更新されます。したがって、挿入の約 4 倍の更新があると予想されます。販売業務、会計業務、発送、注文状況の管理などを担当する多くの従業員が注文ヘッダーについて問い合せるので、多数の検索が発生します。

1 つの注文に、平均して 4 つの項目があります。ORDER ITEMS 表は更新されません。ある項目が削除され、別の項目が入力されるだけです。

ORDER_HEADER 表には、4 つの索引があります。その他の表には、それぞれ主キー索引だけがあります。

BUDGET 表および FORECASTS 表のアクティビティは、注文表より量が少なくなります。これらの表は、読込みの回数は頻繁ですが、変更されることはあまりありません。FORECASTS 表は BUDGET 表より頻繁に更新され、現実化した後は削除されます。

大量の削除処理は、バッチ・ジョブとして夜間に実行されます。したがって、このメンテナンス・アクティビティは、アプリケーションの通常機能の分析に組み込む必要はありません。

表へのアクセスの分析

最初に、データベース内の表に対する既存の（または予期している）アクセス・パターンを分析します。次に、表の分割方法と、アクセス・パターンに従って表をグループ化する方法を決定します。

- 表アクセス分析ワークシート
- ケース・スタディ : 表アクセスの分析

表アクセス分析ワークシート

表 13-2 のようなワークシートに、アクティビティ量の多いデータベース表をすべて示します。

表 13-2 表アクセス分析ワークシート

	1 日のアクセス量							
	読込みアクセス		書込みアクセス					
	選択		挿入		更新		削除	
表名	操作	I/O	操作	I/O	操作	I/O	操作	I/O

このワークシートに記入する前に、各タイプの操作量を見積ります。次に、操作に伴う読込みと書込み（I/O）の数を計算します。

操作量の見積り

1 つの表に対して実行される操作タイプごとに、予想される 1 日当りの標準使用量を示す数値を記入します。

注意： この分析で重要なのは、相対的な値、つまりアプリケーションの標準使用を示すおおよその数値です。アプリケーションがまだ存在していない場合でも、ユーザーのタイプを予測し、アクティビティの相対レベルを見積ることはできます。表のメンテナンス・アクティビティは、この分析には通常は関係ありません。

---

1 操作当りの I/O 数の計算

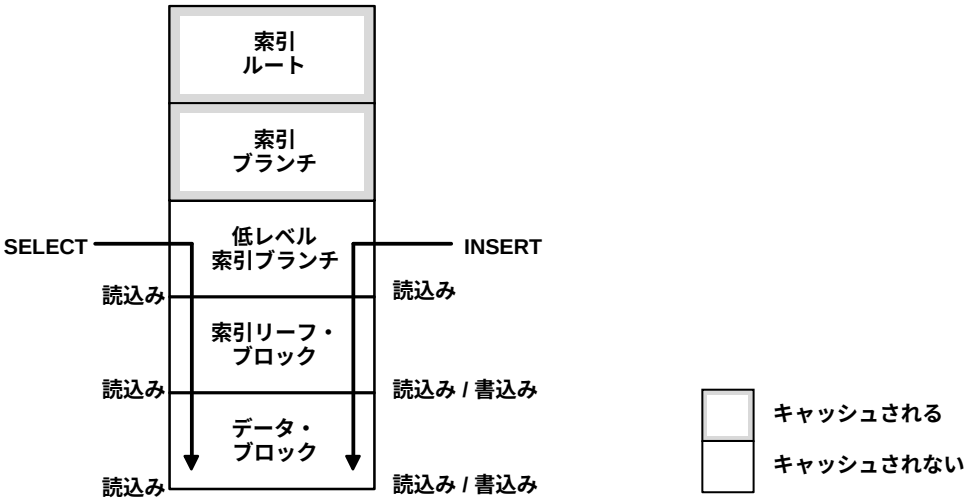
「操作」欄の値ごとに、生成される I/O の数を最悪の事例を使用して計算します。

SELECT 操作には読込みアクセスが含まれ、INSERT、UPDATE および DELETE 操作には読込みアクセスと書込みアクセスの両方が含まれます。これらの操作は、データ・ブロックだけでなく、関連のすべての索引ブロックにもアクセスします。

注意： 各操作で生成される I/O の数は、表のアクセス・パスと表のサイズに応じて、表ごとに変化します。また、表が備えている索引の数によっても異なります。たとえば、索引が小さいと、索引ブランチ・ブロックが 1 つしかないこともあります。

図 13-1 に、大きな表のデータへの読込みアクセスと書込みアクセスの例を示します。この表では、2 つの索引レベルはバッファにキャッシュされず、高レベルの索引のみが SGA にキャッシュされます。

図 13-1 SELECT 操作または INSERT 操作 1 回当りの I/O の数



この例では、主キーを使ってデータにアクセスするものとします。1 回の SELECT には、次の 3 回の I/O が必要です。

1. 第 1 下位レベルの索引ブロックを読み込むための I/O が 1 回
2. 第 2 下位レベルの索引ブロックを読み込むための I/O が 1 回
3. データ・ブロックを読み込むための I/O が 1 回

注意： すべてのルート・ブロックとブランチ・ブロックが SGA の中にあれば、SELECT で実行される I/O は 2 回のみ、つまり、索引リーフ・ブロックの読込みとデータ・ブロックの読込みのみになります。

INSERT 文または DELETE 文は、少なくとも 5 回の I/O が必要です。

1. データ・ブロックを読み込むための I/O が 1 回
2. データ・ブロックを書き込むための I/O が 1 回
3. 各索引について 3 回の I/O（索引エントリを読み込むために 2 回、索引を書き込むために 1 回）

この例の 1 つの UPDATE では、次の 7 つの I/O が必要です。

1. 第 1 下位レベルの索引ブロックを読み込むための I/O が 1 回
2. 第 2 下位レベルの索引ブロックを読み込むための I/O が 1 回

3. データ・ブロックを読み込むための I/O が 1 回
4. データ・ブロックを書き込むための I/O が 1 回
5. 第 1 下位レベルの索引ブロックを再び読み込むための I/O が 1 回
6. 第 2 下位レベルの索引ブロックを再び読み込むための I/O が 1 回
7. 索引ブロックを書き込むための I/O が 1 回

注意： INSERT または DELETE はすべての索引に作用しますが、UPDATE は 1 つの索引にしか作用しないこともあります。変更される索引キーの数をチェックしてください。

サンプル表の 1 操作当り I/O 数

このケース・スタディでは、索引数が表ごとに異なるので、1 操作あたりの I/O 数も表ごとに異なります。

表 13-3 に、ORDER_HEADER 表で生成される I/O の数を操作タイプ別に示します。この ORDER_HEADER 表には、4 つの索引があります。

表 13-3 操作当りの I/O 数 : ORDER_HEADER サンプル表

操作	SELECT	INSERT	UPDATE	DELETE
アクセスのタイプ	読込み	読込み / 書込み	読込み / 書込み	読込み / 書込み
I/O 数	3	14	7	14

注意： これらの数値は、使用するデータベースの各表の実際の索引数とアクセス・パスに基づいて調整する必要があります。

表 13-4 に、このケース・スタディのその他の表について生成される 1 操作当りの I/O 数を示します。各表には主キー索引のみがあるものとします。

表 13-4 1 操作当りの I/O 数 : その他のサンプル表

操作	SELECT	INSERT	UPDATE	DELETE
アクセスのタイプ	読込み	読込み / 書込み	読込み / 書込み	読込み / 書込み
I/O 数	3	5	7	5

この分析では、データが変更されると、ロールバック・セグメントが生成され、それに伴い追加の I/O が必要になるという事実は無視できます。これらの追加 I/O はインスタンス・ベースのものです。したがって、OPS アプリケーションでは問題になりません。

関連項目：『Oracle8*i* 概要』（索引について）

ケース・スタディ : 表アクセスの分析

表 13-5 に、ケース・スタディのアプリケーションの標準使用を表すおおまかな数値を示します。

表 13-5 ケース・スタディ : 表アクセス分析ワークシート

	1 日のアクセス量							
	読込みアクセス		書込みアクセス					
	選択		挿入		更新		削除	
表名	操作	I/O	操作	I/O	操作	I/O	操作	I/O
ORDER_HEADER	20,000	60,000	500	7,000	2,000	14,000	1,000	14,000
ORDER_ITEM	60,000	180,000	2,000	10,000	0	0	4,030	20,150
ORGANIZATIONS	40,000	120,000	10	50	100	700	0	0
BUDGET	300	900	1	5	2	14	0	0
FORECASTS	500	1,500	1	5	10	70	2	10
ACCOUNTS_PAYABLE	230	690	50	250	20	140	0	0

この表から次のことがわかります。

- 大量の書込みアクセスがあるのは、ORDER_HEADER 表および ORDER_ITEMS 表だけである。
- ORGANIZATIONS は、主として読込み専用表である。メンテナンスのためにある程度の数の INSERT、UPDATE および DELETE 操作は実行されるが、その標準の使用方法は SELECT だけである。

ユーザー別のトランザクション量の分析

最初に、データベース内の表に対する既存の（または予期している）アクセス・パターンを分析します。次に、表の分割方法と、アクセス・パターンに従って表をグループ化する方法を決定します。

- トランザクション量分析ワークシート
- ケース・スタディ：トランザクション量の分析

トランザクション量分析ワークシート

大量の書込みアクセスが必要な個々の表について、ユーザーのタイプ別に 1 日あたりのトランザクション量を分析します。

注意： 読込み専用表の場合は、ユーザー・タイプ別のトランザクション量を分析する必要はありません。

表 13-6 のようなワークシートを使用します。

表 13-6 トランザクション量分析ワークシート

表名：									
ユーザー・タイプ	**ユーザー数**	**1 日のトランザクション量**							
		読込みアクセス		**書込みアクセス**					
		選択		**挿入**		**更新**		**削除**	
		操作	**I/O**	**操作**	**I/O**	**操作**	**I/O**	**操作**	**I/O**

最初に、ユーザー・タイプ別のトランザクション量を見積り、次に、それに伴って必要になる I/O の数を計算します。

ケース・スタディ : トランザクション量の分析

次の表は、このケース・スタディで使用する、大量の書込みアクセスのある 3 つの表、つまり、ORDER_HEADER、ORDER_ITEMS および ACCOUNTS_PAYABLE のトランザクション量の分析を示しています。

ORDER_HEADER 表

表 13-7 に、このケース・スタディの ORDER_HEADER 表のおおまかな数値を示します。

表 13-7 ケース・スタディ : トランザクション量の分析 : ORDER_HEADER 表

表名 :ORDER_HEADER									
		1 日のトランザクション量							
		読込みアクセス		**書込みアクセス**					
		選択		**挿入**		**更新**		**削除**	
ユーザー・タイプ	**ユーザー数**	**操作**	**I/O**	**操作**	**I/O**	**操作**	**I/O**	**操作**	**I/O**
注文入力オペレータ	25	5,000	15,000	500	7,000	0	0	0	0
買掛処理オペレータ	5	0	0	0	0	0	0	0	0
売掛処理オペレータ	5	6,000	18,000	0	0	1,000	7,000	0	0
発送処理オペレータ	4	4,000	12,000	0	0	1,000	7,000	0	0
セールス・マネージャ	2	3,000	9,000	0	0	0	0	0	0
財務アナリスト	2	2,000	6,000	0	0	0	0	0	0

この表から次のことがわかります。

- 注文入力オペレータは、この表に対するすべての挿入操作を実行する。
- 売掛処理オペレータおよび発送処理オペレータは、すべての更新操作を実行する。
- セールス・マネージャと財務アナリストは、選択操作のみを実行する。
- 買掛処理オペレータは、この表を使用しない。

削除はメンテナンス操作として実行されるので、この分析では考慮する必要はありません。さらに、セールス・マネージャは通常は当月のデータにアクセスするのに対して、財務アナリストは過去の月のデータに多くアクセスすることがわかります。

ORDER_ITEMS 表

表 13-8 に、このケース・スタディの ORDER_ITEMS 表のおおまかな数値を示します。

表 13-8 ケース・スタディ : トランザクション量の分析 : ORDER_ITEMS 表

表名 :ORDER_ITEMS									
		1 日のトランザクション量							
		読込みアクセス		**書込みアクセス**					
		選択		**挿入**		**更新**		**削除**	
ユーザー・タイプ	**ユーザー数**	**操作**	**I/O**	**操作**	**I/O**	**操作**	**I/O**	**操作**	**I/O**
注文入力オペレータ	25	15,000	45,000	2,000	10,000	0	0	20	100
買掛処理オペレータ	5	0	0	0	0	0	0	0	0
売掛処理オペレータ	5	18,000	54,000	0	0	0	0	10	50
発送処理オペレータ	4	12,000	36,000	0	0	0	0	0	0
セールス・マネージャ	2	9,000	27,000	0	0	0	0	0	0
財務アナリスト	2	6,000	18,000	0	0	0	0	0	0

この表から次のことがわかります。

- 注文入力オペレータは、この表に対するすべての挿入操作を実行する。
- 更新はほとんど実行されない。
- 売掛処理オペレータ、発送処理オペレータ、セールス・マネージャおよび財務アナリストは、表に対して大量の選択操作を実行する。
- 買掛処理オペレータは、この表を使用しない。

ORDER_HEADER 表では ORDER_ITEMS 表より多くの書込み操作が必要です。それは、ORDER_HEADER 表では、使用可能な製品のリストの場合より、住所変更などのステータスの変更が必要になることが多いからです。新規項目はジャーナル・エントリとして示されるので、ORDER_ITEMS 表はほとんど更新されません。

ACCOUNTS_PAYABLE 表

表 13-9 に、このケース・スタディの ACCOUNTS_PAYABLE 表のおおまかな数値を示します。この表は特に書込みアクセスが多いわけではありませんが、買掛処理オペレータが実行する主要操作が含まれているので、分析の対象にしてあります。

表 13-9 ケース・スタディ : トランザクション量の分析 : ACCOUNTS_PAYABLE 表

表名 :ACCOUNTS_PAYABLE									
		1 日のトランザクション量							
		読込みアクセス		**書込みアクセス**					
		選択		**挿入**		**更新**		**削除**	
ユーザー・タイプ	**ユーザー数**	**操作**	**I/O**	**操作**	**I/O**	**操作**	**I/O**	**操作**	**I/O**
注文入力オペレータ	25	0	0	0	0	0	0	0	0
買掛処理オペレータ	5	200	600	50	250	20	140	0	0
売掛処理オペレータ	5	0	0	0	0	0	0	0	0
発送処理オペレータ	4	0	0	0	0	0	0	0	0
セールス・マネージャ	2	0	0	0	0	0	0	0	0
財務アナリスト	2	30	90	0	0	0	0	0	0

この表から次のことがわかります。

- 買掛処理オペレータは、毎日約 50 の購入注文を納入業者に送る。これらのオペレータは、この表のデータを変更する唯一のユーザーである。
- 財務アナリストは、情報をときどき調べる。

削除はメンテナンス操作として実行されるので、この分析では考慮する必要はありません。

ユーザーおよびデータの分割

ここでの目標は、アプリケーションを複数のインスタンス間で分割することです。これには、インスタンス間でのユーザーの分割と、特定タイプのユーザーだけが書き込む必要のあるデータの分割があります。これにより、システムで発生する競合の量が最小化されます。この項では、次の項目について説明します。

- ケース・スタディ : 初期の分割計画
- ケース・スタディ : 詳細な分割計画

ケース・スタディ : 初期の分割計画

このケース・スタディでは、ORDER_HEADER 表および ORDER_ITEMS 表に対して大量の挿入アクティビティを実行する多数の注文入力オペレータを複数のマシン間に分割しない例を示します。注文入力オペレータは、頻繁に使用するこれらの 2 つの表を使用し、1 つのノードを集中的に使用するようにしてください。したがって、図 13-2 に示すように、注文入力オペレータ用に 1 ノードを設定し、他のすべてのユーザー用に 1 ノードを設定することをお薦めします。

図 13-2 ケース・スタディ : ユーザーおよびデータの分割

このシステムでは、ノード間のバランスがよくとれています。財務アナリストが実行するデータベース集中型のレポート作成には大量のシステム・リソースが必要なのに対し、注文入力オペレータが実行するトランザクションは比較的簡単です。

システム全体にわたってユーザー数を調整して負荷を均衡化することは、一般的には有益ですが、必ずしも重要ではありません。ロード・バランシングは、チューニングのための方法としては、競合の軽減より優先順位が低いものです。

ケース・スタディ : 詳細な分割計画

このケース・スタディでは、買掛データを書き込むのが買掛処理オペレータのみであることも明らかです。したがって、図 13-3 に示すように、このデータとユーザーの集合を別のインスタンスに分割すると効果的です。

図 13-3 ケース・スタディ : ユーザーおよびデータの分割 : 設計オプション 1

データの特定部分への書込みアクセスを必要とするユーザー全員が 1 つのノードに集中すると、PCM ロックはすべてそのノードに常駐します。したがって、ロック所有権をインスタンス間で移動させる必要がなくなります。

この分析に基づき、最初に次の 2 つの設計オプションを得ることができます。

設計オプション 1

上の図のように、すべての注文入力オペレータを 1 つのインスタンスにまとめると、表の排他 PCM ロックのための競合が最小限に抑えることができます。これにより、セールス・マネージャおよび財務アナリストは、いつでも詳細な情報を入手できます。セールス・マネージャや財務アナリストにとっては特に履歴データが必要なので、最新のレコードに対する競合は少ないはずです。

設計オプション 2

上の方法のかわりに、ORDER_ITEM および ORDER_HEADER をそれぞれ一時表として実装できます。この表は、新規注文情報の記録のみに使用します。前夜のうちに、すべての問合せの対象となる主表に変更を組み込むようにします。このソリューションがうまく機能するのは、財務分析にとって最新のデータがあまり重要でない場合です。これが該当するのは、財務分析の主要な関心が履歴データの検索にあるときのみです。財務アナリストに最新のデータが必要な場合は、このソリューションは適切ではありません。

図 13-4 ケース・スタディ : ユーザーおよびデータの分割 : 設計オプション 2

索引の分割

システム内の複数のノードから同一索引への挿入を実行する場合は、索引の分割を検討する必要があります。この状況では、それぞれのインスタンスが、索引内のそれぞれ異なるポイントに挿入を実行するように設定する必要があります。

注意： Eddie Bean のケース・スタディでは、アプリケーションとデータの使用方法は分割してあるので、この問題は起こりません。

関連項目： 17-8 ページの「索引用の空きリストの作成」（索引の競合を避けるための空きリスト、空きリスト・グループおよび順序番号の使用について）、19-4 ページの「アプリケーション内のロック競合位置の特定」（競合ポイントとなる索引について）、『Oracle8*i* 概要』（空きリスト・グループを使用しないで表およびインスタンスを物理的に分割する方法について）

ハッシュ・ロックとファイン・グレイン・ロックの実装

多くのアプリケーションについて、DBA は特定のデータベース・ファイルにハッシュ・ロックまたはファイン・グレイン・ロックのどちらを使用するか決定する必要があります。

ハッシュ・ロックを使用した場合の最悪の例を使用して設計する必要があります。その後、設計フェーズまたは監視フェーズでロックの数が多すぎることがわかったり、false ping の疑いがある場合に、ファイン・グレイン・ロックを試してみるようにします。

分析はデータベース・レベルで開始します。表 13-10 のようなワークシートを使用します。

表 13-10 ワークシート：ハッシュ・ロックまたはファイン・グレイン・ロック用のデータベース分析

ブロック・クラス	関連パラメータ	ファイン・グレイン・ロックかハッシュ・ロックか

次に、表 13-11 のようなワークシートにファイルおよびデータベース・オブジェクトをリストします。各ファイルについて、どのロック・モードを使用するかを決定してください。

表 13-11 ワークシート : ハッシュ・ロックまたはファイン・グレイン・ロックを使用すべきタイミング

ファイル名	含まれるオブジェクト	ファイン・グレイン・ロックかハッシュ・ロックか

関連項目 : 9-20 ページの「異なるファイルへのファイン・グレイン・ロックおよびハッシュ・ロックの適用」

設計のインプリメントおよびチューニング

ここまでは概算値を使用して分析してきました。設計の最終仕上げをするには、アプリケーションをプロトタイプ化するか、または実際に実装し、実行する必要があります。実際のシステムを観察することによって、その後のシステムの調整ができます。

次の手法を試してください。

- ping されているブロックを識別し、競合がどこに存在するかを判断する。
- ping および false ping の量を減らすには、ユーザーを他のインスタンスに移動させることを検討する。
- 高レベルの false ping が検出された場合は、各ファイルに多数のロックを設定することで、ロックの細分性の増加を検討する。
- 挿入時に ping が発生する場合は、空きリストを調整するか、または複数の順序番号ジェネレータを使用して、挿入が索引の別の場所で実行されるようにする。

関連項目 : 『Oracle8*i* チューニング』の第 19 章「チューニングによるパフォーマンスの最適化」

14

複数インスタンス用のデータベースおよびオブジェクトの作成

この章では、次の項目について説明します。

- 複数インスタンス環境用のデータベースの作成
- 複数インスタンスをサポートするデータベース・オブジェクトの作成
- CREATE DATABASE オプションの値の変更

複数インスタンス環境用のデータベースの作成

この項では、OPS 固有のデータベース作成作業について説明します。

- 作業の概略
- データベース作成用の初期化パラメータの設定
- データベースの作成および起動
- CREATE DATABASE オプションの設定

作業の概略

OPS 固有のデータベース作成作業を要約すると、次のようになります。

1. ログ・アーカイブを含むすべてのインスタンスの初期化パラメータを設定します。
2. パラレル・サーバーが使用禁止になった状態で、OPS に固有の MAXINSTANCES およびその他の重要なオプションを設定して、CREATE DATABASE 文を入力します。
3. 各ノードのロールバック・セグメントを作成します。
4. データベースを一度ディスマウントし、初期設定ファイルで PARALLEL_SERVER=TRUE パラメータを設定してから再マウントします。再マウントした後、パラレル・サーバーを起動します。

関連項目：『Oracle8*i* 管理者ガイド』の「Oracle データベースの作成」

データベース作成用の初期化パラメータの設定

データベースの作成上重要な、または特定のデータベース操作に影響を与える特定の初期化パラメータは、OPS のすべてのインスタンスで同じ値にする必要があります。複数インスタンス環境のデータベースを作成する前に、これらのパラメータがすべてのインスタンスで同じ値に設定されていることを確認してください。

ARCHIVELOG モードの使用方法

データベースの作成時に ARCH プロセスを使用可能にするには、初期化パラメータ LOG_ARCHIVE_START を TRUE に設定します。その後、データベースを作成するインスタンスを起動する前に、ALTER DATABASE 文を使用してモードを ARCHIVELOG に変更します。

または、NOARCHIVELOG モードでデータベースを作成することによってオーバーヘッドを軽減できます。これはデフォルトの設定です。その後、モードを ARHIVELOG に変更します。

STARTUP コマンドを使用してデータベースのアーカイブ・モードを変更することはできません。データベースを作成した後に、次の Server Manager コマンドを使用してアーカイブ・モードを変更し、パラレル・サーバーが使用可能な状態でデータベースを再オープンします。

```
ALTER DATABASE CLOSE;
ALTER DATABASE ARCHIVELOG;
SHUTDOWN;
STARTUP;
```

関連項目： 21-2 ページの「REDO ログ・ファイルのアーカイブ」および 18-10 ページの「すべてのインスタンスで同じ値にする必要があるパラメータ」

データベースの作成および起動

データベースを作成するには、標準の作成手順を使用します。

注意： CREATE DATABASE 文は、新規作成したデータベースをマウントしてオープンしますが、パラレル・サーバーは使用禁止のままになっています。データベースをクローズして一度ディスマウントし、パラレル・サーバーが使用可能な状態で再マウントする必要があります。

---

1. Server Manager を起動します。
2. SYSDBA 権限を使用して接続します。

3. NOMOUNT オプションを使用して、インスタンスを起動します。
4. CREATE DATABASE 文を発行します。
5. 必要に応じて、追加のロールバック・セグメントおよびスレッドを作成します。
6. データベースをクローズし、ディスマウントします。

   ```
   SHUTDOWN
   ```

7. 初期設定ファイルを更新して、初期設定ファイルが適切なロールバック・セグメントおよびスレッドを指しているかを確認します。
8. パラレル・サーバーが使用可能であることを確認します。
9. データベースを再マウントします。

   STARTUP [OPEN *databasename*]

関連項目： 18-12 ページの「インスタンスの起動」

CREATE DATABASE オプションの設定

この項では、OPS 固有の CREATE DATABASE オプションについて説明します。

MAXINSTANCES の設定

CREATE DATABASE の MAXINSTANCES オプションは、データベースに同時にアクセスできるインスタンスの数を制限します。MAXINSTANCES のデフォルトは、使用しているそれぞれのオペレーティング・システムの最大値です。ほとんどのシステムでは、デフォルトは 2 です。

OPS では、MAXINSTANCES には、同時実行を予定しているインスタンスの最大数より大きい値を設定します。これにより、インスタンス A が失敗しインスタンス B を使って回復する場合でも、インスタンス A が完全に回復する前に、インスタンス C を起動できます。

MAXLOGFILES および MAXLOGMEMBERS の設定

CREATE DATABASE の MAXLOGFILES オプションには、データベース用に作成できる REDO ログ・グループの最大数を指定し、MAXLOGMEMBERS オプションには、1 グループあたりのメンバーまたはコピーの最大数を指定します。

OPS では、MAXLOGFILES には、可能なスレッドの最大数に 1 スレッドあたりの予想最大グループ数を掛けた値を設定します。

MAXLOGHISTORY の設定

CREATE DATABASE の MAXLOGHISTORY オプションには、制御ファイルのログ履歴に記録できる REDO ログ・ファイルの最大数を指定します。ログ履歴は、OPS の自動メディア回復に使用されます。

OPS では、MAXLOGHISTORY を大きな値（たとえば 1000）に設定する必要があります。制御ファイルに格納できるのは、MAXLOGHISTORY で指定した数の REDO ログ・ファイルについての情報だけです。ログ履歴がこの制限を超えると、古い履歴エントリが上書きされます。MAXLOGHISTORY のデフォルトは 0（ゼロ）で、ログ履歴が使用禁止にされます。

MAXDATAFILES の設定

MAXDATAFILES オプションは汎用ですが、OPS で使用するデータ・ファイルおよびログ・ファイルの数は、通常では標準システムより多くなります。プラットフォームによっては、このオプションのデフォルト値では小さすぎる場合があります。

> **関連項目：**『Oracle8*i* SQL リファレンス』（SQL の CREATE DATABASE 文および ALTER DATABASE 文の詳細）、オペレーティング・システム固有の Oracle マニュアル（CREATE DATABASE オプションのデフォルト値について）、「REDO ログ・ファイル」6-3 ページ（REDO ログ・グループおよびメンバーの詳細）、「制御ファイルの REDO ログ履歴」21-6 ページ（MAXLOGHISTORY の詳細）

複数インスタンスをサポートするデータベース・オブジェクトの作成

新規データベースを OPS で使用するための準備作業として、次の追加データベース・オブジェクトを作成して構成します。

- 追加のロールバック・セグメントの作成
- OPS のオンライン REDO ログの構成
- 追加データファイル用のロックの提供

追加のロールバック・セグメントの作成

パラレル・サーバーの各インスタンスについて、少なくとも 1 つのロールバック・セグメントを作成する必要があります。競合を避けるには、これらのロールバック・セグメントを SYSTEM 表領域以外の表領域の中に作成します。

他の表領域内にロールバック・セグメントを作成するには、その前に、SYSTEM 表領域内に追加のロールバック・セグメントを 1 つ作成してオンラインにする必要があります。データベースを作成するインスタンスでは、この追加ロールバック・セグメントおよび新規の表領

域を作成できますが、追加ロールバック・セグメントをオンラインにしない限り、非SYSTEM 表領域にデータベース・オブジェクトを作成することはできません。

プライベート・ロールバック・セグメントの使用

プライベート・ロールバック・セグメントをインスタンスに割り当てる手順は、次のとおりです。

1. SQL 文の CREATE ROLLBACK SEGMENT をキーワード PUBLIC を指定せずに使用し、ロールバック・セグメントを作成します。必要に応じて、ロールバック・セグメントを作成する前に、そのセグメント用の表領域を作成できます。
2. ロールバック・セグメントをパラメータ値として指定することにより、ロールバック・セグメントをインスタンスのパラメータ・ファイルの中で指定します。これで、ロールバック・セグメントがそのインスタンス用に確保されます。
3. ALTER ROLLBACK SEGMENT を使用してロールバック・セグメントをオンライン化します。このインスタンスを再起動して、確保されたロールバック・セグメントを使用することもできます。

プライベート・ロールバック・セグメントは、1 つのインスタンスだけに関連付けられるように、1 つのパラメータ・ファイルだけに指定する必要があります。インスタンスが、他のインスタンスがすでに取得しているプライベート・ロールバック・セグメントを取得しようとすると、Oracle がメッセージを表示し、そのインスタンスの起動を禁止します。

パブリック・ロールバック・セグメントの使用

パブリック・ロールバック・セグメントはどのインスタンスでも作成でき、起動時に要求できるものです。要求されたロールバック・セグメントは、要求元のインスタンスがシャット・ダウンし、他のインスタンスが使用できるようにそのロールバック・セグメントを解放するまで、要求元のインスタンスだけが使用します。

パブリック・ロールバック・セグメントを作成するには、SQL 文の CREATE PUBLIC ROLLBACK SEGMENT を使用します。

パブリック・ロールバック・セグメントはそれを必要とするすべてのインスタンスが使用可能なことが前提になっているので、通常、特定のインスタンス用のパラメータ・ファイルには、パブリック・ロールバック・セグメントを指定しません。しかし、別のインスタンスがすでに使用中でない場合は、パブリック・ロールバック・セグメントの名前を ROLLBACK_SEGMENTS パラメータの値として指定できます。

パブリック・ロールバック・セグメントは、データ・ディクショナリ・ビュー DBA_ROLLBACK_SEGS の中で、所有者 PUBLIC を持つものとして識別されます。

パラメータ・ファイルで ROLLBACK_SEGMENTS 初期化パラメータを指定しないと、インスタンスは、デフォルトによりパブリック・ロールバック・セグメントを使用します。

パブリック・ロールバック・セグメントを必要とするインスタンスが起動し、それを取得すると、そのパブリック・ロールバック・セグメントはオンラインになります。ただし、パブ

リック・ロールバック・セグメントを使用するインスタンスを起動しても、そのインスタンスが使用可能なパブリック・ロールバック・セグメントをすべて取得しない場合は、特定のパブリック・ロールバック・セグメントがオンラインになるという保証はありません。取得されたパブリック・ロールバック・セグメントは、その取得元のインスタンスにより排他的に使用されます。

プライベートでもパブリックでも、ロールバック・セグメントのオンライン化、オフライン化、作成および削除は、OPS が使用可能でも使用禁止でも同じです。

プライベート・ロールバック・セグメントは、オンライン化されるかまたはそれを所有するインスタンスが再起動されるまで、オフラインの状態になっています。パブリック・ロールバック・セグメントは、特定のインスタンス用にオンライン化されるか、またはパブリック・ロールバック・セグメントを必要とするインスタンスが起動され、それを取得するまで、オフラインの状態になっています。

パブリック・ロールバック・セグメントを削除して再作成せずに、オフラインにしておく必要がある場合には、パブリック・ロールバック・セグメントを必要とする他のインスタンスが起動しないようにする必要があります。

ロールバック・セグメントの監視

現行インスタンスが使用するロールバック・セグメントのステータスの情報を表示するには、Server Manager コマンド MONITOR ROLLBACK を使用します。

このコマンドを使用するかわりに、動的パフォーマンス・ビュー V$ROLLNAME および V$ROLLSTAT で問合せを行い、現行インスタンスのロールバック・セグメントについての情報を得ることもできます。

現行インスタンスを変更するには、MONITOR コマンドを使用する前または V$ ビューに問い合せる前に、Server Manager コマンド CONNECT *@instance-path* を使用します。リモート・ノードのインスタンスについて CONNECT コマンドを使用するには、Net8 をインストールしておく必要があります。

データ・ディクショナリ・ビュー DBA_ROLLBACK_SEGS および DBA_SEGMENTS を問い合せて、データベース内のすべてのロールバック・セグメントの現行ステータスについての情報を得ることもできます。

たとえば、現行ロールバック・セグメントのリストを表示するには、次の文を使用して DBA_ROLLBACK_SEGS を問い合せます。

```
SELECT segment_name, segment_id, owner, status
   FROM dba_rollback_segs
```

この問合せによって、次の例に示すように、ロールバック・セグメントの名前、ID 番号、所有者およびセグメントが使用中かどうかが表示されます。

```
SEGMENT_NAME             SEGMENT_ID     OWNER     STATUS
------------------------ ----------     ------    ------------
SYSTEM                            0     SYS       ONLINE
PUBLIC_RS                         1     PUBLIC    ONLINE
USERS1_RS                         2     SYS       ONLINE
USERS2_RS                         3     SYS       OFFLINE
USERS3_RS                         4     SYS       ONLINE
USERS4_RS                         5     SYS       ONLINE
PUBLIC2_RS                        6     PUBLIC    OFFLINE
```

この例では、ユーザー SYS が所有するものとして識別されたロールバック・セグメントがプライベート・ロールバック・セグメントで、ユーザー PUBLIC が所有するものとして識別されたロールバック・セグメントがパブリック・ロールバック・セグメントです。

ビュー DBA_ROLLBACK_SEGS には、ロールバック・セグメントを含む表領域およびセグメント・ヘッダーを含むデータ・ファイル、エクステント・サイズについての情報も含まれています（図には示されていません）。ビュー DBA_SEGMENTS には、各ロールバック・セグメントのエクステント数およびセグメント・サイズについての追加情報が含まれています。

関連項目：『Oracle8*i* 管理者ガイド』（ロールバック・セグメントおよびデータベース接続について）、『Oracle8*i* Net8 管理者ガイド』およびシステム固有の Oracle マニュアル（*instance-path* の接続文字列の形式について）、『Oracle8*i* リファレンス・マニュアル』（DBA_ROLLBACK_SEGS、DBA_SEGMENTS およびその他の動的パフォーマンス・ビューについて）

OPS のオンライン REDO ログの構成

各データベース・インスタンスには、専用のオンライン REDO ログ・グループからなる、専用のオンライン REDO の " スレッド " があります。OPS の実行中は、2 つ以上のインスタンスが単一のデータベースに同時にアクセスし、各インスタンスはそれぞれ専用のスレッドを持っている必要があります。この項では、OPS を使用する場合の複数インスタンスのオンライン REDO スレッドの構成方法について説明します。

各スレッドは、少なくとも 2 つの REDO ログ・ファイル（または多重グループ）を持つように作成し、インスタンスが使用する前にスレッドを使用可能にする必要があります。

CREATE DATABASE 文は、スレッド番号 1 をパブリック・スレッドとして作成し、自動的にそれを使用可能にします。後続のスレッドを作成して使用可能にするには、ALTER DATABASE 文を使用する必要があります。

スレッドの作成

スレッドはパブリックまたはプライベートにすることができます。初期化パラメータ THREAD は、一意のスレッド番号をインスタンスに割り当てます。THREAD がデフォルトの 0(ゼロ) であれば、そのインスタンスは使用可能なパブリック・スレッドを取得します。

作成された各スレッドは、2 つ以上の REDO ログ・ファイルまたは多重グループで構成されている必要があります。また、インスタンスが使用できるように、各スレッドを使用可能にしておく必要があります。

CREATE DATABASE 文は、スレッド番号 1 をパブリック・スレッドとして作成し、自動的にそれを使用可能にします。後続のスレッドは、ALTER DATABASE 文を使用して作成し使用可能にします。たとえば、次の文を使用すると、6-4 ページの図 6-1 に示すように、スレッド 2 はそれぞれ 3 つのメンバーからなる 2 つのグループを持つスレッドとして作成されます。

```
ALTER DATABASE ADD LOGFILE THREAD 2
 GROUP 4 (disk1_file4, disk2_file4, disk3_file4) SIZE 1M REUSE
 GROUP 5 (disk1_file5, disk2_file5, disk3_file5) SIZE 1M REUSE;
ALTER DATABASE ENABLE PUBLIC THREAD 2;
```

スレッドを使用可能にするときにキーワード PUBLIC を指定しないと、そのスレッドはデフォルトでは取得できないプライベート・スレッドになります。ALTER DATABASE ADD LOGFILE 文に指定できるスレッド番号は 1 つだけであり、デフォルトにより現行インスタンスのスレッド番号が選択された場合は、THREAD 句を指定する必要があります。

スレッドの使用禁止

ALTER DATABASE DISABLE THREAD 文を使用すると、パブリック・スレッドまたはプライベート・スレッドを使用禁止にできます。スレッドを使用中のインスタンスがすでにデータベースをマウントしている場合は、スレッドは使用禁止にできません。スレッドをパブリックからプライベートに、またはプライベートからパブリックに変更するには、そのスレッドを使用禁止にしてから、再び使用可能にする必要があります。インスタンスはそれが所有するスレッドを使用禁止にできません。スレッドを使用禁止または使用可能にするときは、データベースをオープンしておく必要があります。

スレッドを使用禁止にすると、Oracle はその現行 REDO ログ・ファイルをアーカイブする必要があるファイルとしてマークします。そのファイルを削除するには、最初に手動でそのファイルをアーカイブすることが必要な場合があります。

スレッドが使用可能な状態でエラーまたは障害が起こると、現行ログ・ファイルの集合を持っているが使用可能になっていないスレッドができます。このようなログ・ファイルの削除またはアーカイブはできません。この場合、スレッドがすでに使用禁止になっていても、一度そのスレッドを使用禁止にして、再び使用可能にする必要があります。

ログのモードの設定

REDO ログを使用するモード、ARCHIVELOG または NOARCHIVELOG は、データベースの作成時に設定します。めったに変更する必要はありませんが、アーカイブ・モードは、SQL 文 ALTER DATABASE を使用して変更できます。アーカイブを使用可能にしてあるときは、オンライン REDO ログ・ファイルは、アーカイブされるまで再使用できません。アーカイブ・モードを切り替えるには、OPS が使用禁止の状態でデータベースをマウントする必要がありますが、データベースがオープン状態であってはなりません。

REDO ログ・モードは、個々のインスタンスでなくデータベースに関連付けられます。ほとんどの場合、REDO ログを ARCHIVELOG モードで使用しているときは、すべてのインスタンスで同じアーカイブ方法、つまり、自動または手動のいずれか一方を使用するようにします。

REDO ログの変更

OPS が使用可能または使用禁止のどちらであっても、データベースがマウントされた状態であれば、ログ・ファイルまたはログ・ファイル・メンバーの追加、削除または改名などを行って REDO ログの構成を変更できます。唯一の制限は、スレッドが現在使用中のログ・ファイルまたはログ・ファイル・メンバーの削除または改名はできないことです。また、スレッド内のログ・グループの数が 2 つより少なくなる場合は、ログ・ファイルを削除できません。

あらゆるインスタンスを使い、他のあらゆるインスタンスのあらゆるグループの REDO ログ・ファイルまたはメンバーを追加または改名できます。1 インスタンスについて 3 つ以上のグループが残っている限り、そのインスタンスの REDO ログ・グループを他の任意のインスタンスが削除できます。REDO ログ・ファイルおよびログ・メンバーに対する変更は、次のログ切替えの時点で有効になります。

関連項目： 21-2 ページの「REDO ログ・ファイルのアーカイブ」

追加データファイル用のロックの提供

OPS の実行中にデータ・ファイルを追加する場合は、新規ファイルを受け持つ十分なロックがあるかどうかを評価する必要があります。

追加したデータ・ファイルが使用する未割当てのロックは、GC_FILES_TO_LOCKS に設定した値から作成されたロックです。残りのロックが不十分で新規ファイルの保護および競合の防止をできない場合は、GC* パラメータを調整してロックの数を増やす必要があります。このような調整を行わないと、パフォーマンスに問題が起きることがあります。

読込み専用データベースの場合は、多数の新規データ・ファイルを追加したときでも、ロックを追加する必要はありません。しかし、頻繁に挿入操作をするデータベースでは、追加ロックが必要な場合があります。

ロックの追加が必要な場合は、次のことを行います。

1. システムをシャットダウンします。
2. 初期化パラメータ GC_FILES_TO_LOCKS を修正して、追加データファイルに十分なロックを提供できるようにします。
3. システムを再起動します。

CREATE DATABASE オプションの値の変更

CREATE CONTROLFILE 文を使用すると、既存のデータベースについて次のデータベース・パラメータの値を変更できます。

- MAXINSTANCES
- MAXLOGFILES
- MAXLOGMEMBERS
- MAXLOGHISTORY
- MAXDATAFILES

関連項目：『Oracle8*i* SQL リファレンス』（CREATE CONTROLFILE 文および ALTER DATABASE BACKUP CONTROLFILE TO TRACE 文の詳細）

15

PCM インスタンス・ロックの割当て

この章では、OPS インスタンスのデータ・ファイルに PCM ロックを割り当てるために必要な init.ora パラメータの設定について説明します。

- PCM ロックの使用およびメンテナンスの計画
- GC_FILES_TO_LOCKS の設定 : 各データ・ファイルの PCM ロック数
- GC_FILES_TO_LOCKS の設定のヒント
- その他の GC_* パラメータの設定
- PCM ロックのチューニング

注意 : PCM ロックをチューニングするだけでは、アプリケーションを適切に拡張できないことがあります。詳細は、第 12 章「アプリケーションの分析」および第 13 章「パラレル・サーバー用のデータベース設計」を参照してください。

関連項目 : 第 9 章「パラレル・キャッシュ管理インスタンス・ロック」（PCM ロックおよび GC_* パラメータの概念について）および『Oracle8*i* リファレンス・マニュアル』（OPS でのロックの割当てに使用する初期化パラメータについて）

PCM ロックの使用およびメンテナンスの計画

この項では、PCM ロックの使用およびメンテナンスの計画について説明します。この項では、次の項目について説明します。

- インスタンス・ロックの計画およびメンテナンス
- PCM ロックの割当て
- データ・ファイルおよびデータ・ブロックの検査
- ワークシートの使用による必要な PCM ロックの分析
- データ・ブロックへの固定 PCM ロックのマッピング
- インスタンス間での PCM ロックの分割

インスタンス・ロックの計画およびメンテナンス

IDLM では、割り当てることができるロックの数に制限があります。したがって、アプリケーションに必要なロックの数を分析し計画を立てる必要があります。また、ロックおよびリソースに必要なメモリーを調べる必要があります。次の影響を考慮してロック数を計画してください。

- 使用するロック数が IDLM 機能の構成を超えると、Oracle でエラー・メッセージが表示されてインスタンスがシャットダウンする。
- Oracle GC_* または LM_* 初期化パラメータを変更して多数のロックを指定すると、SGA の使用メモリー量または使用可能メモリー量に影響が出る。
- インスタンスの数は、システム全体で必要なメモリー量およびロックの数にも影響する。

PCM ロックの割当て

ロックを割り当てるときに重要なのは、INSERT、UPDATE および DELETE コマンドを使用してデータを変更する頻度を分析することです。これを分析すると、読込み専用にするか読込み / 書込みにするかに基づいて、ファイルに入れるオブジェクトをどのようにグループ化するかが決まります。最後に、決定したグループ化に基づいて、ロックを割り当てます。一般に次のガイドラインに従います。

- 読込み専用ファイルには、少数のロックを割り当てる。
- 読込み / 書込みの集中するファイルには多数のロックを割り当てる。
- 表領域全体が読込み専用の場合は、それに 1 つのロックを割り当てるだけでよい。表領域にロックをまったく割り当てないと、システムは予備ロックを使用します。これは、予備ロックを取得するために他の表領域と競い合うことになるため、競合を生じる可能性があります。その結果、不要な強制読込み / 書込みが生じます。

オブジェクトのタイプ（索引または表）を区別するのではなく、オブジェクトに対して実行する操作を区別することが重要です。この操作によって、必要なロックの量が規定されます。

関連項目： 第 12 章「アプリケーションの分析」

データ・ファイルおよびデータ・ブロックの検査

ロックは、次のように各種レベルで割り当てます。

- すべてのデータ・ファイルに割り当てる PCM ロックの最大数を指定する。
- 各データ・ファイル内のブロックに割り当てるロックの数を指定する。
- 対象ファイル内の特定のデータ・ブロック・クラスを受け持つ特定のロックを指定する。

最初に、データ・ファイルとそれに含まれるブロックについて調べます。

ファイル ID、表領域名、ブロックの数の判別方法

すべてのデータベースについてファイル ID およびファイル名、表領域名、ブロックの数を判別するには、次のコマンドを使用します。

```
SELECT FILE_NAME, FILE_ID, TABLESPACE_NAME, BLOCKS
FROM DBA_DATA_FILES;
```

次の例に示すような結果が表示されます。

```
FILE_NAME               FILE_ID     TABLESPACE_NAME     BLOCKS
-----------------------------------------------------------------
/v7/data/data01.dbf     1           SYSTEM               200
/v7/data/data02.dbf     2           ROLLBACK            1600
. . .
```

必要なロック数

次の手法により、特定のユーザーに必要なロック数を見積ります。

- データおよびアプリケーションの性質を考慮する。

  使用頻度が高く、同時に更新されるデータ・ファイルには、多数のロックが必要である。ただし、問合せ専用のアプリケーションは、多数のロックを必要としない。データ・ファイルに 1 つのロックで十分である。

- 多くのインスタンスが同時に変更するファイルには、多数のロックを割り当てる。

  これにより、ロックの競合が減り、I/O アクティビティが最小限になり、ファイル内のデータへアクセスする可能性が増加する。

- 複数のインスタンスが同時にアクセスする必要のないファイルには、割り当てるロックの数を少なくする。

  これにより、ロック管理に不要なオーバーヘッドが生じるのを防止できる。
- ファイル内のオブジェクトを調べ、使用する操作を検討する。
- 読込み専用オブジェクトをグループ化して、読込み専用表領域に入れる。

ワークシートの使用による必要な PCM ロックの分析

大きなアプリケーションでは、関与する業務プロセスを慎重に検討します。次に示すようなワークシートを使用すると便利です。

システムで毎日実行する操作を種類ごとに分けます。X ロックの必要な操作と S ロックの必要な操作を区別することが重要です。モードを変更するたびに、ロックが必要です。同じ表の異なるインスタンス間の相互作用を考慮してください。また、ブロック内の行の数および表内の行の数、増加率も考慮に入れる必要があります。この分析に基づき、オブジェクトをグループ化してファイルに入れ、空きリスト・グループを割り当てます。

図 15-1 PCM ロック・ワークシート 1

オブジェクト	X モードが必要な操作 : 書込み			S モードが必要な操作 : 読込み	表領域 / データ・ファイル
	INSERT	UPDATE	DELETE	SELECT	
A		80%		20% 全表走査 単一行	
B				100%	
C					
D					

図 15-2 PCM ロック・ワークシート 2

オブジェクト	インスタンス 1	インスタンス 2	インスタンス 3
D	INSERT UPDATE DELETE	SELECT	
E			
F			

図 15-3 PCM ロック・ワークシート 3

表名	格納先の表領域	行サイズ	列数

データ・ブロックへの固定 PCM ロックのマッピング

多くの場合、頻繁に更新するデータに比べて読込み専用データには、ごく少数の PCM ロックしか必要ありません。これは、パラレル・サーバーのすべてのインスタンスが読込み専用データを共有できるためです。まったく更新されないデータには、1 つの PCM ロックを設定すれば済みます。読込み専用でないデータには、複数の PCM ロックが必要です。

データが読込み専用であると、一度インスタンスが読込み専用表領域の PCM ロックを所有すると、それを解放しません。最初のロックを取得した後は、統合分散ロックマネージャ（IDLM）の操作は不要です。

読込み専用表領域とその他の表領域を分けて、読込み専用領域に専用の PCM ロックを割り当てると、優れた成果を得ることができます。これには、書込み可能なデータが入っていない表領域に読込み専用データを入れます。その後で、GC_FILES_TO_LOCKS パラメータを使用して、表領域内のデータ・ファイルに PCM ロックを割り当てます。

注意： 読込み専用データと書込み可能データとを同じ表領域に入れないでください。

インスタンス間での PCM ロックの分割

特定のデータ・ブロックに PCM ロックをマップして、各インスタンスがアクセスするデータに基づき、インスタンス間に PCM ロックを分割することもできます。

この方法により、不要な分散ロック管理は最小限ですみます。同様に、要求されたデータ・ブロックは別のインスタンスが所有する PCM ロックが受け持っているので、データ・ブロックを書き込む必要があるインスタンスが原因で生じるディスク I/O も最小限になります。

たとえば、インスタンス X が主にデータ・ファイル 1、2 および 3 のデータを更新し、インスタンス Y が主にデータ・ファイル 4 および 5 のデータを更新する場合は、1 セットの PCM ロックをファイル 1、2 および 3 に割り当て、別のセットをファイル 4 および 5 に割り当てることができます。こうすれば、各インスタンスは、それが更新するデータに対する PCM ロックの所有権を取得します。インスタンスが PCM ロックを解放するのは、他のインスタンスが同じデータにアクセスする必要があるときだけです。

これに対して、データ・ファイル 3 および 4 に 1 セットの PCM ロックを割り当てた場合は、I/O が増加します。それは、両方のインスタンスが同じセットの PCM ロックを定常的に使用するからです。

GC_FILES_TO_LOCKS の設定 : 各データ・ファイルの PCM ロック数

1 つまたは一連のデータ・ファイル内のデータ・ブロックを受け持つ PCM ロックの数は、GC_FILES_TO_LOCKS 初期化パラメータで設定します。この項では、次の項目について説明します。

- GC_FILES_TO_LOCKS の構文
- 固定ロックの例
- 解放可能ロックの例
- ガイドライン

注意 : データ・ファイルの追加またはサイズ変更、表領域の作成、表領域およびそのデータ・ファイルの削除をしたときは、GC_FILES_TO_LOCKS の値を調整してから、OPS が使用可能な状態で Oracle を再起動してください。

関連項目 : 第 9 章「パラレル・キャッシュ管理インスタンス・ロック」（1 つの PCM ロックが受け持つデータ・ブロック数の決定方法について）

GC_FILES_TO_LOCKS の構文

GC_FILES_TO_LOCKS パラメータを設定する構文では、データベース・アドレスとデータベース・ブロック・クラス間の変換、およびそれをプロテクトするロック名を指定します。GC_FILES_TO_LOCKS パラメータに記述されていないファイルについては、この変換は指定できません。

このパラメータを設定するための構文は次のとおりです。

GC_FILES_TO_LOCKS="{*file_list=#locks*[!*blocks*][R][EACH][:]} . . ."

項目の内容は、次のとおりです。

file_list

file_list は、単一のファイル、ファイルの範囲、ファイルおよび範囲のリストを、次のように指定します。*fileidA*[-*fileidC*][,*fileidE*[-*fileidG*]] ...

ファイル名とファイル ID 番号の対応を調べるには、データ・ディクショナリ・ビュー DBA_DATA_FILES に問い合せます。

#locks

file_list に割り当てる PCM ロックの数を設定します。*#locks* に値 0(ゼロ) を指定すると、ハッシュ・ロックでなくファイン・グレイン・ロックを使用することを意味します。

!*blocks*

各ロックが受け持つ連続データ・ブロックの数を指定します。

EACH

#locks で指定されたロック数を *file_list* 内の各ファイルに割り当てます。

R

ハッシュ・ロックが解放可能であることを指定します。不要になったハッシュ・ロックをインスタンスが解放することがあります。解放可能なハッシュ PCM ロックは、GC_RELEASABLE_LOCKS プールから取得されます。

注意 : GC_ROLLBACK_LOCKS も同じ構文を使用します。

GC_FILES_TO_LOCKS パラメータの構文中、引用符で囲まれた範囲内に空白を入れてはなりません。

GC_FILES_TO_LOCKS は、データ・ファイルへの PCM ロックのマッピングを制御するだけでなく、デフォルト・バケットのロック数も制御します。デフォルト・バケットは、GC_FILES_TO_LOCKS で明示的に指示されないすべてのファイルで使用されます。ファイル番号の値 0（ゼロ）が使用され、デフォルトは "0=0" です。たとえば、"0=100"、"0=100R"、"0-9=100EACH" などを設定できます。このバケットのロックは、デフォルトでは解放可能になっていますが、これらのロックを固定に設定できます。

GC_FILES_TO_LOCKS パラメータに R オプションを付けることで、解放可能なハッシュ PCM ロックを指定できます。解放可能なハッシュ PCM ロックは、GC_RELEASABLE_LOCKS プールから取得されます。

REACH キーワードは、"R" と "EACH" を結合したもです。たとえば、GC_FILES_TO_LOCKS="0-9=100REACH" のように使用できます。EACHR は正しいキーワードではありません。

EACH および !*blocks* を指定しないと、*#locks* 個の PCM ロックがまとめて *file_list* に割り当てられ、個々の PCM ロックは、*file_list* 内のすべてのファイルのデータ・ブロックを受け持つことになります。ただし、PCM ロックの数より少ないデータ・ブロックを含むデータ・ファイルがある場合は、一部の PCM ロックはそのデータ・ファイル内のデータ・ブロックを受け持たないことになります。

!*blocks* のデフォルト値は 1 です。指定すると、*#locks* 個の PCM ロックの 1 つ 1 つが、*blocks* の指定値に相当する連続データ・ブロックを受け持ちます。*blocks* に値を指定するには、"!"

区切り子を使用する必要があります。複数のデータ・ファイルを受け持つ PCM ロックのセットを割り当てるには、主として *blocks* を指定し、EACH キーワードは指定しません。*blocks* を使用して、単一のデータ・ファイルを受け持つ一連の PCM ロックを、そのデータ・ファイルに対する PCM ロックの競合が一番少ない場所に割り当てることができ、この方法で PCM ロック管理を軽減することができます。

常に *!blocks* 値を設定することで、空きリスト・グループを使用して確保したデータ分割が崩れないようにしてください。通常は、ディスク領域を事前に割り当てる必要はありません。表に行を挿入するときに、新しいエクステントの割当てが必要になると、GC_FILES_TO_LOCKS パラメータの *!blocks* で指定した値に相当する連続したブロック数が、インスタンスに関連付けられている空きリスト・グループに割り当てられます。

固定ロックの例

たとえば、インスタンスのパラメータ・ファイルに次の行を追加することにより、ファイル 1 に 300 個のロック、ファイル 2 に 100 個のロックを割り当てることができます。

```
GC_FILES_TO_LOCKS = "1=300:2=100"
```

次のエントリは、ファイル 1、2 および 3 にそれぞれ 500 個ずつ、合計 1500 個のロックを指定します。

```
GC_FILES_TO_LOCKS = "1-3=500EACH"
```

これに対して、次のエントリは、3 つのファイル全体で、合計 500 個だけのロックを指定します。

```
GC_FILES_TO_LOCKS = "1-3=500"
```

次のエントリは、ファイル 1 をプロテクトするために、個別のロックが 1000 個必要であることを示します。ファイル内のデータは、25 ブロックのグループに分けてプロテクトされます。

```
GC_FILES_TO_LOCKS = "1=1000!25"
```

次のエントリは、ロックが不要になったとき、25 ブロックずつのグループ単位でファイル 1 をプロテクトしている 1000 個のハッシュ・ロックをインスタンスから解放できることを示しています。

```
GC_FILES_TO_LOCKS = "1=1000!25R"
```

解放可能ロックの例

グループ係数を使用してデータ・ブロックにファイン・グレイン・ロックを指定するには、インスタンスのパラメータ・ファイルに次のように指定します。

```
GC_FILES_TO_LOCKS="1=0!4"
```

これは、ファイル 1 について、グループ係数 4 のファイン・グレイン・ロックを指定しています。

ガイドライン

GC_FILES_TO_LOCKS パラメータを設定するには、次のガイドラインを使用してください。

- 常にすべてのデータ・ファイルを GC_FILES_TO_LOCKS に指定する。
- 同じデータベースにアクセスするすべてのインスタンスについて、GC_FILES_TO_LOCKS に同じ値を指定する。
- データ・ファイルに指定する PCM ロックの数は、そのデータ・ファイルのブロックの数を超えてはならない。この条件に従っていると、データ・ファイルのサイズが増加しても、余っている PCM ロックが新しいブロックを受け持つことができます。

  AUTOEXTEND 句を使用してデータ・ファイルを定義した場合、または ALTER DATABASE ... DATAFILE ... RESIZE コマンドを発行した場合は、データ・ファイルのサイズが増加していないかどうかを定期的に監視する必要があります。サイズが増加した場合は、できるだけ早くパラメータ GC_FILES_TO_LOCKS を更新し、パラレル・サーバーをシャットダウンしてから、再起動する必要があります。

  注意 : OPS を再起動する必要はありません。ただし、シャットダウンと再起動を行わない場合は、それを行った場合よりもロックが受け持つブロック数が多くなります。

  file_list に指定された PCM ロックの数がデータ・ファイル内の実際のデータ・ブロック数より小さい場合、IDLM は、指定数より多数のデータ・ブロックを受け持つ PCM ロックを使用してください。これはパフォーマンスの低下を招くことがあるので、常に、十分な数の PCM ロックを使用可能な状態にしておく必要があります。
- 新しいデータ・ファイルを追加するときは、そのためのロックを必ず GC_FILES_TO_LOCKS に指定して、" 予備 " の PCM ロックが自動的に割り当てられるのを回避できるようにする。

  ある時点で、OPS を実行したままで、ALTER TABLESPACE ... ADD DATAFILE コマンドを使用してデータ・ファイルを追加する必要が生じる場合があります。追加した場合は、できるだけ早く GC_FILES_TO_LOCKS を更新し、パラレル・サーバーをいったんシャットダウンし、再起動してください。
- 異なるインスタンスがアクセスする分離データの作成によってリソースの競合を軽減するには、データ・ファイルを異なるディスクに配置する必要がある。別個のデータ・ファイル内のデータ・ブロックを受け持つ PCM ロックを割り当てるには、GC_FILES_TO_LOCKS を使用してください。
- 変更頻度の低い索引データを含むブロックには、比較的少数の PCM ロックを割り当てる。各索引に個別の PCM ロック・セットを割り当てることができるように、索引を専

用の表領域または表領域内の専用のデータ・ファイルに入れてください。読込み専用索引には、必要な PCM ロックは 1 つだけである。

- GC_FILES_TO_LOCKS に指定されていないファイルは、DBA ファイン・グレイン・ロックを使用する。

GC_FILES_TO_LOCKS の設定のヒント

GC_FILES_TO_LOCKS の設定は、OPS での重要なチューニング作業です。ここでは、いくつかの簡単なチェックを行い、設定したパラメータが優れたパフォーマンスを実現できるかどうかを確認します。この項では、次の項目について説明します。

- 成長の余地
- 有効なロック数のチェック
- 有効なロック割当てのチェック
- 表領域を読込み専用に設定
- ファイルの妥当性のチェック
- パラメータ値を変更しないデータ・ファイルの追加

成長の余地

絶えず実行を続けるサイトでは、パラメータ値を調整するためにシャットダウンする余裕はありません。したがって、これらのパラメータのサイズを決めるときは、成長の余地、つまりファイルを拡張できる余地を見込んで決定してください。

また、データ・ファイルの追加またはサイズ変更、表領域の作成、表領域およびそのデータ・ファイルの削除をしたときは、GC_FILES_TO_LOCKS の値を調整してから、OPS が使用可能な状態で Oracle を再起動してください。

有効なロック数のチェック

ロックの数がデータ・ブロックの数を超えていないか、割り当てた個数を確認します。

注意： 表の中に挿入する時点では、現在のブロック割当てがゼロの場合もあります。

FILE_LOCK データ・ディクショナリ・ビューをチェックし、各ファイルに割り当てられたロック数を調べます。V$DATAFILE をチェックし、データ・ファイルの最大サイズを調べます。

関連項目：『Oracle8*i* リファレンス・マニュアル』（FILE_LOCK および V$DATAFILE の詳細）

有効なロック割当てのチェック

ロック割当ての問題を防止するために、次の点をチェックしてください。

- ロールバック・セグメントだけを入れるファイルには、ロックを割り当てない。
- 内部一時表に使用する一時データだけを入れるファイルには、ロックを割り当てない。
- 読込み専用オブジェクトをまとめてグループ化し、そのファイルにはロックを 1 つだけ割り当てる。これがうまく機能するのは、ファイルへの書込みが絶対になく、ブロックの消去などのブロックに対する変更もない場合だけです。可能であれば、該当する表領域を読込み専用にします。

表領域を読込み専用に設定

表領域が実際に読込み専用である場合は、それを Oracle で読込み専用に設定することを検討してください。そうすれば、データベースへの書込みは決して行われず、PCM ロックもまったく使用されません。ただし、例外として、表領域が競合により予備ロックを取得する必要がないように、ユーザーは 1 つだけロックを割り当てることができます。

ファイルの妥当性のチェック

次の方法で、各ファイル内のオブジェクトの数を計算します。

```
SELECT E.FILE_ID      FILE_ID,
       COUNT(DISTINCT OWNER||NAME ) OBJS
  FROM DBA_EXTENTS     E,
       EXT_TO_OBJ V
 WHERE E.FILE_ID = FILE#
   AND E.BLOCK_ID >= LOWB
   AND E.BLOCK_ID <= HIGHB
   AND KIND != 'FREE EXTENT'
   AND KIND != 'UNDO'
 GROUP BY E.FILE_ID;
```

複数のオブジェクトを格納しているファイルを調べます。CATPARR.SQL を実行し、EXT_TO_OBJ ビューを使用します。同じファイル内に共存できるオブジェクトであることを確認します。つまり、GC_FILES_TO_LOCKS の設定値に矛盾がないことを確認してください。

パラメータ値を変更しないデータ・ファイルの追加

データ・ファイルをデータベースに追加する場合は、その結果、PCM ロックの配分がどうなるかを考慮してください。追加ファイルにロックを割り当てるには、インスタンスをシャットダウンし、GC_FILES_TO_LOCKS パラメータを変更し、データベースを再起動する必要があります。これは、本番データベースでは不可能な場合もあります。

その場合、追加データ・ファイルは残りのロックのプールに割り当てられ、したがって GC_FILES_TO_LOCKS パラメータに指定されていないすべてのファイルと競合することになります。

その他の GC_* パラメータの設定

この項では、さらに 2 つの GC_* パラメータの設定方法について説明します。

- GC_RELEASABLE_LOCKS の設定
- GC_ROLLBACK_LOCKS の設定

GC_RELEASABLE_LOCKS の設定

GC_RELEASABLE_LOCKS には、デフォルトの設定をお薦めします。デフォルトは DB_BLOCK_BUFFERS の値です。デフォルトを使用すると、通常は最適なパフォーマンスを実現します。しかし、メモリーの節約のために GC_RELEASABLE_LOCKS をデフォルト値よりも小さくしたり、ロック・アクティビティを可能な限り削減するためにデフォルト値よりも大きく設定したりすることはできます。GC_RELEASABLE_LOCKS の値が小さすぎると、パフォーマンスを低下させる可能性があります。

V$SYSSTAT ビュー内の統計 "releasable freelist waits" は、システムの解放可能ロックが使い尽くされる回数を追跡します。この状態が発生した場合、つまり、解放可能な空きリスト待ちの値がゼロ以外になった場合は、GC_RELEASABLE_LOCKS の値を増加する必要があります。

GC_ROLLBACK_LOCKS の設定

固定ロックを使用する場合は、割り当てられているデータ・ブロック数よりも多くのロック数が割り当てられていないかを確認することをお薦めします。表の中に挿入する時点では、現在のブロック割当てがゼロの場合もあります。ロールバック・セグメントに割り当てられたブロックの数を検索するには、次のように入力します。

```
SELECT S.SEGMENT_NAME NAME,
       SUM(RBLOCKS) BLOCKS
  FROM DBA_SEGMENTS S,
       DBA_EXTENTS R
WHERE S.SEGMENT_TYPE = 'ROLLBACK'
  AND S.SEGMENT_NAME = R.SEGMENT_NAME
GROUP BY S.SEGMENT_NAME;
```

この問合せでは、各ロールバック・セグメントに割り当てられたブロックの数が表示されます。取消しブロックに多数の不要な強制読込み / 書込みが発生する場合は、解放可能ロックの使用を試みてください。デフォルトで、すべてのロールバック・セグメントは解放可能なロックによってプロテクトされます。

次の例に示すように、GC_ROLLBACK_LOCKS パラメータは、GC_FILES_TO_LOCKS パラメータとほとんど同じ引数をとります。

GC_ROLLBACK_LOCKS="0=100:1-10=10EACH:11-20=20EACH"

この例では、ロールバック・セグメント 0（システム・ロールバック・セグメント）に 100 個のロックがあります。ロールバック・セグメント 1 から 10 には、それぞれ 10 個ずつロックがあり、そしてロールバック・セグメント 11 から 20 には、それぞれ 20 個ずつロックがあります。

注意： GC_ROLLBACK_LOCKS を使用してロールバック・セグメント間でロックを共有させることはできません。次に示す例で、最初の例は無効ですが、2 番目の例では、ロールバック・セグメントがそれぞれに 100 ロックずつ所持しているので、これは有効な例です。

誤り : GC_ROLLBACK_LOCKS="1-10=100"

正しい : GC_ROLLBACK_LOCKS="1-10=100EACH"

PCM ロックのチューニング

この項では、PCM ロックをチューニングする前に検討すべき次の項目について説明します。

- false ping の検出
- PCM ロック変換の所要時間
- PCM ロック変換の完了を待っているセッションの判別
- 必要な PCM ロック数とリソースの合計

false ping の検出

false ping は、別々のノードから同時に更新される複数のブロックがある場合に、これらのブロックをプロテクトするロック要素を下位変換したときに発生します。たとえば、2 つのノードがそれぞれ異なるブロックを更新するとき、それらのブロックが同じロックによってプロテクトされているとします。この場合、各ノードは 1 つのブロックを更新するために、2 つのブロックを ping する必要があります。これは、同一ロックが両方のブロックを受け持つためです。

false ping アクティビティを表示するために利用できる統計はありません。false ping の検出は、周囲の状況から判断するしかありません。ここでは、対象となるアクティビティを見つける方法を説明します。

次の SQL 文は、書込みを生じさせているロック・オペレーションの数と実際に書き込まれたブロックの数を表示します。

```
SELECT VALUE/(A.COUNTER + B.COUNTER + C.COUNTER) "PING RATE"
  FROM V$SYSSTAT,
    V$LOCK_ACTIVITY A,
    V$LOCK_ACTIVITY B,
    V$LOCK_ACTIVITY C
WHERE A.FROM_VAL = 'X'
    AND A.TO_VAL = 'NULL'
    AND B.FROM_VAL = 'X'
    AND B.TO_VAL = 'S'
```

```
    AND C.FROM_VAL = 'X'
    AND C.TO_VAL = 'SSX'
    AND NAME = 'DBWR cross instance writes';
```

表 15-1 に、ping 率の解釈のしかたを示します。

表 15-1 ping 率の解釈

ping 率	意味
< 1	false ping が発生している可能性はありますが、ping の書込みを上回るロック・オペレーションがあります。DBWR が十分な速さでブロックの書出しを行っているので、ロック・アクティビティの書込みは発生していません。これは、"soft ping" とも呼ばれています。ロック・アクティビティだけが I/O を要求しており、ping の I/O は要求されていないことを意味します。
= 1	書込みの発生しそうなロック・アクティビティでは、実際に書込みが発生します。false ping が発生している可能性があります。
> 1	false ping が確実に発生しています。

確実に false である ping の割合を計算するには、次の式を使用します。

$$\frac{\text{ping 率} - 1}{\text{ping 率}} * 100$$

書込みの合計数をチェックし、false ping に基づく数値を計算します。

```
SELECT Y.VALUE "ALL WRITES",
    Z.VALUE "PING WRITES",
    Z.VALUE * pingrate "FALSE PINGS",
FROM V$SYSSTAT Z,
    V$SYSSTAT Y,
WHERE Z.NAME = 'DBWR cross instance writes'
AND Y.NAME = 'physical writes';
```

次の SQL 文により、ping 率が表示されます。

```
CREATE OR REPLACE VIEW PING_RATE AS
SELECT ((VALUE/(A.COUNTER+B.COUNTER+C.COUNTER))-1)/
    (VALUE/(A.COUNTER+B.COUNTER+C.COUNTER)) RATE
FROM V$SYSSTAT,
    V$LOCK_ACTIVITY A,
    V$LOCK_ACTIVITY B,
    V$LOCK_ACTIVITY C
WHERE A.FROM_VAL = 'X'
    AND A.TO_VAL   = 'NULL'
    AND B.FROM_VAL = 'X'
    AND B.TO_VAL   = 'S'
    AND C.FROM_VAL = 'X'
AND C.TO_VAL   = 'SSX'
AND NAME = 'DBWR cross instance writes';
```

目標は、全体的な ping だけでなく false ping も減らすことです。false ping を減らすには、GC_FILES_TO_LOCKS 内のインスタンス・ロックの配分を調べ、ファイル内のデータをチェックしてください。

PCM ロック変換の所要時間

PCM ロックの変換に必要な時間をチェックしてください。この時間は、システムによって異なります。ロックの変換の所要時間を調べるには、次の SQL 文を入力します。

```
SELECT *
FROM V$SYSTEM_EVENT
WHERE EVENT = 'lock element cleanup'
```

この SQL 文は、次のような表を表示します。

```
                      TOTAL_    TOTAL_      TIME_     AVERAGE_
EVENT                 WAITS     TIMEOUTS    WAITED    WAIT
--------------------  ------    -------     ------    ----------
lock element cleanup  32709     44          685660    20.9624262
```

これは、ロックの変換に 20.9 x 1 / 100 秒（0.209 秒）かかることを表しています。

PCM ロック変換の完了を待っているセッションの判別

PCM ロック変換が完了するのを待機中のセッション、および待ち状態が終わったばかりのセッションがどれかを調べるには、次の SQL 文を入力します。

```
SELECT *
FROM V$SESSION_WAIT
WHERE EVENT = 'lock element cleanup'
```

必要な PCM ロック数とリソースの合計

ここでは、使用しているシステムに PCM ロック数とリソースがどのくらい必要かを判断する方法について説明します。これは、LM_LOCKS および LM_RESS パラメータを設定するために必要な値です。

PCM ロック数およびリソースの計算式

この値を割り出すには、インスタンス当りの固定（非解放可能）ロック数（GC_FILES_TO_LOCKS と GC_ROLLBACK_LOCKS の固定ロック数のみの合計）と、解放可能ロック数の合計（GC_RELEASABLE_LOCKS の値）を加算し、その結果に 2 を掛けます。

2 * (GC_FILES_TO_LOCKS + GC_ROLLBACK_LOCKS 固定ロック + GC_RELEASABLE_LOCKS)

この数字は、使用しているシステムに必要な PCM ロックとリソースの最大数を表します。この計算は、インスタンス数とは無関係です。

次のことも考慮してください。

- GC_FILES_TO_LOCKS: デフォルトの値は解放可能です。すべてのインスタンスで同じ設定にする必要があります。
- GC_ROLLBACK_LOCKS: デフォルトは解放可能です。すべてのインスタンスで同じ設定にする必要があります。
- GC_RELEASABLE_LOCKS: デフォルトは解放可能で、値は DB_BLOCK_BUFFERS です。

PCM ロック数およびリソースの計算 : 例

たとえば、使用しているシステムで、各インスタンスで次のように設定されているとします。

GC_FILES_TO_LOCKS="1=100:2-5=1000:6-10=1000EACH:11=100R"

GC_ROLLBACK_LOCKS="1-10=10EACH:11-20=20EACH"

GC_RELEASABLE_LOCKS=50,000

GC_FILES_TO_LOCKS の合計値は、次のように計算します。ファイル 1 に 100 個の固定ロックがあります。ファイル 2、3、4 および 5 は、1000 個のロックを共有します。ファイル 6 に 1000 個の固定ロック、ファイル 7 に 1000 個の固定ロック、ファイル 8 に 1000 個の固定ロック、ファイル 9 に 1000 個の固定ロック、そしてファイル 10 に 1000 個の固定ロックがあります。ファイル 11 には、固定ロックがありません。したがって、GC_FILES_TO_LOCKS によって設定される固定ロック数の合計は 6,100 です。

GC_ROLLBACK_LOCKS の合計値は、次のように計算します。ファイル 1 から 10 までは各ファイルに 10 ずつ固定ロックがあり、ファイル 11 から 20 までは各 20 ずつ固定ロックがあります。これらを合計すると、全部で 300 個の固定ロックがあります。

これらの数字を式に代入し、次のように計算します。

2 * (6,100 + 300 + 50,000) = 112,800

したがって、LM_LOCKS および LM_RESS パラメータに設定する値は 112,800 です。

注意： 上記の式は、PCM リソースの計算だけに使用します。

関連項目： 18-11 ページの「必要なロックの量の決定および LM_* パラメータの設定」（ロックおよびリソースを計算する式について）、15-6 ページの「GC_FILES_TO_LOCKS の構文」（このパラメータを使用するときの構文について）

16

リソースおよびロック用の IDLM 容量の確保

共有リソースに対する競合を減らし、Oracle Parallel Server（OPS）のパフォーマンスを最大にするには、システムに必要なすべてのロックおよびリソースに対して、統合分散ロック・マネージャ（IDLM）が適切に構成されている必要があります。この章では次の項目について説明します。

- 概要
- IDLM 容量の計画
- 非 PCM リソース数の計算
- Oracle 初期化パラメータの調整
- 表ロック数の最小化によるパフォーマンスの最適化

関連項目： 第 10 章「非 PCM インスタンス・ロック」（概念の説明）

概要

システムのロックを管理するには、PCM ロックを計画するだけでは不十分です。パラレル・キャッシュ管理のロックを明示的に割り当てる以外に、必要なすべての PCM ロックと非 PCM ロック、およびリソースについて、各ノード上に IDLM が適切に構成されることを確認する必要があります。また、データベースを大きくして並列度を高めると、多数のリソースへの要求が増加することも考慮に入れる必要があります。

非 PCM ロックには多数の異なるタイプがあり、それぞれ取扱い方法が異なります。数を直接調整することはできませんが、必要な非 PCM リソースおよびロックの総数を見積ることができ、十分な領域を確保するための LM_* または GC_* 初期化パラメータ（またはその両方）を調整することができます。表ロックを最小限に抑えてパフォーマンスを最適化するためのオプションもあります。

IDLM 容量の計画

IDLM によって管理するリソースおよびロックの数を、慎重に計画して構成します。これらのロックおよびリソースは、初期化パラメータ LM_LOCKS および LM_RESS を使用して割り当てます。追加のロックおよびリソースを動的に割り当てることはできますが、それは行わないでください。

リソースおよびロックの動的割当ての回避

必要なロック数またはリソース数が、割り当てた数量を上回ると、追加のロックまたはリソースが SGA 共有プールから割り当てられます。これにより、インスタンスは停止せずに稼働し続けることができます。

動的割当てが発生すると、Oracle によりアラート・ファイルにメッセージが書き込まれます。このメッセージには、次回にデータベースを起動するときのために、初期化パラメータを再計算して調整する必要があることが示されています。動的割当ては、パフォーマンスおよびメモリー使用に悪影響を及ぼすことがあるので、必要なロックおよびリソースを正確に計算しておくことをお奨めします。

SHARED_POOL_SIZE の推奨値

SHARED_POOL_SIZE のデフォルト値は、64 ビット・アプリケーションでは 16MB を、32 ビット・アプリケーションでは 8MB をお奨めします。

必要なロックおよびリソースの計算

次の方法により、必要な PCM と非 PCM のリソース数およびロック数の合計について、1 ノードあたりの IDLM 容量を慎重に計画します。

1. フェイルオーバー要件を考慮します。

   フェイルオーバーが発生した場合に、システムが操作を続行できるようにするには、残りのインスタンスに十分なリソースを構成している必要があります。したがって、リソースが 10 のインスタンスで共有されているときに 5 インスタンスに障害が発生した場合、システムは残りの 5 インスタンスでも処理を実行できる必要があります。それには、オーバーヘッドを考慮に入れてシステム・リソースを多少多めに見積る必要があります。つまり、各インスタンスごとに IDLM ロックおよびリソースを決定する Oracle 初期化パラメータに対して、十分に大きい値を設定します。

2. 各ノードの各インスタンスのサイズ設定を考慮します。つまり、ユーザーの数、トランザクションの量などです。各インスタンスの初期化パラメータに割り当てる値を決定します。

3. この章に示すワークシートに記入することによって、必要な非 PCM リソースおよびロックの数を計算します。

4. 15-17 ページの「必要な PCM ロック数とリソースの合計」に示したスクリプトを使用して、必要な PCM リソースおよびロック数を計算します。

5. 次のリソースおよびロックの必要数を収容できるように IDLM を構成します。
   - 非 PCM リソース
   - 非 PCM ロック
   - PCM リソース
   - PCM ロック

リソース使用率の監視

V$RESOURCE_LIMIT ビューでは、数種のシステム・リソースについてのグローバルなリソース使用率の情報が提供されます。このビューによって、リソース使用率の現在値および最大値を監視できます。これらの値が制限値に接近したとき、それに気づくことが重要です。この情報を使用することで、リソースの制限値を制御するパラメータに値を選択する際、最適な決定を行うことができます。

関連項目： 18-11 ページの「必要なロックの量の決定および LM_* パラメータの設定」、『Oracle8*i* リファレンス・マニュアル』（V$RESOURCE_LIMIT について）、『Oracle8*i* チューニング』（リソース制限についての説明）

非 PCM リソース数の計算

システム・リソースを分析するには、次のワークシートを使用します。

1. 次のワークシートで、PROCESSES、DML_LOCKS、TRANSACTIONS および ENQUEUE_RESOURCES 初期化パラメータの値を各インスタンスごとに記入します。
2. 各インスタンスごとに、DB_FILES パラメータの値を入力します。この値はすべてのインスタンスについて同じです。
3. 各インスタンスごとに、エンキュー・ロックの値を入力します。この値は、各インスタンスごとに、次の方法で計算できます。

   エンキュー・ロック = 20 + (10 * SESSIONS) + DB_FILES + 1 + (2 * PROCESSES) + (DB_BLOCK_BUFFERS / 64)

4. 各インスタンスについて、「PQ（パラレル問合せ）オーバーヘッド」にインスタンス間の通信を受け持つ値を記入します。この値は、個々のインスタンスについて、次の方法で計算できます。

   PQ オーバーヘッド = 7 + (MAXINSTANCES * PARALLEL_MAX_SERVERS) + PARALLEL_MAX_SERVERS + MAXINSTANCES

5. 項目を横に加算して、「小計 : 1 インスタンス当たりの非 PCM リソース数」を出します。
6. 各インスタンス当たりの小計を加算して、「システム全体の非 PCM リソースの合計数」を出します。

表 16-1 ワークシート : 非 PCM リソース数の計算

インスタンス番号	PROCESSES	DML_LOCKS	TRANSACTIONS	ENQUEUE_RESOURCES	DB_FILES（1 つまたは複数のインスタンスについて）	エンキュー・ロック	PQ オーバーヘッド	オーバーヘッド	小計 : 1 インスタンス当りの非 PCM リソース数
1								200	
2								200	
3								200	
4								200	

システム全体の非 PCM リソース数の合計 :

7. 最後に、このワークシートから得た数値を使用して、LM_RESS パラメータがすべての非 PCM リソースを収容する値に設定されているかを確認します（16-4 ページのステップ 6 を参照）。

注意： このワークシートでは、各インスタンスのオーバーヘッド値を標準値 200 に想定しています。

表 16-2 に、PARALLEL_MAX_SERVERS がインスタンス 1 と 3 に対して 8 に設定され、インスタンス 2 と 4 に対して 4 に設定された、4 つのインスタンスが実行されているシステムについてサンプル値を示します。バッファ・キャッシュ・サイズは 10K を想定しています。

表 16-2 非 PCM リソース数の計算（例）

インスタンス番号	PRO-CESSES	DML_ LOCKS	TRANS-ACTIONS	ENQUEUE_ RESOURCES	DB_FILES（1 つまたは複数のインスタンスについて）	エンキュー・ロック	PQ オーバーヘッド	オーバーヘッド	小計：1 インスタンス当りの非 PCM リソース数
1	200	500	50	800	30	2808	51	200	4,639
2	350	600	100	1,000	--	4128	31	200	6,409
3	175	400	75	800	--	2453	51	200	4,154
4	225	350	125	1,200	--	3103	31	200	5,234
システム全体の非 PCM リソース数の合計：									20,436

Oracle 初期化パラメータの調整

必要な非 PCM ロックおよびリソースのための十分な領域をシステムに確保するためには、次の Oracle 初期化パラメータの値を調整する方法もあります。

- DB_BLOCK_BUFFERS
- DB_FILES
- DML_LOCKS
- PARALLEL_MAX_SERVERS
- PROCESSES
- SESSIONS
- TRANSACTIONS

最初に、この章に示したワークシートの値を使用してテストしてみます。ワークシートのパラメータ値を故意に大きくして、その結果 IDLM がフェイルオーバー用の余地を提供するかどうかを調べることができます。

ただし、実際のパラメータ値を、各インスタンスについて必要な値よりあまり大きい値に設定しないでください。これらのパラメータを不必要に大きく設定すると、パラレル・サーバー環境でのオーバーヘッドが増加します。

表ロック数の最小化によるパフォーマンスの最適化

この項では、表ロック数を最小限に抑えることによってパフォーマンスを改善する、次の 2 つの方針について説明します。

- DML_LOCKS を 0（ゼロ）に設定
- 表ロックの使用禁止

挿入、削除および更新のために表ロック（DML ロック）を取得すると、OPS でのパフォーマンスが低下することがあります。OPS で表をロックすると、その表でロックを保持しているすべてのインスタンスがロックを解除する必要が生じるので、このような表のロックは不要な操作です。この種のロックを完全に使用禁止にすることを考慮してください。

注意： これらの方針のどちらかを使用すると、インスタンスおよび表に対して DDL コマンドを実行できません。

DML_LOCKS を 0（ゼロ）に設定

表ロックは、初期化パラメータ DML_LOCKS を使用して設定します。DROP TABLE、CREATE INDEX および LOCK TABLE コマンドが必要ない場合は、ロック変換を最小限に抑えて最も高いパフォーマンスが得られるように、DML_LOCKS を 0（ゼロ）に設定します。

注意： 1 つのインスタンスについて DML_LOCKS を 0（ゼロ）に設定した場合は、すべてのインスタンスについても 0（ゼロ）に設定する必要があります。その他の値を使用する場合は、このパラメータをすべてのインスタンスについて同じ値にする必要はありません。

表ロックの使用禁止

どのユーザーも表ロックを獲得できないようにするには、次のコマンドを使用します。

```
ALTER TABLE table_name DISABLE TABLE LOCK
```

表ロックが使用禁止になっているときにその表をロックしようとすると、エラーが発生します。

表ロックを再び使用可能にするには、次のコマンドを使用します。

```
ALTER TABLE table_name ENABLE TABLE LOCK
```

上記のコマンドでは、現在実行中のすべてのトランザクションがコミットするまで待ってから、表がロックされます。このコマンドは、ENABLE コマンドの発行後に開始された新規トランザクションの完了を待つ必要はありません。

ある表についてその表ロックが使用可能になっているか使用禁止になっているかを判別するには、データ・ディクショナリ表 USER_TABLES の TABLE_LOCK 列を問い合せます。DBA_TABLES または ALL_TABLES の選択権限を持っている場合は、他のユーザーの表のロック状態についても問い合せることができます。

17

空きリスト・グループによるデータ分割

この章では、空きリストおよび空きリスト・グループを割り当ててデータを分割する方法について説明します。データを分割することによって、複数インスタンスを使用するときに生じる空き領域獲得のための競合を最小限に抑えることができます。

注意： アプリケーションの制約により表を分割できないときは、データを分割するための空きリスト・グループを使用するだけです。空きリスト・グループと同じことを行うには、パーティション表とパーティション索引を使用した方が簡単です。

この章では、次の事項について説明します。

- 概要
- データベース・オブジェクト用の空き領域の分割方法の決定
- CREATE 文の FREELISTS および FREELIST GROUPS の設定
- インスタンス、ユーザーおよびロックと空きリスト・グループとの関連付け
- エクステントの事前割当て（オプション）
- エクステントの動的割当て
- 未使用領域の識別および割当て解除

関連項目： 第 11 章「領域管理および空きリスト・グループ」（概念の説明）および『Oracle8*i* 概要』

概要

複数インスタンスの空き領域を管理するには、次の手順に従ってください。

1. データベース・オブジェクトを分析して、空き領域およびデータをどのように分割するかを決定します。
2. 個々の表、クラスタおよび索引について、CREATE 文の FREELISTS および FREELIST GROUPS を設定します。
3. インスタンス、ユーザーおよびロックを空きリストに関連付けます。
4. 空きリストにブロックを割り当てます。
5. 必要な場合には、エクステントを事前に割り当てます。

空き領域を効果的に管理することによって、OPS にとって最適とはいえないアプリケーション構成の場合でもパフォーマンスを改善できます。

注意： 最適のシステム・パフォーマンスを得るには、十分に注意して FREELIST オプションおよび FREELIST GROUPS オプションを設定してください。これらの値は再設定できません。

データベース・オブジェクト用の空き領域の分割方法の決定

この項では、データベース・オブジェクトを分析し、最適のパフォーマンスを得るための空き領域およびデータの分割方法を決定するのに役立つワークシートを示します。

- データベース・オブジェクト特性
- 空き領域ワークシート

データベース・オブジェクト特性

作成するデータベース・オブジェクトを分析し、この項で説明するカテゴリに分類します。

静的表のオブジェクト

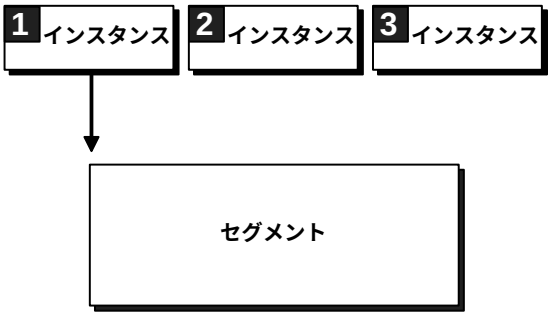
表への挿入アクティビティが頻繁でない場合は、空きリストまたは空きリスト・グループは不要です。

図 17-1 静的表のデータベース・オブジェクト

分割済みのアプリケーションのオブジェクト

特定のアプリケーションを適正に分割した場合は、表またはセグメントに挿入する必要のあるノードは 1 つだけです。この場合、ユーザーが多ければ空きリストが必要なこともありますが、空きリスト・グループは必要ありません。

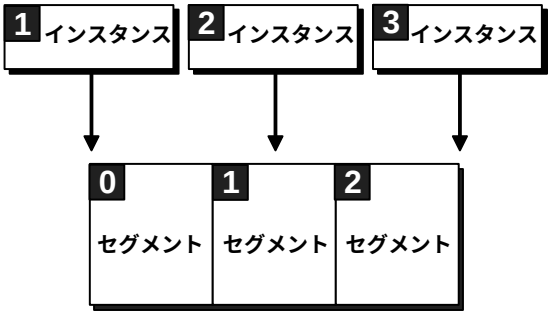
図 17-2 分割済みのアプリケーションのデータベース・オブジェクト



分割済みのデータに関連するオブジェクト

分割済みのデータを持つオブジェクトには、複数の空きリストおよび空きリスト・グループは必要ありません。

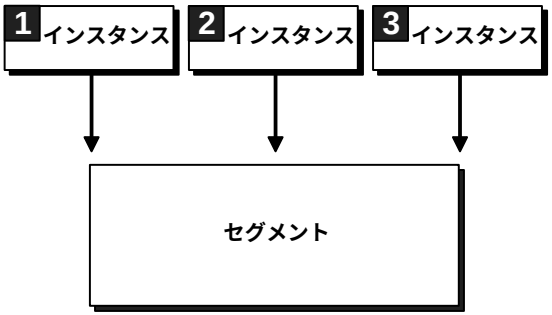
図 17-3 分割済みのデータに関連するデータベース・オブジェクト



ランダム挿入のある表のオブジェクト

同じ表の中で複数のインスタンスからランダム挿入が発生する場合は、空きリストおよび空きリスト・グループが必要です。セグメントに書き込むすべてのインスタンスは、マスター空きリストをチェックして、どこに書き込むかを判別しなければなりません。したがって、マスター空きリストが入っているセグメント・ヘッダーを獲得するための競合が発生します。

図 17-4 ランダム挿入のある表のデータベース・オブジェクト



空き領域ワークシート

個々のデータベース・オブジェクト（表、クラスタ、索引など）を次のようなワークシートに示し、各オブジェクトごとに空きリストおよび空きリスト・グループを計画します。

表 17-1 データベース・オブジェクト用の空き領域ワークシート

データベース・オブジェクト特性	空きリスト・グループ	空きリスト
静的表のオブジェクト	なし	なし
	なし	なし
	なし	なし
	なし	なし
分割済みのアプリケーションのオブジェクト	なし	
	なし	
	なし	
	なし	
分割済みのデータに関連するオブジェクト	なし	なし
	なし	なし
	なし	なし
	なし	なし
ランダム挿入のある表のオブジェクト		

注意： 上記の分割済みデータと Oracle8*i* パーティションを混同しないでください。Oracle8*i* パーティションは、使用されている場合とそうでない場合があります。

CREATE 文の FREELISTS および FREELIST GROUPS の設定

この項では、次の項目について説明します。

- FREELISTS オプション
- FREELIST GROUPS オプション
- クラスタ用の空きリストの作成
- 索引用の空きリストの作成

空きリストおよび空きリスト・グループを作成するには、CREATE TABLE 文、CLUSTER 文または INDEX 文に FREELISTS および FREELIST GROUPS 記憶領域オプションを指定します。この作成操作は、排他モードまたは共有モードのどちらかでデータベースにアクセスしている間に実行できます。

注意： これらの記憶領域オプションを設定した後で、ALTER TABLE 文、CLUSTER 文または INDEX 文を使用して記憶領域オプションの値を変更することはできません。

関連項目： 『Oracle8*i* SQL リファレンス』の STORAGE 句（前述のオプションの構文）

FREELISTS オプション

FREELISTS は、各空きリスト・グループ内の空きリストの数を指定します。FREELISTS のデフォルト値は 1 で、最小値でもあります。最大値は、データ・ブロック・サイズに応じて異なります。指定した値が大きすぎると、エラー・メッセージによって最大値が表示されます。FREELISTS の最適値は、この表の空きリスト・グループ 1 つ当りについて予期されている同時挿入の数に応じて異なります。

FREELIST GROUPS オプション

各空きリスト・グループは、起動時に 1 つまたは複数のインスタンスに関連付けられます。FREELIST GROUPS のデフォルト値は 1 で、表の空きリストがある場合にそれをすべてのインスタンスが使用できることを意味します。通常、FREELIST GROUPS には、OPS のインスタンス数を設定します。空きリスト・グループを使用した場合も、データ分割が生じます。一方のインスタンスに割り当てられたブロックや他方のインスタンスから解放されたブロックは、最初のインスタンスでは使用できません。

注意： 非共有環境であっても、複数の空きリスト・グループを設定する方がパフォーマンスを向上できます。複数の空きリスト・グループを使用すると、空きリスト構造体がセグメント・ヘッダーから切り離され、セグメント・ヘッダー獲得のための競合が減少します。これは、大量の UPDATE および INSERT トランザクションがある場合に非常に便利です。

例 次の文では、それぞれ 4 つの空きリストを含む 7 つの空きリスト・グループを持つ、DEPT という名前の表が作成されます。

```
CREATE TABLE dept
         (deptno   NUMBER(2),
          dname    VARCHAR2(14),
          loc      VARCHAR2(13) )
         STORAGE ( INITIAL 100K        NEXT 50K
                   MAXEXTENTS 10       PCTINCREASE 5
                   FREELIST GROUPS 7   FREELISTS 4 );
```

クラスタ用の空きリストの作成

クラスタ表の場合は、CREATE TABLE 文に FREELISTS および FREELIST GROUPS 記憶領域オプションを指定することはできません。空きリスト・オプションは、個々の表についてでなく、クラスタ全体について指定する必要があります。なぜならば、クラスタ内の表では、CREATE CLUSTER 文の記憶領域パラメータが使用されるからです。

クラスタは、共通の列を持つ表のグループにデータを格納するためのオプションの方法です。アクセス時間を短縮するために、同じクラスタ内の 2 つ以上の表の関連する列は、物理的にいっしょにまとめてデータベース内に格納されます。OPS では、ハッシュ・クラスタ以外のクラスタは、複数の空きリストおよび空きリスト・グループを使用できます。

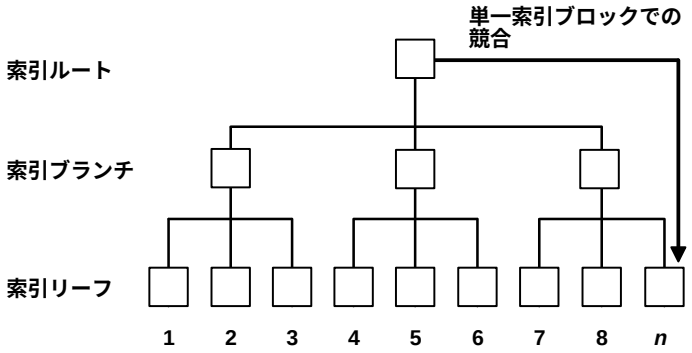
ハッシュ・クラスタの中には、ハッシュ関数についてユーザー定義のキーを使用して作成され、そのキーがインスタンス別に分割されている場合に、複数の空きリストおよび空きリスト・グループを使用できるものもあります。

索引用の空きリストの作成

CREATE INDEX 文の FREELISTS および FREELIST GROUPS 記憶領域オプションを使用して、同時ユーザー・プロセス用の複数の空き領域リストを作成することができます。これらのオプションの使用方法は、表の場合と同じです。

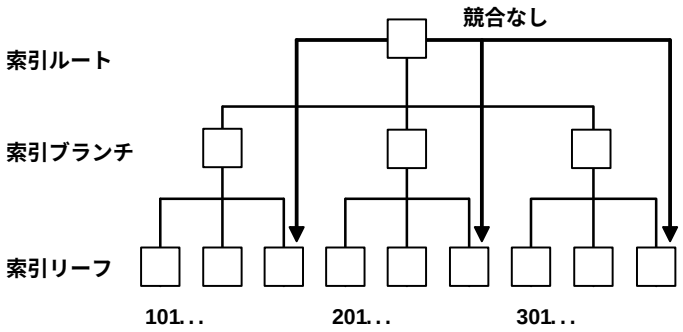
索引を持つ 1 つの表に複数のインスタンスが同時に行を挿入する場合は、インスタンス別に索引値を分離できない限り、索引ブロック獲得のための競合が発生してパフォーマンスが低下します。ある索引の同一リーフ・ブロック（*n*）の中にすべてのインスタンスが挿入しようとしている状況を、図 17-5 に示します。

図 17-5 1 つの索引ブロックを獲得するための競合



この問題を回避するには、図 17-6 に示すように、各インスタンスが専用のツリーに挿入するようにします。

図 17-6 競合のない索引



索引値を計算するには、次のようなアルゴリズムを使用します。

*instance_number * (100000000) + sequence_number*

インスタンス、ユーザーおよびロックと空きリスト・グループとの関連付け

ここでは、次のものを空きリスト・グループと関連付ける方法について説明します。

- インスタンスと空きリスト・グループとの関連付け
- ユーザー・プロセスと空きリスト・グループとの関連付け
- PCM ロックと空きリスト・グループの関連付け

インスタンスと空きリスト・グループとの関連付け

インスタンスをエクステントまたは空きリスト・グループと関連付けるには、次のようにします。

INSTANCE_NUMBER パラメータ	INSTANCE_NUMBER 初期化パラメータとともに各種の SQL オプションを使用して、データ・ブロックのエクステントをインスタンスに関連付けることができます。
SET INSTANCE オプション	ALTER SESSION コマンドの SET INSTANCE オプションを使用して、セッションが接続されているインスタンスに関係なく、そのセッションが特定インスタンスに関連付けられている空きリスト・グループを使用できるようにすることができます。次に例を示します。 ALTER SESSION SET INSTANCE = *inst_no*

SET INSTANCE 機能は、インスタンスに障害が起きてユーザーが別のインスタンスに接続するときに便利です。たとえば、表の空きリスト・グループに領域が事前に割り当てられているデータベースを考えてみます。ユーザーがインスタンス間に分散していてデータが適切に分割され、データ・ブロックの ping の発生が最小限に抑えられています。あるインスタンスに障害が起きても、各新規セッションが障害の起きたインスタンスに関連付けられている元の空きリスト・グループを使用できるので、すべてのユーザーを他のインスタンスに移動させても、それによってデータ分割が損なわれることはありません。

ユーザー・プロセスと空きリスト・グループとの関連付け

ユーザー・プロセスは、次に示すように、各ユーザーが実行中のプロセスの Oracle プロセス ID に基づき、空きリストに自動的に関連付けられます。

(oracle_pid modulo #free_lists_for_object) + 1

ALTER SESSION SET INSTANCE 文を使用することで、特定のインスタンスに関連付けられた空きリスト・グループを使用することができます。

PCM ロックと空きリスト・グループの関連付け

表の各エクステントが別個のデータ・ファイルに入っている場合は、GC_FILES_TO_LOCKS パラメータを使用して、特定範囲の PCM ロックを各エクステントに割り当て、PCM ロックの各セットを 1 つの空きリスト・グループだけに関連付けることができます。

関連項目： 11-14 ページの「インスタンス、ユーザーおよびロックに関連付けられた空きリスト」

エクステントの事前割当て（オプション）

この項では、エクステントの事前割当ての方法について説明します。この方法は便利ですが、エクステント割当てへの静的アプローチにはデータベース管理上のオーバーヘッドがある程度必要になるので注意してください。

- ALLOCATE EXTENT オプション
- MAXEXTENTS、MINEXTENTS および INITIAL パラメータの設定
- INSTANCE_NUMBER パラメータの設定
- エクステントの事前割当ての例

ALLOCATE EXTENT オプション

ALTER TABLE 文または ALTER CLUSTER 文の ALLOCATE EXTENT オプションを使用すると、エクステント・サイズおよびデータ・ファイル、空きリスト・グループを指定するオプションを用いて、表、索引またはクラスタにエクステントを事前に割り当てることができます。

ALLOCATE EXTENT オプションの構文は、『Oracle8*i* SQL リファレンス』の ALTER TABLE 文および ALTER CLUSTER 文の説明の中で示しています。

排他モードと共有モード – データベースを排他モードで実行している間は、共有モードの場合と同様に、ALTER TABLE（または CLUSTER）ALLOCATE EXTENT 文を使用できます。インスタンスは、排他モードで実行している間でも、領域の検索については同じルールに従います。トランザクションは、そのインスタンスについて、マスター空きリストまたは特定の空きリスト・グループを使用することができます。

SIZE オプション – ALLOCATE EXTENT 句の SIZE オプションには、エクステント・サイズをブロック・サイズの倍数に切り上げて、バイト単位で指定します。SIZE を指定しない場合には、エクステント・サイズは、記憶領域パラメータ NEXT および PCTINCREASE の値をもとにして計算されます。

後続のエクステント割当てでは、SIZE の値は計算の基礎として使用されません。後続のエクステント割当ては、NEXT および PCTINCREASE に基づいて決定されます。

DATAFILE オプション – このオプションでは、エクステント用の領域を確保するためのデータ・ファイルを指定します。このオプションを指定しないと、領域は、該当の表を含む表領域の中のアクセス可能なデータ・ファイルから割り当てられます。

ファイル名は、大 / 小文字の区別も含めて、制御ファイルに格納されている文字列に正確に一致している必要があります。DBA_DATA_FILES データ・ディクショナリ・ビューをチェックして、この文字列を確認できます。

INSTANCE オプション – このオプションでは、インスタンス番号の整数に対応付けられた空きリスト・グループに新しい領域を割り当てます。各インスタンスは、起動時に、そのインスタンスを空きリスト・グループにマップする一意のインスタンス番号を獲得します。最小インスタンス番号は、0 ではなく 1 です。最大値はオペレーティング・システムによって異なります。構文は、次のとおりです。

ALTER TABLE *tablename* ALLOCATE EXTENT (...INSTANCE *n*)

ここで、*n* は同じ番号の空きリスト・グループにマップされます。インスタンス番号が空きリスト・グループの数より大きい場合は、次のようにハッシュされて、割り当てられる空きリスト・グループが判別されます。

modulo(*n*,*#_freelistgroups*) + 1

INSTANCE オプションを指定しない場合は、新規領域は表には割り当てられますが、どの空きリスト・グループにも割り当てられません。このような領域は、他に使用可能な領域がないときに、必要に応じて空きブロックのマスター空きリストに組み込まれます。

注意： INSTANCE 値は、実際のインスタンス番号でなく、使用したい空きリスト・グループに対応したインスタンス番号を使用してください。

関連項目： 18-14 ページの「インスタンス番号と起動順序」

MAXEXTENTS、MINEXTENTS および INITIAL パラメータの設定

自動割当てを防止するには、特定のインスタンスに関連付けられている空きリスト・グループにエクステントを事前に割り当て、MAXEXTENTS をエクステントの現行数（事前に割り当てたエクステントの数に MINEXTENTS の設定値を足した数）に設定します。表またはクラスタを作成するときの初期割当てを最小限に抑えるには、MINEXTENTS を 1（デフォルト）に設定し、INITIAL をその最小値（2 データ・ブロック、またはブロック・サイズが 2048 バイトの場合は 10K）に設定します。

インスタンス間のデータ・ブロック獲得のための競合を最小限に抑えるには、各表ごとに複数のデータ・ファイルを作成し、各インスタンスを異なるファイルに関連付けます。

疎結合システムで、将来ノード数を増やすことを計画している場合は、現行のインスタンス数より多数の空きリスト・グループを持つ表またはクラスタを作成することによって、追加インスタンス用に余裕を残しておくことができます。このような空きリスト・グループには、必要になるまで領域を割り当てる必要はありません。この場合、自動割当てされる領域を持っているのは、空きブロックのマスター空きリストだけです。

空きリスト・グループに関連付けられるデータ・ブロックは、その空きリスト・グループを使用するインスタンスで実行中のプロセスにより PCTUSED より下に入れるか、またはその空きリスト・グループに特別に割り当てる必要があります。したがって、使用されない空きリスト・グループは、未使用空きデータ・ブロックから離れることはありません。

INSTANCE_NUMBER パラメータの設定

INSTANCE_NUMBER 初期化パラメータを使用すると、インスタンスを起動でき、そのインスタンスに割り当てられているエクステントだけを挿入および更新に使用できます。これにより、インスタンスが他のインスタンス用の領域を使用しないことを保証できます。このインスタンスは、同じ INSTANCE_NUMBER により再起動されるまでは、別の空きリスト内のデータ・ブロックを使用できません。

ALTER SESSION 文を使用すれば、インスタンス番号をそのセッション内で上書きすることもできます。

エクステントの事前割当ての例

この項では、エクステントの事前割当ての例を示します。

例 1 次の文では、データ・ファイル DEPT_FILE7 の表 DEPT に使用するエクステントが、インスタンス番号 7 に割り当てられます。

```
ALTER TABLE dept
ALLOCATE EXTENT ( SIZE 20K
                 DATAFILE 'dept_file7'
                INSTANCE 7);
```

例 2 次の SQL 文では、それぞれ 10 個の空きリストを含む 3 つの空きリスト・グループを持つ表が作成されます。

```
CREATE TABLE table1 ... STORAGE (FREELIST GROUPS 3 FREELISTS 10);
```

次の SQL 文で新しい領域を割り当て、割り当てたブロックを 2 番目の空きリスト・グループ内の空きリスト間に分割します。

```
ALTER TABLE table1 ALLOCATE EXTENT (SIZE 50K INSTANCE 2);
```

FREELIST GROUPS 記憶領域オプションの値より多数のインスタンスを実行するパラレル・サーバーでは、複数のインスタンスが新しい領域割当てを共有します。この例では、3 番目に起動されるインスタンスはすべて、同じ空きリスト・グループに関連付けられます。

例 3 次の CREATE TABLE 文では、初期エクステントを 1 つ、空きリスト・グループを 3 つ持った EMP という名前の表を作成し、3 つの ALTER TABLE 文が各空きリスト・グループに新しいエクステントを割り当てます。

```
CREATE TABLE emp ...
 STORAGE ( INITIAL 4096
           MINEXTENTS 1
           MAXEXTENTS 4
           FREELIST GROUPS 3 );
ALTER TABLE emp
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile1' INSTANCE 1 )
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile2' INSTANCE 2 )
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile3' INSTANCE 3 );
```

MAXEXTENTS は、自動割当てを防止するために、4（つまり MINEXTENTS の値と FREELIST GROUPS の値の合計値）に設定されます。

この割当て以上の追加領域が必要な場合は、追加エクステントを割り当てる前に、ALTER TABLE を使用して MAXEXTENTS を増加します。たとえば、空きリストの 2 番目のグループに挿入および更新用の追加の空き領域が必要な場合、MAXEXTENTS を 5 に設定して、その空きリスト・グループ用に別のエクステントを割り当てることができます。

```
ALTER TABLE emp  ...
 STORAGE ( MAXEXTENTS 5 )
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile2' INSTANCE 2 );
```

エクステントの動的割当て

この項では、GC_FILES_TO_LOCKS の !*blocks* オプションを使用して、ロック境界内の高水位標からブロックを空きリストに動的に割り当てる方法について説明します。この項では、次の項目について説明します。

- ブロック・データベース・アドレスからロック名への変換
- ALLOCATE EXTENT 構文での !blocks

ブロック・データベース・アドレスからロック名への変換

「PCM インスタンス・ロックの割当て」の章で説明したように、GC_FILES_TO_LOCKS パラメータを設定するための構文は、ブロックのデータベース・アドレスとそれをプロテクトするロック名との間の変換を指定します。構文を次に示します。

GC_FILES_TO_LOCKS = "{ *file_list=#locks* [!*blocks*] [EACH] [:] } ..."

次のエントリは、このバケット内のファイルをプロテクトするために 1000 個の個別のロック名を使用することを示しています。ファイル内のデータは、25 ブロックのグループに分けてプロテクトされます。

```
GC_FILES_TO_LOCKS = "1000!25"
```

ALLOCATE EXTENT 構文での !*blocks*

同様に、!*blocks* パラメータにより、エクステント内で使用可能なブロック数を制御できます。（ブロックを使用可能にするには、ブロックを空きリストに入れておく必要があります。）!*blocks* を使用すると、1 つのエクステント内に割り当てるブロックの割合を指定できます。最大で 1 回に 255 ブロックです。次の例を見てください。

```
GC_FILES_TO_LOCKS = 1000!10
```

この例は、インスタンスにブロックの割当てが必要になるたびに、10 ブロックを使用可能にすることを示しています。

関連項目： 第 15 章「PCM インスタンス・ロックの割当て」

未使用領域の識別および割当て解除

この項では、次の項目について説明します。

- 未使用領域の判別方法
- 未使用領域の割当て解除
- 削除および更新により使用可能となった領域

未使用領域の判別方法

DBMS_SPACE パッケージには、表内の空きリスト・グループの使用済み領域と未使用領域の量を判別するためのプロシージャが含まれています。このプロシージャを使用して、領域を割り当てなおす必要のあるインスタンスを判別することができます。このパッケージは、『Oracle8*i* リファレンス・マニュアル』で説明しているとおり、DBMSUTIL.SQL スクリプトを使用して作成されています。

未使用領域の割当て解除

ALLOCATE EXTENT コマンドを使用して割り当てた未使用領域は、高水位標より下にあるので、割当て解除することはできません。

ただし、領域が高水位標より上に動的に割り当てられたエクステント内部にある場合は、未使用領域の割当てをセグメントから解除することができます。DEALLOCATE UNUSED オプションを ALTER TABLE コマンドまたは ALTER INDEX コマンドとともに使用して、セグメントを高水位標まで切り捨てます。

削除および更新により使用可能となった領域

削除または、更新によって行が短縮されて使用可能になったブロックは、ブロックを削除したプロセスの空きリストおよび空きリスト・グループに戻されます。

第 IV 部

Oracle Parallel Server システムの
メンテナンス手順

18

複数インスタンスの管理

Justice is a machine that, when someone has once given it the starting push, rolls on of itself.

John Galsworthy: *Justice.Act II.*

この章では、パラレル・サーバーのインスタンスを管理する方法について説明します。この章では次の項目を説明します。

- 概要
- Oracle Parallel Server Management
- パラメータ・ファイルによる複数インスタンスの定義方法
- 複数インスタンス用の初期化パラメータの設定
- 必要なロックの量の決定および LM_* パラメータの設定
- 複数インスタンスのデータベース・オブジェクトの作成方法
- インスタンスの起動
- インスタンスの指定
- インスタンスの停止
- パラレル問合せのインスタンスの制限方法

概要

この章では、次の手順を使用して OPS のインスタンスを構成および起動する方法について説明します。

1. パラメータ・ファイルの設定により複数インスタンスを定義します。
2. 複数インスタンスの初期化パラメータを設定します。
3. 必要な PCM ロックおよび非 PCM ロックの数を判断し、LM_* パラメータを設定します。
4. 複数インスタンスのデータベース・オブジェクトを作成します。
5. インスタンスを起動します。

この手順の詳細は、Oracle Parallel Server Management の簡単な説明の後に説明します。

関連項目：『Oracle8*i* 管理者ガイド』の「起動と停止」

Oracle Parallel Server Management

Oracle Parallel Server Management（OPSM）は、OPS の広範で統合化されたシステム管理ソリューションです。OPSM では、異機種環境で稼働する複数インスタンスのデータベースをオープン・クライアント・サーバー・アーキテクチャによって構成および管理できます。

OPSM では、パラレル・データベースの管理以外に、ジョブのスケジュール、イベントの管理、パフォーマンスの監視、パラレル・データベースのチューニングのための統計値の取得ができます。

OPSM の詳細は、『Oracle Parallel Server Management for UNIX 構成ガイド』および『Oracle8*i* Parallel Server セットアップおよび構成ガイド』を参照してください。インストールの手順は、プラットフォーム固有のインストール・マニュアルを参照してください。

パラメータ・ファイルによる複数インスタンスの定義方法

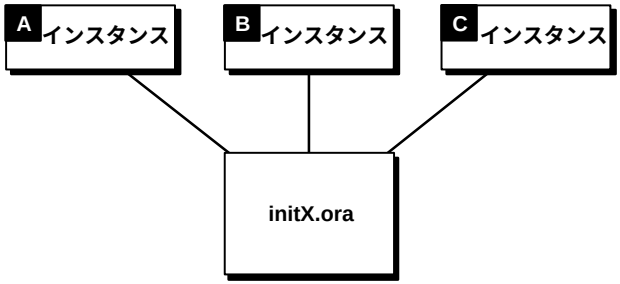
インスタンスが起動すると、Oracle は初期化パラメータ・ファイル内の値を使用して当該インスタンスのシステム・グローバル領域（SGA）を作成します。複数インスタンスの定義には、パラメータ・ファイルをさまざまな方法で使用します。

- 複数インスタンスの共通パラメータ・ファイルの使用方法
- 複数インスタンスの個別パラメータ・ファイルの使用方法
- IFILE によるパラメータ・ファイルの埋込み方法
- PFILE による非デフォルト・パラメータ・ファイルの指定方法

複数インスタンスの共通パラメータ・ファイルの使用方法

図 18-1 に示す全インスタンスの共通パラメータ・ファイルを使用すると、管理が簡単になります。複数ノード間でファイル・システムを共有している場合は、1 箇所だけ変更を加えることにより全インスタンスを更新できます。

図 18-1 共通パラメータ・ファイルを持つインスタンス

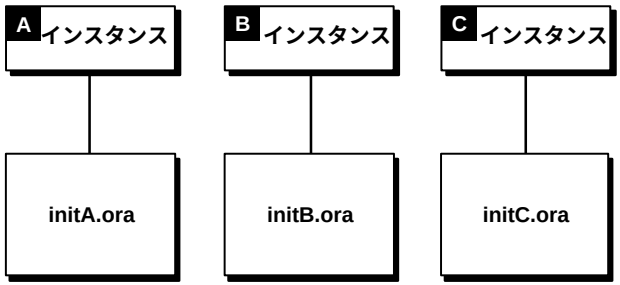


ただし、ほとんどのクラスタ・システムはファイル・システムを共有しません。その場合は、各ノードごとに共通ファイルの物理コピーを別に作成してください。

複数インスタンスの個別パラメータ・ファイルの使用方法

パラメータの多くがインスタンスごとに異なるときは、個別パラメータ・ファイルが便利です。たとえば、規模の異なるマシンがある場合、初期化パラメータを使用して各マシンに合ったサイズの SGA を作成すると、パフォーマンスが大幅に向上する場合があります。

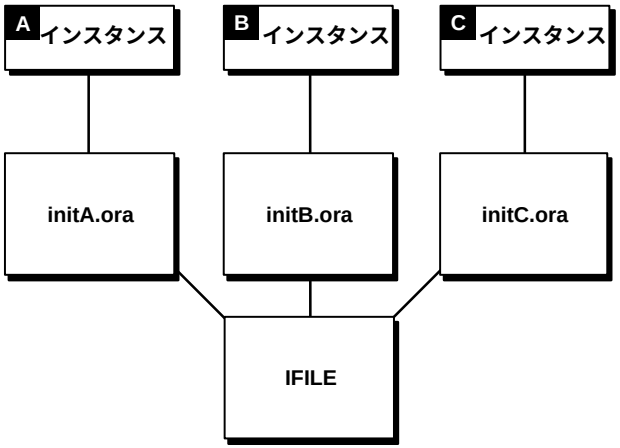
図 18-2 個別パラメータ・ファイルを持つインスタンス



IFILE によるパラメータ・ファイルの埋込み方法

IFILE パラメータを設定することにより、それぞれの個別パラメータ・ファイルは、共通の値を含む追加パラメータ・ファイルを埋め込むことができます。このアプローチを、図 18-3 に示します。

図 18-3 個別パラメータ・ファイルと IFILE を持つインスタンス



Oracle Parallel Server（OPS）では、個別パラメータ・ファイルと共通パラメータ・ファイルのどちらを使用する場合でも、初期化パラメータによってはすべてのインスタンスについて同じ値を持っている必要があります。IFILE パラメータを使用して同じファイルを参照することにより、インスタンスはその固有のパラメータ・ファイルを使用することができ、すべてのインスタンスで同じ値にする必要があるパラメータに対しても正しい値を持つことができます。

次の場合、インスタンスでは個別パラメータ・ファイルを使用する必要があります。

- プライベート・ロールバック・セグメントを使用するすべてのインスタンスは、独自のパラメータ・ファイルを持っている必要がある。ただし、パブリック・ロールバック・セグメントを使用するだけのインスタンスはすべて、同じパラメータ・ファイルを使用できる。
- INSTANCE_NUMBER パラメータまたは THREAD パラメータを指定するインスタンスはすべてそれ自身のパラメータ・ファイルを持っている必要がある。

例

たとえば、ローカル・ノード上の Server Manager セッションは、INIT_OPS1.ORA および INIT_OPS2.ORA という名前の個別パラメータ・ファイルを使用してリモート・ノード上で 2 つのインスタンスを起動できます。

```
SET INSTANCE INSTANCE1;
STARTUP PFILE=INIT_A.ORA;
SET INSTANCE INSTANCE2;
STARTUP PFILE=INIT_B.ORA;
```

"INSTANCE1" および "INSTANCE2" は、これら 2 つのインスタンスの Net8 別名です。これらの別名は、TNSNAMES.ORA で定義されます。

どちらの個別パラメータ・ファイルでも、IFILE パラメータを使用して INIT_COMMON.ORA ファイルからパラメータ値を組み込むことができます。個別パラメータ・ファイルは、次のようにしてこのファイルを参照できます。

INIT_OPS1.ORA の場合

```
IFILE=INIT_COMMON.ORA
INSTANCE_NAME=OPS1
INSTANCE_NUMBER=1
THREAD=1
```

INIT_OPS2.ORA の場合

```
IFILE=INIT_COMMON.ORA
INSTANCE_NAME=OPS2
INSTANCE_NUMBER=2
THREAD=2
```

注意： INSTANCE NAME は、SID と同じにすることをお薦めします。

INIT_COMMON.ORA ファイルでは、双方のインスタンスで同じにする必要があるパラメータ設定値を次のように設定できます。

```
DB_NAME=DB1
SERVICE_NAMES=DB1
CONTROL_FILES=(CTRL_1,CTRL_2,CTRL_3)
GC_FILES_TO_LOCKS="1=600:2-4,9=500EACH:5-8=800"
GC_ROLLBACK_SEGMENTS=10
GC_SEGMENTS=10
LOG_ARCHIVE_START=TRUE
PARALLEL_SERVER=TRUE
```

注意： SERVICE_NAMES は、DBNAME と同じ設定にすることをお薦めします。

それぞれのパラメータ・ファイルでは、CONTROL_FILES パラメータに同じ値を設定する必要があります。これは、すべてのインスタンスが制御ファイルを共有しているなどの理由によります。

共通の初期化パラメータの値を変更するときは、両方の個別パラメータ・ファイルを変更するのではなく、INIT_COMMON.ORA ファイルを修正します。

IFILE の使用方法

IFILE が参照する共通パラメータ・ファイルで同じ値が設定されたパラメータを指定する場合、デフォルト値を使用するパラメータは省略できます。

別のノードで複数の Server Manager セッションを使用してインスタンスを起動する場合、ファイル・システムが共有されていなければ、各ノードにはそれ自身の共通パラメータ・ファイルのコピーが必要になります。

インスタンス別ファイルと共通ファイルでパラメータが重複している場合、または 1 つのファイル内でパラメータが重複している場合は、最後に指定した値によって前の値が上書きされます。したがって、個別パラメータ・ファイルの最後に IFILE パラメータを指定することにより、共通パラメータ値が確実に使用されるようになります。個別ファイルの最初に IFILE を指定すれば、共通値が上書きされます。

パラメータ・ファイル内に IFILE を複数回指定して、複数の共通パラメータ・ファイルを組み込むことができます。他の初期化パラメータとは異なり、IFILE は前の値を上書きしません。たとえば、1 つの個別パラメータ・ファイルに、INIT_COMMON.ORA ファイルと LOG_* および GC_* パラメータ用の個別のコマンド・ファイルを組み込むことが可能です。

```
IFILE=INIT_COMMON.ORA
IFILE=INIT_LOG.ORA
IFILE=INIT_GC.ORA
LOG_ARCHIVE_START=FALSE
THREAD=3
ROLLBACK_SEGMENTS=(RB_C1,RB_C2,RB_C3)
```

IFILE = INIT_LOG.ORA 文が LOG_ARCHIVE_START パラメータの指定の前にきているため、LOG_ARCHIVE_START の値は INIT_LOG.ORA に指定された値を上書きします。IFILE = INIT_GC.ORA が IFILE = INIT_COMMON.ORA の後にきているため、INIT_GC.ORA に指定された個々の GC_* 値は、INIT_COMMON.ORA に指定される値を上書きします。

関連項目： 18-14 ページの「インスタンス番号と起動順序」、6-3 ページの「REDO ログ・ファイル」、および 18-10 ページの「すべてのインスタンスで同じ値にする必要があるパラメータ」

PFILE による非デフォルト・パラメータ・ファイルの指定方法

STARTUP コマンドの PFILE オプションを使用すると、インスタンスの起動時にデフォルト・ファイル以外のパラメータ・ファイルを指定できます。PFILE により指定されるパラメータ・ファイルは、リモート・ノード上のインスタンスに対するものであっても、ローカル・ノードにアクセス可能なディスク上に存在している必要があります。

複数インスタンス用の初期化パラメータの設定

この項では、複数インスタンスに重要な OPS 初期化パラメータについて説明します。

- GC_* グローバル・キャッシュ・パラメータ
- 複数インスタンスのパラメータに関する注意
- すべてのインスタンスで同じ値にする必要があるパラメータ

関連項目：『Oracle8*i* リファレンス・マニュアル』（その他の Oracle 初期化パラメータについて）

GC_* グローバル・キャッシュ・パラメータ

GC（Global Cache）という接頭辞の付いている初期化パラメータは、OPS 専用のパラメータです。これらのパラメータの設定によって、すべてのインスタンス上のデータベース・バッファをプロテクトするグローバル・ロック集合のサイズが決まります。設定した値は、オペレーティング・システムの特定のリソースの使用に影響を与えます。

共有モードで起動する最初のインスタンスは、すべてのインスタンスのグローバル・キャッシュ・パラメータの値を決定します。制御ファイルは、最初のインスタンスの起動時に、GC_* パラメータの値を記録します。

別のインスタンスを共有モードで起動しようとすると、Oracle はインスタンスのパラメータ・ファイルの中のグローバル・キャッシュ・パラメータの値と、すでに使用中のグローバル・キャッシュ・パラメータの値を比較して、一致しない値があればメッセージを出します。正しいグローバル・キャッシュ・パラメータ値でなければ、インスタンスはデータベースをマウントできません。

OPS のグローバル・キャッシュ・パラメータを次に示します。

GC_FILES_TO_LOCKS	データ・ブロック・ロックの制御
GC_ROLLBACK_LOCKS	取消しブロック・ロックの制御
GC_RELEASABLE_LOCKS	ロック数の制御

関連項目： 第 15 章「PCM インスタンス・ロックの割当て」

複数インスタンスのパラメータに関する注意

表 18-1 は初期化パラメータに関連した複数インスタンスの問題についてまとめています。

表 18-1 複数インスタンスの初期化パラメータに関する注意

パラメータ	パラレル・サーバーの注意事項
CHECKPOINT_PROCESS	OPS のデータベースには、単一インスタンス環境の場合よりも多くのデータ・ファイルがあります。チェックポイントを高速化するには、CHECKPOINT_PROCESS パラメータを指定します。
DELAYED_LOGGING_BLOCK_ CLEANOUTS	デフォルト値の TRUE に設定しておくと、インスタンス間での ping が減少します。
DML_LOCKS	0（ゼロ）に設定した場合だけ、すべてのインスタンスで同じにする必要があります。
INSTANCE_NUMBER	このパラメータを指定する場合は、それぞれのインスタンスに一意の値を設定する必要があります。
LOG_ARCHIVE_FORMAT	スレッド番号を含む必要があります。
MAX_COMMIT_PROPAGATION_ DELAY	リモート・インスタンス上にコミットをすぐに反映させるときは、このパラメータの値を変更する必要があります。
NLS_* パラメータ	それぞれのインスタンスについて異なる値を設定できます。
PARALLEL_SERVER	OPS を使用可能にするには、初期化ファイルでこのパラメータを TRUE に設定する必要があります。デフォルトでは FALSE に設定されます。
PROCESSES	すべてのバックグラウンド・プロセスおよびユーザー・プロセスに対応した十分に大きい値を持っている必要があります。オペレーティング・システムによっては、DBWR プロセスを追加できるものもあります。 SESSIONS パラメータおよび TRANSACTIONS パラメータのデフォルトは、PROCESSES パラメータの値から直接的または間接的に導出されます。 デフォルトを使用しない場合は、LCKn やその他のオプションのバックグラウンド・プロセスのために、前述のいくつかのパラメータ値を増やすことをお薦めします。
RECOVERY_PARALLELISM	ロール・フォワードまたはキャッシュ回復フェーズを高速化するには、このパラメータを設定します。
ROLLBACK_SEGMENTS	それぞれのインスタンスについてプライベート・ロールバック・セグメントを指定します。
THREAD	このパラメータを指定する場合は、それぞれのインスタンスについて一意の値をとる必要があります。

関連項目：『Oracle8*i* リファレンス・マニュアル』（各パラメータについて）

すべてのインスタンスで同じ値にする必要があるパラメータ

データベースの作成上重要な、またはある種のデータベース操作に影響を与える特定の初期化パラメータは、OPS のすべてのインスタンスで同じ値を設定しておく必要があります。たとえば、DB_BLOCK_SIZE および CONTROL_FILES の値は、すべてのインスタンスで同じにする必要があります。その他のパラメータは、インスタンスごとに異なる値を設定できます。次の初期化パラメータは、パラレル・サーバー内のすべてのインスタンスで同じ値にする必要があります。

CONTROL_FILES
CPU_COUNT
DB_BLOCK_SIZE
DB_FILES
DB_NAME
DB_DOMAIN
SERVICE_NAMES
DML_LOCKS（「0（ゼロ）」に設定した場合だけ、同じにする必要があります。）
GC_FILES_TO_LOCKS
GC_ROLLBACK_LOCKS
LM_LOCKS（同じ値にすることをお薦めします。）
LM_PROCS（同じ値にすることをお薦めします。）
LM_RESS（同じ値にすることをお薦めします。）
MAX_COMMIT_PROPAGATION_DELAY
PARALLEL_DEFAULT_MAX_SCANS
ROLLBACK_SEGMENTS
ROW_LOCKING

関連項目：『Oracle8*i* リファレンス・マニュアル』（各パラメータについて）

必要なロックの量の決定および LM_* パラメータの設定

LM_* 初期化パラメータの値を設定します。リソース、ロック、およびプロセスの構成は、OPS インスタンスごとに異なります。管理をしやすくするには、すべてのインスタンスでこれらのパラメータが一致している必要があります。

LM_RESS	LM_RESS パラメータは、統合分散ロック・マネージャ（IDLM）がロックできるリソース数を制御します。このパラメータには、DML および DDL（データ・ディクショナリ・ロック）、データ・ディクショナリ・キャッシュ・ロック・プラス・ファイルおよびログ管理ロックに対して割り当てるロック・リソース数などの、非 PCM リソースが含まれます。LM_RESS の値を導出するには、PCM リソースおよび非 PCM リソースの数を追加します。それぞれの計算方法は、第 15 章「PCM インスタンス・ロックの割当て」および第 16 章「リソースおよびロック用の IDLM 容量の確保」に示しています。
LM_LOCKS	ロック数。N は、ノードの合計数です。 LM_LOCKS = LM_RESS + (LM_RESS * (N - 1)) / N
LM_PROCS	プロセス数。PROCESSES 初期化パラメータの値にノード数を掛けた値です。

パラレル DML またはパーティション・オブジェクトで実行する DML を使用する計画がある場合は、値を増加します。

複数インスタンスのデータベース・オブジェクトの作成方法

データベースを作成すると、自動的に単一のインスタンスが OPS 使用禁止状態で起動します。ただし、複数のインスタンスを起動するためには、一定の管理操作を実行する必要があります。これらの作業には次のものがあります。

- 各追加インスタンスに対して追加のロールバック・セグメントを作成する。
- 各追加インスタンスに対してスレッドを追加し、使用可能にする。
- 追加したデータ・ファイルにロックを与える。

排他モードまたは共有モードでこれらの操作を単一のインスタンスで実行できます。

関連項目： 14-4 ページの「追加のロールバック・セグメントの作成」、6-3 ページの「REDO ログ・ファイル」、および 15-17 ページの「必要な PCM ロック数とリソースの合計」

インスタンスの起動

Oracle インスタンスは、OPS が使用可能または使用禁止のどちらであっても起動できます。この項では、その手順について説明します。

- パラレル・サーバーの使用可能化およびインスタンスの起動
- OPS を使用禁止にした状態での起動
- 共有モードでの起動

パラレル・サーバーの使用可能化およびインスタンスの起動

注意： Oracle8 では、SHARED、EXCLUSIVE および PARALLEL の各キーワードは、STARTUP および ALTER DATABASE MOUNT 文で使用できなくなりました。

SQL を使用したインスタンスの起動

1. Oracle8 で OPS を使用可能にするには、初期化ファイルで PARALLEL_SERVER パラメータを TRUE に設定します。デフォルトでは FALSE に設定されます。
2. 必要なオペレーティング・システム固有のプロセスを起動します。

   詳細は、使用しているシステム固有の Oracle マニュアルを参照してください。
3. Cluster Manager ソフトウェアを実行しておいてください。

   詳細は、18-20 ページの「Cluster Manager」を参照してください。
4. SYSDBA または SYSOPER の権限で接続します。

   ```
   CONNECT username/password AS SYSDBA
   ```
5. OPS を使用可能にして実行する場合は PARALLEL_SERVER 初期化パラメータを TRUE に設定し、OPS を使用禁止にして実行する場合は FALSE に設定してあることを確認してください。
6. インスタンスを起動します。

   ```
   STARTUP NOMOUNT
   ```
7. データベースをマウントします。

   ```
   ALTER DATABASE database_name MOUNT
   ```

8. データベースをオープンします。

```
ALTER DATABASE OPEN
```

Server Manager によるインスタンスの起動

注意： Server Manager コマンド STARTUP を OPEN オプションで使用すると、前述の手順 4、5、6 が実行されます。

1. 必要なオペレーティング・システム固有のプロセスを起動します。

   詳細は、使用しているシステム固有の Oracle マニュアルを参照してください。

2. OPS を使用可能にして実行する場合は PARALLEL_SERVER 初期化パラメータを TRUE に設定し、OPS を使用禁止にして実行する場合は FALSE に設定します。

3. Cluster Manager ソフトウェアを実行しておいてください。

   詳細は、18-20 ページの「Cluster Manager」を参照してください。

4. Server Manager を起動します。

5. OPEN オプションを使用して、インスタンスを起動します。

```
STARTUP OPEN database_name
```

OPS を使用禁止にした状態での起動

アーカイブ・モードを変更する（ARCHIVELOG または NOARCHIVELOG）場合は常に OPS が使用禁止になっている必要があります。アーカイブ・モードを変更するには、データベースがマウントされている必要がありますが、オープンしている状態ではいけません。

あるインスタンスが PARALLEL_SERVER を FALSE に設定してデータベースをマウントすると、他のどのインスタンスもデータベースをマウントできません。

インスタンスを排他モードで起動する前に、共有モードで稼働中のインスタンスをすべて停止してください。共有モードで稼働中の 1 つのインスタンスと排他モードで稼働中のインスタンスは異なるものであり、最後に共有モードで稼働したインスタンスが自動的に排他モードに戻ることはありません。

OPS を使用禁止にして起動するインスタンスでは、INSTANCE_NUMBER パラメータでインスタンス番号を指定できます。このインスタンス番号が必要になるのは、インスタンスが挿入と更新の両方を行う場合、およびデータベース内の表で FREELIST GROUPS 記憶領域オプションを使用して、インスタンスに空き領域を割り当てる場合だけです。OPS を使用禁止にして管理動作を実行するためだけにインスタンスを起動する場合は、パラメータ・ファイルの INSTANCE_NUMBER パラメータを省略します。

OPS を使用禁止にして起動するインスタンスでは、1 以外のスレッドを指定し、そのスレッドに対応付けられたオンライン REDO ログ・ファイルを使用できます。

関連項目： 第 17 章「空きリスト・グループによるデータ分割」

共有モードでの起動

OPS の各インスタンスは、共有モードでデータベースをマウントする必要があります。各インスタンスのそれぞれの初期化パラメータ・ファイルでは、SINGLE_PROCESS パラメータが FALSE に設定され、かつ PARALLEL_SERVER パラメータが TRUE に設定されている必要があります。複数インスタンスを共有モードで起動する前に、同じデータベースを共有している各インスタンスにロールバック・セグメントを 1 つ以上作成し、追加インスタンスごとに REDO ログ・ファイル・グループを 2 つ以上含むスレッドを使用可能にしてください。

あるインスタンスが共有モードでデータベースをマウントすると、他のインスタンスも共有モードでデータベースをマウントできますが、排他モードでデータベースをマウントすることはできません。

PARALLEL_SERVER を FALSE に設定すると、デフォルトによってインスタンスは OPS を使用禁止にした状態での起動を試みます。

共有モードでのデータベースのマウント再試行

他のインスタンスが同じデータベースを回復中にインスタンスを起動し共有モードでデータベースをマウントしようとしても、その回復が終了するまでは、新しいインスタンスによるデータベースへのマウントはできません。

インスタンスの起動を繰り返し実行するかわりに、STARTUP RETRY 文が使用できます。RETRY オプションを指定すると、再試行が成功するか、最大再試行回数に達するまで、新しいインスタンスは 5 秒ごとにデータベースをマウントしようとします。次に例を示します。

```
STARTUP OPEN MAILDB RETRY
```

インスタンスがデータベースのマウントを試行する最大回数を設定するには、Server Manager の SET コマンドに RETRY オプションを指定して使用します。RETRY オプションには 10 などの整数かキーワード INFINITE を指定できます。

他のインスタンスによる回復でしかデータベースがオープンできない場合、RETRY は再試行されません。たとえば、あるインスタンスでデータベースが排他モードでマウントされている場合には、他のインスタンスが共有モードで STARTUP RETRY コマンドを試みても機能しません。

インスタンス番号と起動順序

インスタンスが起動すると、インスタンスはインスタンス番号を取得します。このインスタンス番号によって、FREELIST GROUPS 記憶領域オプションで作成された各表用の空きリストの中の 1 つのグループにインスタンスが対応付けられます。

OPS を使用可能または使用禁止にした状態でインスタンスを起動するとき、初期化パラメータ INSTANCE_NUMBER を使用すると明示的にインスタンス番号を指定できます。INSTANCE_NUMBER パラメータが指定されないと、そのインスタンスは自動的に使用可能な最小番号を取得します。

INSTANCE_NUMBER パラメータを指定していないインスタンスのインスタンス番号は、起動した順序で決まります。インスタンスがパラレル・モードで起動した場合、起動番号の制御は困難ですが、インスタンスの停止および再起動後に変更することはできます。

INSTANCE_NUMBER パラメータを使用するインスタンスでは、一意の番号を指定する必要があります。Server Manager コマンド SHOW PARAMETERS INSTANCE_NUMBER を実行すると、各インスタンスが使用しているインスタンス番号を表示します。インスタンス番号が起動順序に基づいて割り当てられている場合は、このコマンドにより、NULL 値が表示されます。

インスタンス停止後、そのインスタンス番号は再び使用可能となります。最初のインスタンスが再起動する前に別のインスタンスが起動すると、2 番目のインスタンスは最初のインスタンスが使用していたインスタンス番号を取得します。

起動順序に基づくインスタンス番号は、INSTANCE_NUMBER パラメータで指定されるインスタンス番号とは別のものです。INSTANCE_NUMBER パラメータを指定してもしなくても、インスタンスがこれらのいずれかの方法でインスタンス番号を取得すると、別のインスタンスは別の方法で同じ番号を取得することはできません。すべての番号は、その取得方法にかかわらず、一意になります。

挿入および更新するのにインスタンスの一貫した割当てが必要な場合は、常に INSTANCE_NUMBER パラメータを使用してください。これにより、インスタンス間でのデータ分割を維持できます。

関連項目： 6-7 ページの「ロールバック・セグメント」、14-4 ページの「追加のロールバック・セグメントの作成」、6-3 ページの「REDO ログ・ファイル」、および第 17 章「空きリスト・グループによるデータ分割」（挿入および更新用の空き領域の割当てについて）

インスタンスの指定

複数インスタンス環境で管理操作を行うときは、必ず正しいインスタンスを指定してください。この項では、インスタンス固有の管理に関連した次の項目について説明します。

- 現行インスタンスとデフォルト・インスタンスの区別
- インスタンスへの SQL 文の適用方法
- インスタンスへの Server Manager コマンドの適用方法
- インスタンス・グループの指定方法
- 複数インスタンス上でのユーザー認証のためのパスワード・ファイルの使用方法

現行インスタンスとデフォルト・インスタンスの区別

Server Manager のコマンドには、Server Manager が現在接続されているインスタンスに適用されるものとデフォルトのインスタンスに適用されるものがあります。

デフォルト・インスタンス	デフォルト・インスタンスは、Server Manager を起動するマシン上にあります。ホスト・コマンドの実行中など、インスタンスに接続しているときに使用できない Server Manager コマンドは、デフォルト・インスタンスに適用されます。
現行インスタンス	現行インスタンスは、CONNECT コマンドによって決定します。インスタンスに接続しているときに使用できる Server Manager コマンドは、現行インスタンスに適用されます。

CONNECT コマンドで接続文字列を指定すると、現行インスタンスをデフォルト・インスタンス以外のインスタンスにすることができます。

リモート・ノードで実行中のインスタンスに対して SET INSTANCE または CONNECT コマンドを使用するには Net8 をインストールする必要があります。

> **関連項目：** Net8 のインストールに関する情報および SET INSTANCE コマンドと CONNECT コマンドで使用される接続文字列に必要な正確なフォーマットについては、使用しているプラットフォーム固有の Oracle マニュアルを参照してください。

インスタンスへの SQL 文の適用方法

インスタンス固有の SQL 文は、現行インスタンスに適用されます。たとえば、ALTER DATABASE ADD LOGFILE 文は、デフォルト・インスタンスまたはすべてのインスタンスに適用されるのではなく、現在接続中のインスタンスにだけ適用されます。

ALTER SYSTEM CHECKPOINT LOCAL は、現行インスタンスに適用されます。それに対して、ALTER SYSTEM CHECKPOINT または ALTER SYSTEM CHECKPOINT GLOBAL は、すべてのインスタンスに適用されます。

ALTER SYSTEM SWITCH LOGFILE は、現行インスタンスにだけ適用されます。ALTER SYSTEM ARCHIVE LOG CURRENT を使用し、グローバル・ログ・スイッチを強制することができます。ALTER SYSTEM ARCHIVE LOG の THREAD オプションにより、特定のインスタンスに対してオンライン REDO ログ・ファイルをアーカイブできます。

インスタンスへの Server Manager コマンドの適用方法

Server Manager の起動時は、入力したコマンドはデフォルト・インスタンスに適用されます。つまり、デフォルト・インスタンスが現行インスタンスでもあります。

SET INSTANCE コマンドを使用して現行インスタンスを設定するまで、コマンドはデフォルト・インスタンスに対して実行されます。現行インスタンスを設定した後は、Server Manager コマンドはすべて現行インスタンスに対して実行されます。

表 18-2 これらのコマンドとインスタンスの関係を示します。

表 18-2 インスタンスへの Server Manager コマンドの適用方法

Server Manager コマンド	対応するインスタンス
ARCHIVE LOG	常に現行インスタンスに対して適用されます。
CONNECT	CONNECT コマンドの中にインスタンスの指定がない場合、デフォルト・インスタンスを使用します。
HOST	現行インスタンスとデフォルト・インスタンスの位置に関係なく、Server Manager セッションを実行中のノードに適用されます。
MONITOR	MONITOR 画面での左上隅の表示から、デフォルト・インスタンスではなく、現行インスタンスが分かります。
RECOVER	特定のインスタンスではなく、データベース自体が対象になります。
SHOW INSTANCE	デフォルト・インスタンスに関する情報を表示します。デフォルト・インスタンスは現行インスタンスと異なる場合があります。
SHOW PARAMETERS	現行インスタンスに関する情報を表示します。
SHOW SGA	現行インスタンスに関する情報を表示します。
SHUTDOWN	常に現行インスタンスに対して適用されます。権限付きの Server Manager コマンド。
STARTUP	常に現行インスタンスに対して適用されます。権限付きの Server Manager コマンド。

注意： 権限付きの Server Manager コマンドの使用時に起動されるセキュリティ・メカニズムは、使用しているオペレーティング・システムによって異なります。オペレーティング・システムのほとんどは、オペレーティング・システムへのログイン時に安全性の高い認証メカニズムを備えています。これらのシステムでは通常、STARTUP および SHUTDOWN が使用できるかどうかは使用しているオペレーティング・システムのデフォルトの権限によって決まります。詳細は、使用しているシステム固有の Oracle マニュアルを参照してください。

SET INSTANCE および SHOW INSTANCE コマンド

Server Manager 文を使用してデフォルト・インスタンスを変更できます。

```
SET INSTANCE instance_path
```

この構文で、*instance_path* は、ユーザー ID とパスワードがない Net8 の有効な接続文字列です。インスタンスと接続している場合、SET INSTANCE を使用する前に切断する必要があります。または、現行インスタンスの接続を切断しないときは 、CONNECT コマンドで *instance_path.* を使用できます。

リモート・ノード上のインスタンスを指定するには、STARTUP コマンド および SHUTDOWN コマンドのかわりに、SET INSTANCE コマンドを使用できます。リモート・インスタンス用のパラメータ・ファイルは、ローカル・ノード上に存在する必要があります。

SHOW INSTANCE コマンドは、デフォルト・インスタンスに対する接続文字列を表示します。Server Manager セッション中に SET INSTANCE を指定しなかった場合、SHOW INSTANCE は値 *local* を戻します。

デフォルト・インスタンスにリセットするには、接続文字列を指定しないで SET INSTANCE を使用するか、または LOCAL を指定します（DEFAULT と指定しないでください。これは "DEFAULT" という名前のインスタンスに対する接続文字列を意味します）。

次の Server Manager 行モードの例は、SHOW INSTANCE と SET INSTANCE の間の関連を示しています。

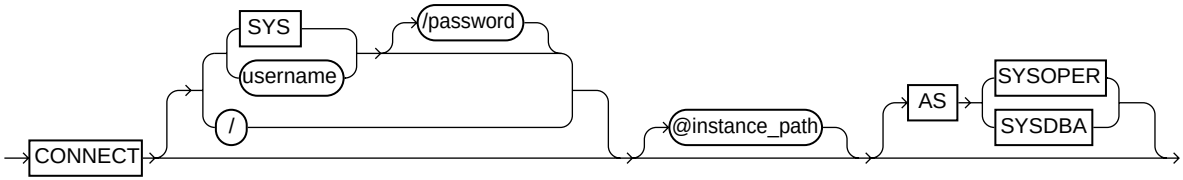
```
SHOW INSTANCE
INSTANCE                        LOCAL

SET INSTANCE node1
Oracle8 Server Release 8.1 - Production
With the distributed, parallel query and Parallel Server options
PL/SQL V8.1 - Production

SHOW INSTANCE
Instance                        node2
```

```
SET INSTANCE
ORACLE8 Server Release 8.1 - Production
With the procedural, distributed, and Parallel Server options
PL/SQL V8.1 - Production

SHOW INSTANCE
INSTANCE                        LOCAL

SET INSTANCE DEFAULT
ORA-06030: NETDNT: connect failed, unrecognized node name
```

CONNECT コマンド

CONNECT コマンドは、Server Manager をデフォルト・インスタンスまたは明示的に指定したインスタンスと対応付けます。Server Manager が接続したインスタンスが現行インスタンスになります。

CONNECT コマンドの構文は次のとおりです。



instance-path は、有効な Net8 接続文字列です。引数 *@instance-path* を省略して CONNECT コマンドを入力すると、（前回 SET INSTANCE で設定された）デフォルト・インスタンスに接続されます。

SYSOPER または SYSDBA で接続すると、インスタンスの起動や停止などの権限を必要とする操作を実行できます。

複数の Server Manager セッションが一度に同じインスタンスに接続できます。1 つのインスタンスに接続しているときは、DISCONNECT コマンドを使用しないで別のインスタンスに接続できます。別のインスタンスに接続すると、Server Manager が最初のインスタンスとの接続を自動的に切断します。

CONNECT *@instance-path* コマンドを使用すると、インスタンスを指定してから Server Manager コマンド、MONITOR、STARTUP、SHUTDOWN、SHOW SGA、SHOW PARAMETERS を使用できます。

> **関連項目：**『Oracle Server Manager User's Guide』（Server Manager コマンドの構文について）、『Oracle8*i* Net8 管理者ガイド』（instance_path の適切な指定方法）、『Oracle8*i* 管理者ガイド』（SYSDBA または SYSOPER 権限での接続について）

Cluster Manager

高い可用性を実現するために、OPS では Cluster Manager（CM）というプラットフォーム固有のソフトウェア・コンポーネントと連携しています。通常、このコンポーネントはハードウェア・ベンダーによって提供されます。

CM は、クラスタ内のさまざまなリソースの状態を監視します。それには、ノード、インターコネクトしたハードウェアとソフトウェア、共有ディスク、Oracle インスタンスなどが含まれます。CM は、インスタンスが起動、停止するときに、自動的に起動、停止します。

リソースの状態が変化すると、CM によってクライアントと Oracle Server に通知されます。Oracle Server は、別のデータベース・インスタンスが CM に登録されたときや CM から切り離されたときについて知る必要があります。データベース・インスタンスは、起動のマウント・フェーズ中に CM に登録されます。

CM の切断は、次の 3 つの理由で発生します。つまり、クライアントが自発的に切断するか、クライアントのプロセスが終了するか、またはクライアント・ノードが停止したりクラッシュしたりすることで切断されます。CM の重要な機能は、障害発生時でもクラスタのグローバル・ビューを提供することです。これにより、各インスタンスが他のすべてのインスタンスを認識して共有ディスクへのアクセスの調整を行うような、OPS データベースの整合性が保証されます。

Oracle は、" ノード・モニター API" というインタフェースを介して CM にアクセスします。このインタフェースは、メンバーが任意にクラスタ全体に分散するプロセス・グループへの抽象リンクを提供します。1 つの OPS インスタンスをマウントすると、LMON プロセスによりこれらのグループのどれかが結合されます。すでに実行されていた同じデータベースのすべてのインスタンスに、その新しい OPS インスタンスの情報が通知されます。これにより、すべてのインスタンスが同期化して、アクティブ・インスタンスの同じビューを所有します。CM は新しいインスタンスについて IDLM レイヤーに通知し、IDLM の再構成も開始します。インスタンスが停止するか異常終了した場合には、CM は残りのインスタンスに通知します。もう一度 CM はインスタンスを同期化して IDLM に通知します。

OPS のクラスタ管理

OPS インスタンスを起動するときは、最初に CM ソフトウェアが実行中であることを確認してください。CM の管理についての詳しい説明は、プラットフォーム固有のマニュアルを参照してください。CM が使用できないか Oracle が CM と通信できない場合は、Oracle からエラー "ORA-29701: Cluster Manager に接続できません。" が表示されます。

クラスタでの複数バージョンの互換性

使用中の Oracle のバージョン番号が 8.1 より大きい限り、複数のバージョンが同じクラスタ上に共存できます。つまり、8.1 よりも古い複数のバージョンの Oracle は、同じクラスタ上で共存できません。たとえば、バージョン 8.0 と 8.1 の OPS データベースは、同じクラスタ上で互換性がありません。

インスタンス・グループの指定方法

別々のインスタンスを論理的にグループ化し、すべての関連インスタンスへのパラレル操作を一度に実行すれば、管理が容易になります。リソースの割当て、パラレル問合せ、その他のパラレル操作などの特定の目的に使用するインスタンスの集合として、インスタンス・グループを定義できます。これにより、効果的なリソースの分割が可能になります。

DBA は、たとえば、ユーザー・プロセスまたは問合せプロセスによるすべてのインスタンスでのリソースの取得を阻止することが必要な場合があります。また、DBA は、OLTP プロセスを実行しているユーザー専用に一定のインスタンスを確保し、パラレル問合せを実行しているユーザーを特定のインスタンス集合にだけ制限することが必要な場合があります。

たとえば、午前 9 時から午後 5 時まではグループ B を使用し、午後 5 時以降はグループ D を使用するように、インスタンス・グループを作成できます。または、通常の OLTP の挿入や更新のためにグループ C を使用し、大きいパラレル・タスクのためにグループ D を使用することで、OLTP のパフォーマンスに影響を与えないようにすることができます。

- インスタンスを起動した後では、グループに対して動的にインスタンスを追加したり削除したりできないので、起動時に可能な限りすべてのグループ構成を定義してください。
- 指定すればいつでも、同じインスタンスを複数のグループのメンバーにすることができます。グループ間でオーバーラップが生じることがあります。
- 多数のグループを定義しておき、必要に応じて使用できます。インスタンス・グループによるオーバーヘッドは小さく、いったんインスタンス・グループを定義してしまえば、それらを参照する必要はありません。

並列度だけを指定すれば、どの特定のインスタンスを使用するかはディスクの親和性および実際に実行中であるインスタンスの数に基づいてシステムによって選択されます。インスタンス・グループを指定することで、パラレル操作に使用するインスタンスを直接指定できます。

問合せを開始したインスタンスが、問合せを実行するインスタンス・グループのメンバーである必要はありません。パラレル・コーディネータが現行インスタンス上で実行します。

インスタンス・グループの指定のしかた

インスタンス・グループを指定するには、グループに関連付ける各インスタンスのパラメータ・ファイル内で INSTANCE_GROUPS 初期化パラメータを設定します。このパラメータによってただちにグループが定義され、現行インスタンスがグループに追加されます。

たとえば、インスタンス 1 では、パラメータを次のように設定できます。

```
INSTANCE_GROUPS = GROUPB, GROUPD
```

インスタンス 3 では、パラメータを次のように設定できます。

```
INSTANCE_GROUPS = GROUPA, GROUPD
```

結果的に、インスタンス 1 およびインスタンス 3 はどちらもインスタンス・グループ D に所属する一方で、他のインスタンス・グループにも所属することになります。

INSTANCE_GROUPS を動的に変更することはできません。

インスタンス・グループの使用方法

インスタンス・グループを使用して、PARALLEL_INSTANCE_GROUP でパラレル操作に使用するグループを識別できます。

PARALLEL_INSTANCE_GROUP のデフォルトは、現在実行中のすべてのインスタンスで構成されるグループです。

特定のインスタンス・グループを一定のパラレル処理のために使用するには、初期化パラメータ・ファイルに次のパラメータを指定します。

```
PARALLEL_INSTANCE_GROUP = GROUPNAME
```

このインスタンスから開始されたすべてのパラレル処理は、ランダムまたはディスクの親和性に基づいた、以前と同じアルゴリズムを使用して、同じグループ内でのみプロセスを発生させます。

PARALLEL_INSTANCE_GROUP パラメータは、ALTER SESSION 文または ALTER SYSTEM 文によって変更する動的パラメータです。このパラメータを使用して、1 つのインスタンス・グループのみを参照できます。デフォルトでは、現行のすべてのアクティブ・インスタンスを含んだデフォルト・グループを参照するように設定されています。現在実行しているインスタンスが、特定処理のために使用されるインスタンス・グループの一部である必要はありません。

> **関連項目：**『Oracle8*i* リファレンス・マニュアル』（初期化パラメータおよびビューについて）

インスタンス・グループのメンバー一覧の表示のしかた

別のインスタンス・グループに所属するメンバーを表示させる必要があるときは、GV$ グローバル動的パフォーマンス・ビュー GV$PARAMETER に対して問合せを行います。INSTANCE_GROUPS パラメータ名の全エントリを参照してください。

インスタンス・グループ例

この例では、インスタンス 1 の初期化パラメータ・ファイルに次のパラメータが設定されています。

```
INSTANCE_GROUPS = GA, GB
PARALLEL_INSTANCE_GROUP  GB
```

インスタンス 2 の初期化パラメータ・ファイルには、次のパラメータが設定されています。

```
INSTANCE_GROUPS = GB, GC
```

```
PARALLEL_INSTANCE_GROUP = GC
```

インスタンス 1 では、次に示された文を入力すると、GB のインスタンスが使用されます。インスタンス 1 に 2 つのサーバー・プロセスが発生し、インスタンス 2 にも 2 つのサーバー・プロセスが発生します。

```
ALTER TABLE TABLE PARALLEL (DEGREE 2 INSTANCES 2);
SELECT COUNT(*) FROM TABLE;
```

インスタンス 1 で次に示された文を入力すると、GC が使用されます。インスタンス 2 だけに 2 つのサーバー・プロセスが発生します。

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = 'GC';
SELECT COUNT (*) FROM TABLE;
```

インスタンス 1 で次に示された文を入力すると、デフォルト・インスタンス・グループ（現在実行中のすべてのインスタンス）が使用されます。インスタンス 1 に 2 つのサーバー・プロセスが発生し、インスタンス 2 にも 2 つのサーバー・プロセスが発生します。

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = '';
SELECT COUNT(*) FROM TABLE;
```

複数インスタンス上でのユーザー認証のためのパスワード・ファイルの使用方法

OPS 上の複数のインスタンスを実行中に、データベース管理を行っているユーザーを認証するためにパスワード・ファイルを使用できます。この場合、各インスタンス用の環境変数は同じパスワード・ファイルを指している必要があります。同様に、各インスタンスに対する REMOTE_LOGIN_PASSWORDFILE 初期化パラメータは適切で、かつ同じ値に設定されている必要があります。

> **関連項目：**『Oracle8*i* 管理者ガイド』（REMOTE_LOGIN_PASSWORDFILE パラメータについて）。パスワード・ファイルの正確な名前およびオペレーティング・システムにそのパスワード・ファイル名を指定するときに使用される環境変数名については、使用しているシステム固有の Oracle マニュアルを参照してください。

インスタンスの停止

インスタンスを停止するには、次の手順に従ってください。

1. SYSDBA に接続します。

   ```
   CONNECT username/password AS SYSDBA
   ```

2. データベースをクローズします。

   ```
   ALTER DATABASE database_name CLOSE
   ```

3. データベースをディスマウントします。

   ```
   ALTER DATABASE database_name DISMOUNT
   ```

また、Server Manager コマンド SHUTDOWN も使用できます。このコマンドは、現行インスタンスに対してこの 3 つの手順をすべて実行します。

OPS では、インスタンスを停止しても稼働中の他のインスタンスの動作を妨げることはありません。

共有モードでマウントしたデータベースを停止するときは、パラレル・サーバー内のインスタンスをすべて停止します。OPS を使用すると、異なるノードでインスタンスを並列に停止できます。インスタンスが異常停止すると、Oracle はそのインスタンスで実行中のすべてのユーザー・プロセスをデータベースから強制的にログオフします。ユーザー・プロセスが現在そのデータベースにアクセス中の場合、Oracle はそのアクセスを終了させ、"ORA-1092: Oracle インスタンスが終了し強制的に切り離されました " というメッセージを通知します。インスタンスが停止したときにユーザー・プロセスがデータベースにアクセス中でない場合、Oracle は、次のコール時または Oracle への要求実行時に "ORA-1012: ログオンされていません " というメッセージを出します。

NORMAL または IMMEDIATE での停止後のインスタンスの回復は不要です。ただし、SHUTDOWN ABORT コマンドの実行後またはインスタンスが異常終了した後はインスタンスの回復が必要です。実行中のインスタンスの SMON プロセスが、停止したインスタンスに対してインスタンス回復を実行します。他に実行中のインスタンスがない場合は、次にデータベースをオープンするインスタンスが、それを必要としているインスタンス用にインスタンス回復を行います。

1 つのインスタンスに複数の Server Manager セッションが同時に接続されている場合、そのインスタンスを正常に停止するには、1 つを除いたすべてのセッションの接続を切断しておく必要があります。複数の Server Manager セッション（またはその他のセッション）がインスタンスに接続されているときは、SHUTDOWN コマンドの IMMEDIATE オプションまたは ABORT オプションを使用してそのインスタンスを停止できます。

関連項目：『Oracle8*i* 管理者ガイド』の「起動と停止」（SHUTDOWN コマンドの各オプションの説明）

パラレル問合せのインスタンスの制限方法

パラレル問合せ機能にとって OPS は必須ではありませんが、パラレル問合せにはパラレル・サーバーにだけ適用されるいくつかの側面があります。

CREATE TABLE、ALTER TABLE、CREATE CLUSTER、ALTER CLUSTER コマンドの PARALLEL 句の INSTANCES キーワードを使用すると、表をパラレル問合せで走査したときに、OPS の使用可能な全インスタンスのバッファ・キャッシュに表やクラスタが分割されるように指定できます。

使用可能な全インスタンス間で、表を動的に分割しないようにする場合は、ALTER SYSTEM コマンドを使用して、走査またはキャッシングに関与できるインスタンスの数を指定します。

起動時にパラレル問合せ処理に関与するインスタンス数を指定する場合は、初期化パラメータ PARALLEL_MAX_INSTANCES の値を指定します。

パラレル問合せ処理に使用可能なインスタンス数を動的に制限するときは、ALTER SYSTEM コマンドを使用します。たとえば、パラレル・サーバーで 10 個のインスタンスが実行中で、そのうちの 8 つだけをパラレル問合せ処理に関与させ、残りの 2 つのインスタンスはそれ以外の用途専用とする場合、次のコマンドを実行します。

```
ALTER SYSTEM SET SCAN_INSTANCES = 8;
```

これ以降は、表の定義で INSTANCES キーワードの値として 10 が指定されている場合、その表は 10 あるインスタンスのうち 8 つだけについて問合せサーバーで走査されます。この例では、Oracle は最初の 8 つのインスタンスを選択します。パラレル問合せ処理に関与させないインスタンスに対しては初期化パラメータ PARALLEL_MAX_SERVERS を 0(ゼロ) に設定できます。

表をキャッシュに格納するインスタンス数を制限するときは、次のコマンドを実行します。

```
ALTER SYSTEM SET CACHE_INSTANCES = 8;
```

これ以降は、表の定義で INSTANCES キーワードに 10 が設定され、CACHE キーワードが指定されていれば、その表は 10 個の使用可能なインスタンスのバッファ・キャッシュのうちの 8 つに均等に分割されます。

関連項目： 18-21 ページの「インスタンス・グループの指定方法」および『Oracle8*i* リファレンス・マニュアル』（各パラメータについて）。パラレル問合せまたはパラレル実行の詳細は、『Oracle8*i* チューニング』を参照してください。

PARALLEL_SERVER_INSTANCES

PARALLEL_SERVER_INSTANCES パラメータでは、OPS 環境で構成されたインスタンスの数を指定します。このパラメータを使用すると、インスタンスの数によって決まる SGA 構造のサイズを管理するための情報が Oracle に与えられます。このパラメータを設定すると、システムによって使用可能メモリーがより効率的に利用されます。

インスタンス登録およびクライアント / サービス接続

インスタンス登録は、3 つの異なるプロセスに関係しています。これらのプロセスは、次の各項目を登録します。

- PMON によるサービス・ハンドラ
- 専用サーバー・ハンドラ
- インスタンス登録

PMON は、MTS ディスパッチャに TNS リスナーを登録することで、サービス・ハンドラ登録を実行します。また、PMON は TNS リスナーと動的に通信することによって専用サーバー・ハンドラも登録します。PMON では、新しい専用プロセスの起動方法を TNS リスナーに通知することで、専用サーバー・ハンドラを動的に登録することもできます。

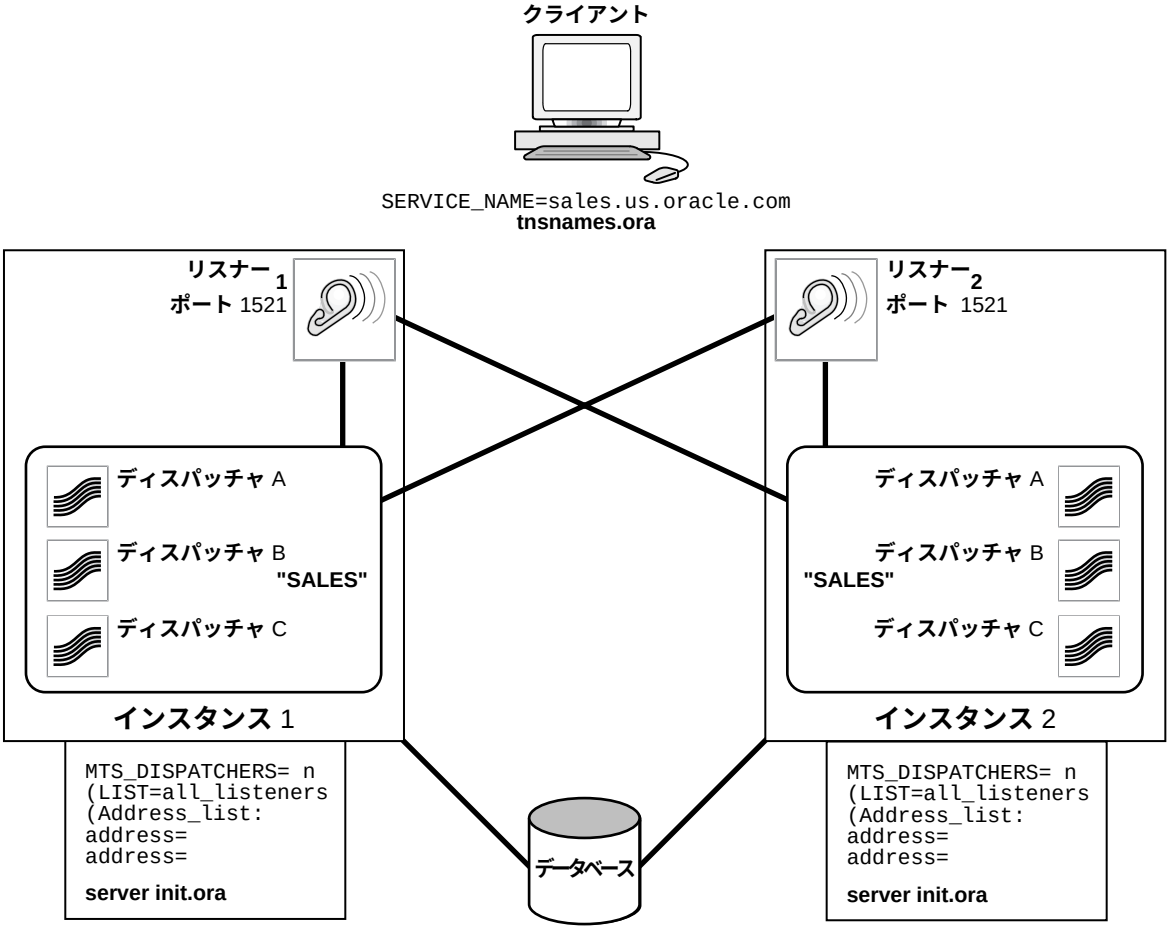
専用サーバー・ハンドラを登録するのは、インスタンスが登録済みでそのインスタンスの TNS リスナーが専用サーバーを作成するだけの場合です。これは接続時フェイルオーバーのプロセスの一部です。起動したプロセスが使用可能になり、専用クライアント・サーバー接続を処理します。

インスタンス登録は、関連するすべてのリスナー・プロセス間で、インスタンス固有の情報を記録することを意味します。最も重要なことは、ある特定のインスタンスの負荷についての情報が同じサービスの他のリスナー・プロセス間で記録されることです。

クライアントからサービスへのアクセス方法

クライアントは、TNSNAMES.ORA ファイルに指定されたとおりにリスナーと交信することによってサービスに接続します。クライアントは、TNSNAMES.ORA からリスナーに CONNECT_DATA を渡します。この情報によって、クライアントが接続するサービスがリスナーに伝わります。リスナーは、クライアントを適切なハンドラにリダイレクトするか、クライアント用の専用サーバーを新しく作成します。

図 18-4 クライアントとサービスの接続



クライアント対サービス接続の構成

クライアント対サービス接続または " クライアント対インスタンス " 接続を構成するときは、INIT.ORA のエントリをこの項の説明どおりにしてください。またこの項では TNSNAMES.ORA へのエントリについても説明します。TNSNAMES.ORA の構成の詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。

この項では、これらのファイルのエントリが次の OPS 機能に与える影響について説明します。

- データベース・インスタンス登録
- 接続時フェイルオーバー
- クライアント・ロード・バランシング
- 接続ロード・バランシング

データベース・インスタンス登録

データベース・インスタンス登録は、インスタンスをリスナーに " 登録する " プロセスです。複数のインスタンスを構成し、同じサービスのローカル・リスナーとリモート・リスナーに登録できます。登録を設定するときは、次の INIT.ORA パラメータを構成してください。

```
SERVICE_NAMES = SALE.US.ORACLE.COM
LOCAL_LISTENER = protocol address
MTS_DISPATCHERS = (LIST = ... )
INSTANCE_NAME = instance name
```

SERVICE_NAMES には、この例のように完全修飾されたサービス名を入力します。INIT.ORA の SERVICE_NAMES に入力する値は、TNSNAMES.ORA のエントリと同じにする必要があります。

注意： SERVICE_NAMES パラメータを設定するかわりに使用できる設定方法があります。詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。

接続時フェイルオーバー

接続時フェイルオーバーは、クライアントが最初のリスナーへの接続に失敗した場合に、2 番目のリスナーへの接続を試みることを意味します。クライアントが 2 番目のリスナーへの接続の試行をどのように実行するかは、TNSNAMES.ORA 内のアドレス・リストにリスナー・アドレスをどのように入力するかによって制御します。

そのためには、リスナー・アドレスを TNSNAMES.ORA に入力するときに、クライアントがクライアント対サービス接続を試行する順番に入力します。また、TNSNAMES.ORA の FAILOVER パラメータを ON に設定します。

接続時フェイルオーバーは、クライアントがリスナーへの接続に成功するまで続きます。

注意： 接続時フェイルオーバーを実装しても、静的サービス設定パラメータは使用できません。したがって、Oracle Enterprise Manager（OEM）と接続時フェイルオーバーを同時に実装することはできません。OEM の動的検出機能は、静的な設定パラメータを使用して、サービス内でアクティブな接続を検索します。

関連項目： 透過的アプリケーション・フェイルオーバーの詳細は、『Oracle8*i* チューニング』を参照してください。

クライアント・ロード・バランシング

クライアント・ロード・バランシングは、データベースにサービスを提供するすべてのリスナーの間で、クライアント接続を均衡化することを意味します。この機能を使用可能にするには、TNSNAMES.ORA の LOAD_BALANCE パラメータを ON に設定します。

クライアント・ロード・バランシングは、処理負荷を均衡化するために複数のインスタンスにサーバー・プロセスを分散させる " パラレル問合せのロード・バランシング " とは異なります。パラレル問合せのロード・バランシングについては、18-30 ページの「パラレル実行のロード・バランシング」で詳しく説明します。

接続ロード・バランシング

接続ロード・バランシングでは、使用可能なすべてのノード、インスタンス、およびそれらのディスパッチャの間で、クライアント接続を均等に分配しようとします。接続の分配は、各ノードの処理負荷、および各インスタンスと各ディスパッチャでのアクティブな接続の数に基づいて行われます。接続ロード・バランシングは、マルチスレッド・サーバーを構成するときに自動的に使用可能になります。

注意： この機能を使用できるのは、マルチスレッド・サーバーを使用可能にしたときに限られます。

パラレル実行のロード・バランシング

パラレル実行のロード・バランシング機能では、サーバー・プロセスを複数のインスタンスに広げて負荷を均衡化します。これにより、複数インスタンスでの PDML 操作およびパラレル実行のロード・バランシングが改善します。

この自動化された並列性の特定の側面を調整することはできませんが、データベース・スケジューラの値を調整して、自動化されたパラレル問合せのロード・バランシング・アルゴリズムに影響を与えることはできます。

MPP システムでは、非類似ノードへの移入の前に、Oracle によって類似ノードへの移入が行われます。一般的に、類似ノードの負荷は非類似ノードよりも約 10 ～ 15 パーセント高くなります。

ロード・バランシング機能では、複数インスタンス間での通信にベンダー固有の Cluster Manager ソフトウェアを使用します。

関連項目： 並列性とその使用方法の詳細は、『Oracle8*i* チューニング』を参照してください。

Managed Standby およびスタンバイ・データベース

スタンバイ・データベースを使用することで、OPS システムを障害から守ることができます。スタンバイ・データベースの管理を簡単にするには、『Oracle8*i* バックアップおよびリカバリ・ガイド』で説明しているように Managed Standby 機能の使用を検討してください。

19

チューニングによるパフォーマンスの最適化

Last of the gods, Contention ends her tale.

Aeschylus, *Antigone*

この章では、Oracle Parallel Server（OPS）のチューニングの概要について説明します。次の項目について説明します。

- 一般ガイドライン
- 競合
- 高い可用性を得るためのチューニング

関連項目： 18-2 ページの「Oracle Parallel Server Management」（OPSM を使用して複数のインスタンスを管理する方法について）

一般ガイドライン

この項では、次の項目について説明します。

- 概要
- 全インスタンスの統計の保管
- 保管する統計
- 一度に 1 つのパラメータを変更する

概要

経験を積むにしたがって、アプリケーションのロールアウトおよびテストを行う前に、OPS アプリケーションに関するほとんどの問題を予測できるようになります。それには、この章で説明する方法を使用してください。さらに、いくつかのチューニング可能なパラメータを使用することでシステムのパフォーマンスが向上します。パラメータをチューニングすることでシステムのパフォーマンスを向上できますが、統合分散ロック・マネージャ（IDLM）のロックの競合が発生する可能性について十分に分析しないことによって起こる問題は解決できません。

単一インスタンスのアプリケーションで使用されるテクニックは、OPS アプリケーションの調整時にも有効です。しかし、バッファ・キャッシュ、共有プール、および全ディスク・サブシステムを効果的に調整することも重要です。OPS では、ユーザーが理解する必要のある追加パラメータと、ユーザーが収集する必要のある OPS 固有の統計情報を提供しています。

注意： ロックは動的にマスター化および再マスター化されるため、インスタンスを一定の順序で起動する必要はありません。

統計情報を収集して OPS のパフォーマンスを監視するときは、次の一般ガイドラインに従うことでシステムのデバッグおよび監視の作業をより簡単で正確にできます。

関連項目： マスター化の詳細は、8-9 ページの「ロックのマスター化」を参照してください。

全インスタンスの統計の保管

インスタンスの監視は全インスタンスについて同じ方法で行い、統計は各インスタンスごとに保管することが重要です。採用した分割方針の結果、対称性が著しく損なわれる場合には特に重要です。インスタンスを監視することで最も負荷が大きいノードがわかり、システム分割のパフォーマンスをテストします。

関連項目： 統計情報のメンテナンスの詳細は、『Oracle8*i* チューニング』を参照してください。

保管する統計

データベース内で監視する最適の統計は、"V$" 表や "X$" 表など、SGA 内に保管されている統計です。一定期間を通してこれらのビューの " スナップショット " を取るのが効果的です。さらに、オペレーティング・システムの統計情報の正確な履歴を維持して、監視やデバッグの処理に役立てます。これらの統計情報の中で最も重要なものは、CPU 使用率、ディスク I/O、仮想メモリーの使用率、ネットワークの使用率、およびロック・マネージャの統計情報です。

一度に 1 つのパラメータを変更する

ベンチマークや容量計画を実施するときに重要なことは、システムのセットアップへの変更を効果的に管理することです。それぞれの変更を文書に記録し、その変更の影響を有効に定量化することにより、システムとアプリケーションのメカニズムの概要をつかみ、理解することができます。システムのデバッグを行うときや、ハードウェア・リソースの追加が必要かどうかを決めるときには、特に重要です。パフォーマンス・プロジェクトの測定およびチューニング・フェーズでは系統的アプローチをとる必要があります。こうしたアプローチは、一見時間がかかりすぎるようにみえますが、長期的にみれば時間とシステム・リソースが節約できます。

競合

この項では、次の項目について説明します。

- ロック変換の検出
- アプリケーション内のロック競合位置の特定

ロック変換の検出

ロック変換が数多く起きているかどうかを検出するには、"V$" 表を調べてください。これを調べると、システムが現在アップグレードまたはダウングレードしているロックがわかります。ロックの競合が発生しているかどうかを初期段階で判別するのに最適なビューは、V$LOCK_ACTIVITY および V$SYSSTAT です。

ある期間内のロック変換の回数を判別するには、V$LOCK_ACTIVITY 表に問合せを実行します。これにより、IDLM の最大ロック変換率に達しているかどうかを判別できます。最大ロック変換率に達している場合は、アプリケーションを再分割してボトルネックを取り除く必要があります。この状況では、CPU、ディスク・ドライブ、メモリーなどのハードウェア・リソースを追加しても、システム・パフォーマンスを大幅に向上させることは困難です。

注意： 最大ロック変換率は、使用しているプラットフォームでの IPC メカニズムの実現方法に応じて異なります。

ロック変換数が多すぎるかどうかを判別するには、読込みまたは変更のいずれかの目的でデータ・ブロックにアクセスするときに、トランザクションで必要なロック変換操作の回数を算出します。これを確認するには、V$SYSSTAT 表に問合せを実行します。

このようにしてロック・ヒット率を計算し、キャッシュ・ヒット率と比較することができます。算出された値は、ロック変換を要求しないデータ・ブロックのアクセス回数であり、

データ・ブロックの合計アクセス回数と比較されます。ロック・ヒット率は、次のようにして計算します。

$$\frac{consistent_gets\ -\ global_lock_converts_(async)}{consistent_gets}$$

この率を計算する SQL 文は次のとおりです。

```
SELECT (b1.value - b2.value) / b1.value ops_ratio
  FROM V$SYSSTAT b1, V$SYSSTAT b2
WHERE b1.name = 'consistent gets'
  AND b2.name = 'global lock converts (async)';
```

この率が 95% 以下になると、追加ノードを加えても、パフォーマンスの最適なスケール変更が行われない可能性があります。

PCM ロック変換が過剰であるかどうかは、ping/ 書込み率によってもわかります。ping/ 書込み率は次のようにして決定されます。

$$ping_write_ratio = \frac{DBWR_cross_instance_writes}{physical_writes}$$

関連項目： 15-14 ページの「PCM ロックのチューニング」

アプリケーション内のロック競合位置の特定

アプリケーションのパフォーマンスが悪く、その主な原因が過剰なロック変換数にあると判断した場合は、その問題の原因となっているトランザクションおよび SQL 文を識別してください。ロック競合が過剰な場合、OPS のセットアップ時に生じる 3 つの問題領域のいずれかが原因となっている可能性があります。これらの領域は、次のとおりです。

- 共通リソース獲得の競合
- ロックの欠如
- 制約

過剰なロック変換率 : 共通リソース獲得の競合

この項では、共通リソース獲得の競合に対応する過剰ロック変換率について説明します。

OPS 内で、システムが期待どおりに機能しない場合があります。その原因として、データベースのセットアップ時やアプリケーション設計プロセスで、そのアプリケーションを実行している時間全体ですべてのインスタンスがアクセスする必要があるブロックを見落としたことが考えられます。このため、システム全体が、このリソースについて " 単一スレッド " を効率良く実行することが必要になります。

この問題は、すべてのユーザーが 1 つのリソースへの排他アクセスを要求する単一インスタンスの場合でも起こります。たとえば、在庫システムの全ユーザーが共通の在庫レベルを変更する場合などがあります。

OPS アプリケーションで起こる競合ポイントで最も一般的なのが、データベース・ブロックの共通集合獲得上の競合に関係しています。これが発生しているかどうかを判断するために、V$ 表の他の集合を問い合せることができます。それらは、V$BH 表、V$CACHE 表および V$PING 表です。

これらの表は基本的には同じデータを扱いますが、V$CACHE および V$PING は、追加データ・ディクショナリ表を結合して使いやすくするビューとして作成されたものです。これらの表およびビューでは、インスタンスの SGA 内の現行のデータ・ブロックの状態を調べます。これを使用すると、ノード間でブロックが何回 ping されたか、また同一データ・ブロックの異なるバージョンが SGA 内にいくつ存在するかを確認するための問合せを作成することもできます。これらの機能の両方を使用して、単一データベース・ブロックへの過剰な単一スレッドが起こっているかどうかを判別できます。

注意 : GV$BH ビュー、GV$CACHE ビュー、および GV$PING ビューも使用できます。これらのビューを使用すると、すべてのインスタンスで問合せを実行できます。

ブロック競合の発生率の高い最も一般的な領域は次のとおりです。

- **表に挿入するための空き領域が必要となる INSERT 文による空きリストの競合。**ほとんどの場合は、V$PING を問い合せて SGA 内の 1 つのブロックに複数のコピーがあることに気づくことで、この競合を認識できます。ブロックがファイル内の 2 番目のブロックであれば、空きリストの競合が発生している可能性があります。実際には、空きリスト・グループおよび複数の空きリストを使用することで、この問題を解決できる場合があります。また、空きリストの競合は単一インスタンス・システム、特に多くの CPU を持つ SMP マシンで起こる場合があります。この問題は V$WAITSTAT 表に問合せを実行することにより判別できます。
- **同じ領域ヘッダー管理ブロックを共有するトランザクションを獲得しようとして起こるセグメント・ヘッダーの競合。**これは、パラレル問合せスレーブが一時表領域からソート領域を割り当てるときに、パラレル索引の作成中に起こることがあります。OPS で同時領域管理を行う各セグメントでは、領域管理アクティビティの調整専用の分散ロックが約 9 個必要になります。

- **索引付きの表に対して実行される INSERT 文および DELETE 文による索引の競合。** V$PING を問い合せることにより、索引の第 1 エクステント内のいくつかのデータ・ブロックの複数コピーが両方の文について SGA 内に存在し、同時にこれらのブロックがブロック ping を多数回受けていることがわかります。この問題はチューニングしても解決できません。かわりに、この索引へのすべてのアクセスを単一のインスタンスにローカライズします。それには、読取りアクセスと書込みアクセスの両方が含まれます。

  これには、この表を単一インスタンスに変更するトランザクションのルーティングが必要なだけでなく、システムの非対称的な実行を行う必要があります。これができない場合は、表の分割およびデータ依存ルーティング方針の使用について検討してください。

ロックの欠如による過剰なロック変換率

SELECT 文、UPDATE 文および DELETE 文に対してランダム・アクセスを行う表では、各ノードは PCM ロックのアップグレードおよびダウングレードをいくつか実行する必要があります。これらのロック変換操作でディスク I/O が必要な場合は、コストが高くなり、パフォーマンスに不利な影響がでます。

しかし、ロック変換の多くがロックを変換するだけで、ディスク I/O が必要ない場合は、パフォーマンスを改善できます。多くの場合、これは I/O なしの ping、または I/O を必要としない ping と呼ばれます。ロック変換を I/O なしで行うことができる理由は、単一インスタンスの場合と同様に、データベースが DBWR を使用して SGA からデータ・ブロックのエージングを行うことができるからです。これが起こる可能性があるのは、その表が SGA のサイズに比較して非常に大きいときだけです。表が小さい場合は、ディスク I/O が必要になることがあります。これは SGA からエージングされにくくなるためです。

ランダム・アクセスが起こる小さな表でも、積極的なチェックポイント機能を使用することでパフォーマンスを向上できます。それには、表 PCTFREE の値を増やし、PCM ロックが管理するデータ・ブロックの数を減らすことによって、データ・ブロックに格納される行数を減らします。PCM ロックごとに管理されている行数を調整するプロセスは、ロック変換が最少となるか、ハードウェア構成から PCM ロックがなくなるまで実行できます。

IDLM によって管理されている PCM ロックの数は、無限のリソースではありません。各ロックには各 OPS ノード上にメモリーが必要となるため、このリソースはすぐに使い果たされてしまうことがあります。OPS 環境では、複数の PCM ロックを追加すると、OPS インスタンスの再起動または回復に要する時間が長くなります。高い可用性が必要な環境では、インスタンスの再起動や回復に要する時間から、実際に割当て可能な PCM ロックの最大数を判別できる場合があります。

制約による過剰なロック変換率

特定の状況では、特定の制約によって、過剰なロック変換率を下げることはできません。サイズが数ギガバイトもある大きな表またはクラスタ、索引では、ロック変換率の上昇（たとえ false ping であっても）を防ぐのに十分な PCM ロックを割り当てることができなくなります。これは主に、十分なロックを割り当てるのに物理的な制限があるためです。この状況

では、単一の PCM ロックで 1,000 個以上のデータ・ブロックを効率よく管理できる場合があります。

ランダム・アクセスが起こっている場合、1 つのデータ・ブロックを獲得するのに競合が発生しない場合でも、ロック変換が実行されます。この状況では、ロック数をチューニングしても、パフォーマンスが向上しにくくなります。これは、必要なロック数が IDLM で実際に作成されるロックの回数を大幅に上回るためです。

このような場合には、これらのデータベース・オブジェクトへのアクセスを制限するか、パーティションによる解決法を探るか、いずれかを行ってください。

高い可用性を得るためのチューニング

OPS のノード上で Oracle インスタンスの障害が発生した場合、その原因となっている問題は、障害が発生したノードをリブートする必要がある場合としなくてもよい場合があります。障害が発生したノードでリブートまたは再起動が必要な場合、他のノードの回復プロセスには時間がかかります。完全回復が必要な場合、この回復プロセスは 3 つの個別フェーズにより実行されます。

- エラーの検出
- IDLM ロックの回復および再マスター化
- 障害インスタンスの回復

エラーの検出

回復の第 1 フェーズでは、障害が発生しているノードまたは OPS インスタンスを検出します。ノード全体の障害または 1 つの Oracle インスタンスの障害は、オペレーティング・システムのノード管理機能によって検出されます。

IDLM ロックの回復および再マスター化

あるノードが完全に障害に陥った場合、残りのノードは、障害が発生したノードが保持しているロックを再マスター化する必要があります。障害が発生していないインスタンスでは、回復が終了するまですべてのデータベース処理は停止します。IDLM の処理を高速化するには、PCM ロック回数を最小にすることが重要です。これは結果的に、データベースのパフォーマンスとデータベースの可用性という 2 つの要件のトレード・オフとなります。

障害インスタンスの回復

IDLM がすべてのロック情報を回復すると、残りのノードの 1 つが障害インスタンスの IDLM インスタンス・ロックに対して排他ロックを取得します。このノードによって、障害インスタンスが、その REDO ログのロールフォワード / ロールバックワード回復を実行できるようになります。これは、SMON バックグラウンド・プロセスにより実行されます。このプロセスの実行に必要な時間は、回復する REDO ログの数によって決まります。これは、実行時のチェックポイント数の関数です。これも、チェックポイント数の少ない方が好都合なシステム実行時のパフォーマンスと、システムの可用性要件とのトレードオフです。

関連項目： 22-5 ページの「Oracle インスタンス回復のフェーズ」

20

キャッシュ・フュージョンとインスタンス間パフォーマンス

この章では、キャッシュ・フュージョンによる Oracle Parallel Server の reader / writer 競合の解決について説明します。ほとんどの種類のシステムとアプリケーションに適用するキャッシュ・フュージョンとその利点について概要を説明します。また、この章では OPS 関連の統計情報とキャッシュ・フュージョン関連の統計情報についても説明し、これらの統計情報を使用してパフォーマンスの監視と調整を行う方法について多くの手順を示します。

この章では、次の項目について説明します。

- キャッシュ一貫性矛盾を解決する際のキャッシュ・フュージョンの役割
- キャッシュ・フュージョンによる一貫性読込みブロックの作成方法
- キャッシュ・フュージョンによる拡張性の向上
- OPS 用のインターコネクトおよびインターコネクト・プロトコル
- パフォーマンス期待値
- キャッシュ・フュージョンとインスタンス間パフォーマンスの監視

注意： キャッシュ・フュージョン処理の概要は、5-6 ページの「キャッシュ・フュージョン処理およびブロック・サーバー・プロセス」を参照してください。

キャッシュ一貫性矛盾を解決するときのキャッシュ・フュージョンの役割

データ・ブロックのインスタンス間競合とその結果として生じるキャッシュ一貫性の問題は、OPS のパフォーマンス上の主な問題です。ほとんどの場合、適切なパーティション化によって大部分の競合問題が解決されます。

しかし実際には、ほとんどのパッケージ・アプリケーションは効果的にパーティション化されていないか、限られた範囲でしかパーティション化されていません。このようなアプリケーションでは、次の 3 種類のインスタンス間競合が発生します。

- Reader / Reader
- Reader / Writer
- Writer / Writer

reader / writer 競合は、あるインスタンスがデータ・ブロックを一貫性モードで読み取る必要があり、そのインスタンスのキャッシュの中に正しいバージョンのブロックが存在しない場合に発生します。OPS では、複数のインスタンスが同じブロックを共有してキャッシュ一貫性矛盾のない読取りアクセスができるので、この種の競合は簡単に解決できます。ただし、その他の種類の競合は、キャッシュ一貫性の観点から見てもっと複雑です。

たとえば、表への挿入処理の場合、writer / writer 競合は空きリスト・グループによって部分的に処理されます。ただし、その他の場合、ロックを使用してホット・ブロックを切り離すか、遅延 ping をインプリメントするか、またはアプリケーションのパーティション化によって writer / writer のキャッシュ一貫性の問題を処理することが唯一の代替方法です。reader / writer 競合は、それよりも一般的であり簡単に解決できます。

reader / writer 競合は、OLTP とハイブリッド・アプリケーションの最も一般的な種類の競合です。一般的なアプリケーションで DSS と OLTP 処理を組み合せる能力は、このような競合を解決するときの OPS の効率によって決まります。

reader / writer 競合の "reader" 側については、2 つのサブカテゴリがあります。現行の読取り側によって発生する競合と、一貫性読込みの読取り側によって発生する競合です。この 2 種類の競合のうち、通常は一貫性読込みの読取り側がより一般的であり、これはキャッシュ・フュージョンによって直接処理されます。

キャッシュ・フュージョンが一貫性読込みブロックを作成する方法

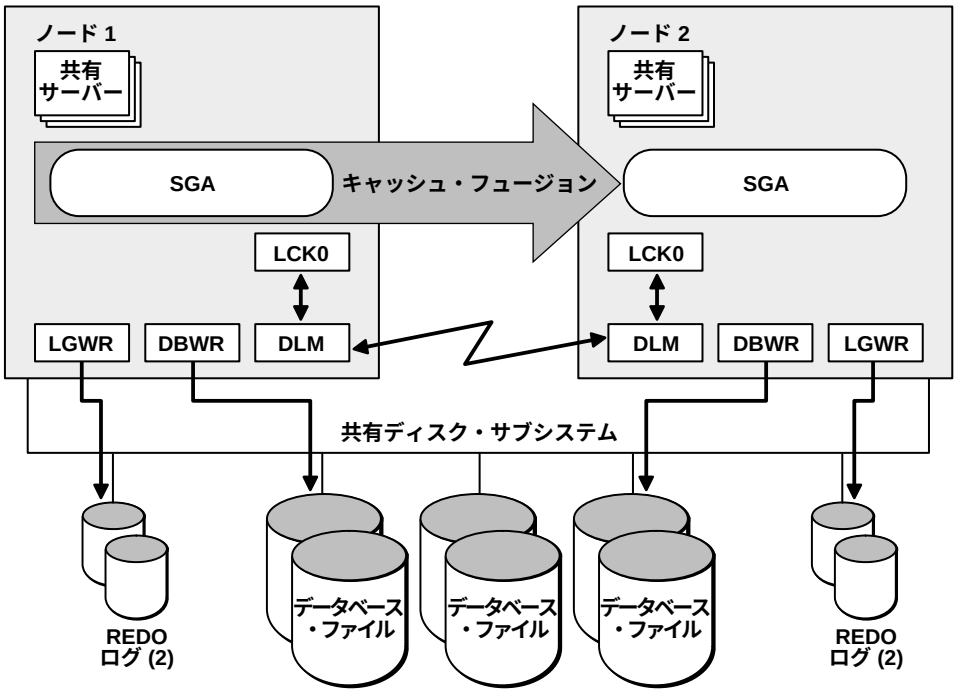
あるインスタンスが要求したデータ・ブロックがリモート・インスタンスのメモリー・キャッシュの中にある場合、キャッシュ・フュージョンはディスク・アクセスではなくリモート・メモリー・アクセスによって競合を解決します。要求元のインスタンスは、ブロックの一貫性読込みコピーに対する要求を保持側のインスタンスに送信します。保持側のインスタンスの BSP（ブロック・サーバー・プロセス）は、要求されたブロックの一貫性読込み

イメージを、高速インターコネクトを介して保持側のインスタンスのバッファ・キャッシュから要求元のインスタンスのバッファ・キャッシュに直接送信します。

図 20-1 に示すように、キャッシュ・フュージョンでは、待ち時間の短い高帯域幅のインターコネクトによって、1 つのノードのバッファ・キャッシュから別のノードのバッファ・キャッシュにデータ・ブロックを直接送ることができます。そのため、パラレル・キャッシュ管理において高コストのディスク I/O の必要性が削減されます。

またキャッシュ・フュージョンでは、新しいインターコネクト技術を活用して、待ち時間の短いユーザー領域ベースのプロセッサ間通信を実現しています。これにより、ノード間メッセージのためのオペレーティング・システムのコンテキスト切替えが減り、CPU 使用率が大幅に低下します。

図 20-1 キャッシュ・フュージョンによるインターコネクトを介したキャッシュ間ブロック移送



Oracle では、write/write 競合の管理に従来のディスク・ベースの PCM（パラレル・キャッシュ管理）を使用しています。将来のバージョンの Oracle では、キャッシュ・フュージョンによって writer/writer 競合をより高速に解決します。

注意： キャッシュ・フュージョンは常に使用可能になっています。

関連項目： パーティション化の詳細は、『Oracle8*i* 概要』を参照してください。

write/write 競合の解消を改善するためのデータの分割

キャッシュ・フュージョンは、多くの reader / writer 競合が発生するアプリケーションに対して改善された拡張性を提供しますが、それはブロック競合解消の問題の一部を解決するにすぎません。多くの writer / writer 競合が発生するアプリケーションでは、アプリケーションの表を正確にパーティション化して writer / writer 競合が発生する可能性を減らす必要があります。

関連項目： パーティション化の詳細は、『Oracle8*i* チューニング』および『Oracle8*i* 概要』を参照してください。

キャッシュ・フュージョンによる拡張性の向上

キャッシュ・フュージョンには次の効果があり、それによってアプリケーションのトランザクション・スループットと拡張性が向上します。

- オペレーティング・システムのコンテキスト切替えを大幅に削減。その結果、reader / writer キャッシュ一貫性矛盾の発生時における CPU 使用率を削減。
- ユーザー・モード IPC プラットフォームでは、CPU 使用率をさらに削減。
- ブロックの ping のための I / O の削減と、X-to-S ロック変換の削減。
- 高速インターコネクトを介した一貫性読込みブロック転送。

キャッシュ・フュージョンの効果が最も大きいアプリケーションは、ディスク・ベースの PCM で reader / writer 競合の発生率が高いアプリケーションです。

注意： 使用しているアプリケーションのインスタンス間アクティビティの 20% 以上が reader / writer 競合から発生している場合は、キャッシュ・フュージョンによってパフォーマンスの大幅な向上が期待できます。

パッケージ・アプリケーションでも、キャッシュ・フュージョンの結果、より効果的な拡張が可能です。キャッシュ・フュージョンは、OLTP とレポート作成機能が別々の複数のノードで実行されるアプリケーションにも効果があります。OLTP 機能が変更した表のデータにアクセスするレポート作成機能は、高速インターコネクトを介してレポート作成用のバージョンのデータ・ブロックを受信します。そのため、ディスクに対するデータ・ブロックの "ping" が減少します。

パフォーマンスの向上は、主に X-to-S ロック変換の減少と、それに伴う X-to-S ロック変換のためのディスク I/O の減少によるものです。

注意： キャッシュ・フュージョンにより、すべてのアプリケーションでパフォーマンスが向上します。向上の度合いは、オペレーティング・システム、アプリケーションの作業負荷、および全般的なシステム構成によって異なります。

プロセッサ間通信時に複数のインスタンス間でデータ・バッファを直接マッピングすることによって、1 つのアドレス領域から別のアドレス領域へメモリーをコピーする必要を回避できます。この短縮された実行パスにより CPU 所要量が減少し、この後で説明するように通信の帯域幅が増加します。

コンテキスト切替えと CPU 使用率の減少

キャッシュ・フュージョンにより、オペレーティング・システムのコンテキスト・スイッチは大幅に減少します。その結果、CPU 使用率が減少し、アプリケーション処理のための CPU サイクルが解放されます。

ユーザー・モード IPC による CPU 使用率の減少

キャッシュ・フュージョンは、ユーザー・モード IPC を利用して CPU 使用率を削減します。このユーザー・モード IPC は、Unix ベースおよび NT ベースの両プラットフォームで " メモリー・マップ IPC" とも呼ばれます。適切なハードウェアがサポートされていれば、オペレーティング・システムのコンテキスト・スイッチは、キャッシュ・フュージョンのみで実現できる基本的な削減量よりもさらに減少し、最小限になります。そのため、コストのかかるデータのコピーとシステム・コールも不要になります。

ユーザー・モード IPC では、オペレーティング・システムのカーネルを使用しなくても複数のユーザー・プロセスが通信できるので CPU 使用率が減少します。つまり、ユーザー実行モードからカーネル実行モードに切り替える必要がありません。

ブロックの ping のための I/O の削減と X-to-S ロック変換の削減

キャッシュ・フュージョンは、あるインスタンスのバッファ・キャッシュから別のインスタンスのバッファ・キャッシュに一貫性読込みブロックを直接送信することで、データ・ブロックおよび UNDO セグメント・ブロックに対するコストのかかるロック・オペレーションとディスク I/O を削減します。これにより、reader/writer 競合を解消するために必要な待ち時間を 90% 削減できます。

ディスク・ベースの PCM では、reader/writer 競合の解消に 80ms（ミリ秒）かかることがあります。これには、ディスクにロールバック・セグメント・ブロックを書き込む I/O だけでなく、要求されたブロックのディスク I/O も含まれます。

キャッシュ・フュージョンは、ディスク・ベースの PCM に必要な処理量の約 1/10 の処理量で、reader/writer 競合を解消します。このとき、ディスク I/O は発生しないか、発生してもごくわずかです。キャッシュ・フュージョンによって生じるオーバーヘッドは、一貫性読込み要求を処理し、要求されたブロックの一貫性読込みコピーをメモリーに作成し、作成したコピーを要求元のインスタンスに転送するためのオーバーヘッドのみです。プラットフォームによっては、これは 1ms もかかりません。

高速インターコネクトを介した一貫性読込みブロック転送

キャッシュ・フュージョンは高速の IPC を利用します。そのため、クラスタ・インターコネクトを介した待ち時間の短い通信を実現する最新技術によって、OPS のパフォーマンスが向上します。VIA やユーザー・モード IPC などのもっと効率のよいプロトコルにより、さらにパフォーマンス向上が期待できます。

OPS 用のインターコネクトおよびインターコネクト・プロトコル

キャッシュ・フュージョンのパフォーマンスに影響を与える主な要素は、インターコネクトと、ノード間通信を処理するプロトコルです。インターコネクトの帯域幅、その待ち時間、および IPC プロトコルの効率によって、キャッシュ・フュージョンが一貫性読込みブロックの要求を処理するスピードが決まります。

インターコネクト処理への影響

インターコネクトが動作可能になった後は、インターコネクトのパフォーマンスに大きな影響を与えることはできません。ただし、IPC のバッファ・サイズを調整することで、プロトコルの効率を変更できます。

関連項目： 詳細は、ベンダー固有のインターコネクトのマニュアルを参照してください。

サポートしているインターコネクト・ソフトウェア

OPS とキャッシュ・フュージョンをサポートするインターコネクトは、次のプロトコルのいずれかを使用します。

- TCP/IP（伝送制御プロトコル / インターネット・プロトコル）
- UDP（ユーザー・データグラム・プロトコル）
- VIA（仮想インタフェース・アーキテクチャ）
- ハードウェア・ベンダー固有のその他の専用プロトコル

OPS では、これらのプロトコルをサポートしている任意のインターコネクト製品を使用できます。使用するインターコネクト製品は、OPS のハードウェア・クラスタ・プラットフォーム用に保証されている必要もあります。

パフォーマンス期待値

キャッシュ・フュージョンのパフォーマンス・レベルを示す待ち時間とスループットは、アプリケーションごとに異なります。さらに、システムが処理するトランザクションの種類と組合せもパフォーマンスに影響します。

キャッシュ・フュージョンによるパフォーマンス向上の度合いは、作業負荷によって異なります。ハードウェア、インターコネクト・プロトコルの仕様、およびオペレーティング・システムのリソース使用率もパフォーマンスに影響します。

前述のように、キャッシュ・フュージョン導入前のアプリケーションに大きな一貫性読込み競合がなかった場合は、キャッシュ・フュージョンによるパフォーマンスの変化はおそらくありません。しかし、アプリケーションで一貫性読込み競合の結果として多数のロック変換と大量のディスク I/O が発生していた場合は、キャッシュ・フュージョンによってパフォーマンスが大幅に向上します。

例として、V$SYSSTAT の統計情報を問い合せると、キャッシュ・フュージョンを使用するシステムの処理では、X-to-S ロック変換が少なくなることが認められるはずです。アプリケーションが生成する X-to-S ロック変換が少ないほど、システムが要求するディスク I/O も少なくなります。キャッシュ・フュージョンのパフォーマンスの評価方法は、20-8 ページの「キャッシュ・フュージョンとインスタンス間パフォーマンスの監視」の項で詳しく説明します。

> **関連項目：** ロックの種類の詳細は、第 7 章「ロック・メカニズムの概要」を参照してください。

キャッシュ・フュージョンのブロック要求の待ち時間

キャッシュ・フュージョンでは、使用するプロトコルに応じてブロック要求の待ち時間が異なります。TCP/UDP では、待ち時間の範囲は 5 ～ 50ms です。ただし、TCP/UDP プロトコル使用時のパフォーマンス統計は、システムによって異なります。

NT でのみ使用可能な VIA では、1ms 以下の待ち時間も可能です。VIA プロトコルはコンテキスト切替えが少ないため、VIA プロトコル使用時のパフォーマンス統計に大きな違いはありません。コンテキスト切替えが少ないということは、オーバーヘッドの削減を意味します。

キャッシュ・フュージョンとインスタンス間パフォーマンスの監視

この項では、OPS とキャッシュ・フュージョンの統計情報を取得および分析してインスタンス間パフォーマンスを監視する方法について説明します。この項で説明する項目は、次のとおりです。

- キャッシュ・フュージョンと OPS のパフォーマンス監視の目標
- OPS とキャッシュ・フュージョンを監視するための統計情報
- V$SYSTEM_EVENTS を使用してパフォーマンスの問題を識別する方法

キャッシュ・フュージョンと OPS のパフォーマンス監視の目標

キャッシュ・フュージョンと OPS のパフォーマンス監視の主な目標は、待ち時間を調べ、時間の経過に伴う傾向を観察して、システムの処理を微調整することです。そのためには、以降の項で説明するように、パフォーマンスの統計情報をさまざまな観点から分析します。以降に示す監視手順を継続して使用し、処理の傾向を観察して処理を最適なレベルに保ちます。

OPS での待ち時間の統計

以降の項に示す手順では、待ち時間の観点からパフォーマンスの問題を説明します。また、統合分散ロック・マネージャ（IDLM）のリソース使用率や一般的なシステム・イベントのステータスなど、パフォーマンス関連の他の問題の分析方法についても説明します。

OPS 処理での待ち時間の役割

待ち時間は、OPS とキャッシュ・フュージョンのパフォーマンスで最も重要な側面です。待ち時間とは、一貫性読込みブロックに対する要求を完了するのにかかる時間です。待ち時間は、一貫性読込み操作に伴う要求と応答の種類に影響されます。次に説明するように、要求は種類ごとに結果が異なります。

- あるインスタンスが一貫性読込みバージョンのデータ・ブロックを必要とするとき、そのインスタンスは IDLM に対してロック要求を送信し、現在データ・ブロックを変更中であるインスタンスを判別します。この要求は、一般的に共有ロックの要求です。データ・ブロックが現在別のインスタンスによって変更中である場合、IDLM はそのブロックの X ロックの所有者に一貫性読込み要求を転送します。ブロックがインスタンスによって変更中でない場合、IDLM が要求元のインスタンスに S ロックを付与し、要求元のインスタンスはそのブロックをディスクから読み取ります。
- 要求されたブロックのロックを保持しているインスタンスの BSP は、メッセージ機能を使用して一貫性読込みバージョンのブロックを準備し、要求元のノードに転送します。BSP は最初に、要求されたブロックのバージョンをキャッシュ内で探します。そこに正しいバージョンがない場合は、保持側のインスタンスが、ローカル・キャッシュのロールバック・セグメント情報を使用して必要なバージョンを作成します。このため、ディスク I/O は発生しません。
- 要求されたブロックが保持側のインスタンスのキャッシュにない場合、保持側のインスタンスは要求元のインスタンスに対して、ディスクからブロックを読み取る許可を要求元に付与するメッセージを表示して応答することがあります。つまり、ほとんどの一貫性読込み要求はディスク I/O なしで処理されますが、要求元がディスクからブロックを読み取ることが必要なトランザクションもあります。

OPS とキャッシュ・フュージョンを監視するための統計情報

Oracle は、バッファ・キャッシュ・レイヤーと IDLM レイヤーからキャッシュ・フュージョン関連のパフォーマンス統計情報を収集します。また、ロック要求とロック待機に関する OPS 全般の統計情報も収集します。複数のビューを使用して、統計情報を調べることができます。

システム・パフォーマンスの適切な履歴を維持すると、統計情報の変化が表す傾向を識別しやすくなります。待ち時間の増大と作業負荷の不都合な変化を識別するには、これが特に重要です。

この項の手順では、次のトピックに従ってグループ化した統計情報を使用します。

- グローバル・キャッシュとキャッシュ・フュージョンの統計情報の分析
- グローバル・ロックの統計情報の分析
- IDLM リソース、ロック、メッセージ、およびメモリー・リソースの統計情報の分析
- OPS の I/O 統計情報の分析
- ロック変換のタイプ別分析
- ラッチ、OPS、および IDLM 関連の統計情報の分析

多くの場合、パフォーマンスの問題を解決するには、最初にそれぞれの統計情報グループに固有の手順を使用して問題を識別する必要があります。次に、V$SYSTEM_EVENT ビューを

使用して原因を特定します。これについては、20-26 ページの「V$SYSTEM_EVENTS の明確な OPS 関連イベント」の項で説明しています。

この項の手順で説明しているほとんどのビューの統計情報を収集するには、TIMED_STATISTICS パラメータに TRUE を設定する必要があります。この章で説明するビューの定期的な統計情報は、1 秒の 1/100 の単位で表しています。

注意： CATPARR.SQL を実行して、統計情報を格納および表示するための OPS 関連のビューと表を作成する必要があります。これについては次の項で説明しています。

CATPARR.SQL による OPS のデータ・ディクショナリ・ビューの作成

SQL スクリプト CATPARR.SQL では、パラレル・サーバーのデータ・ディクショナリ・ビューを作成します。このスクリプトを実行するには、SYSDBA 権限が必要です。

『Oracle8*i* リファレンス・マニュアル』で説明しているとおり、CATALOG.SQL は標準の V$ 動的ビューを作成します。また、次のビューも含まれます。

- GV$CACHE
- GV$PING
- GV$CLASS_PING
- GV$FILE_PING

エクステントを追加した後、CATPARR.SQL を再実行することで EXT_TO_OBJ 表に最新情報を格納できます。オブジェクトを削除し、CATPARR.SQL を再実行しなかった場合は、EXT_TO_OBJ の情報は正確でないことがあります。

関連項目：『Oracle8*i* リファレンス・マニュアル』（動的ビューおよびデータベースの監視について）

グローバル動的パフォーマンス・ビュー

Oracle データベースの調整情報およびパフォーマンス情報は、"V$ 固定ビュー " という一連の動的パフォーマンス表に格納されています。各アクティブ・インスタンスには、それぞれ一連の固定ビューがあります。OPS では、グローバル動的パフォーマンス（GV$）ビューに問合せを実行して、条件に合ったすべてのインスタンスから V$ ビュー情報を検索できます。V$ROLLNAME、V$CACHE_LOCK、V$LOCK_ACTIVITY および V$LOCKS_WITH_COLLISIONS を除く既存の動的パフォーマンス・ビューのすべてにグローバル固定ビューがあります。

グローバル・ビューには、ローカル・ビューのすべての列に加えて、INST_ID 列（データ・タイプ INTEGER）が表示されます。INST_ID 列には、対応する V$ 情報が取得されたイン

スタンスの番号が表示されます。INST_ID 列をフィルタとして使用し、使用可能なインスタンスのサブセットから V$ 情報を検索できます。たとえば、次のような問合せが可能です。

```
SELECT * FROM GV$LOCK WHERE INST_ID = 2 or INST_ID = 5;
```

この問合せでは、インスタンス 2 と 5 の V$ ビューから情報を検索できます。

各グローバル・ビューには、GLOBAL ヒントが含まれています。GLOBAL ヒントは、各インスタンスのローカル・ビューの内容を検索するパラレル問合せを作成します。

インスタンスがすでに PARALLEL_MAX_SERVERS の制限値に達している状態で GV$ ビューの問合せを試みると、この目的のために 1 つの追加パラレル・サーバー・プロセスが生成されます。この追加プロセスは、GV$ 問合せ以外のパラレル操作には利用できません。

注意： インスタンスの PARALLEL_MAX_SERVERS を 0（ゼロ）に設定した場合は、GV$ 問合せのための追加パラレル・サーバー・プロセスは発生しません。

インスタンスがすでに PARALLEL_MAX_SERVERS の制限値に達している状態で GV$ ビューへの複数の問合せを発行すると、最初の問合せだけが成功し、他のすべての問合せは失敗します。通常のパラレル問合せでは、サーバー・プロセスを割り当てることができない場合は、エラーになるか、問合せが問合せコーディネータによって順次実行されます。

グローバル・ビューの場合は、問合せのパラレル実行を続行し、問合せ用のサーバーを割り当てることができるインスタンスのデータを戻すことができます。サーバー割当てが成功したインスタンスだけから結果を取り出せばよい場合には、この値を TRUE に設定してください。

関連項目： 18-21 ページの「インスタンス・グループの指定方法」および『Oracle8*i* リファレンス・マニュアル』（GV$ ビューの制限事項について、および関連するすべてのパラメータと V$ 動的パフォーマンス・ビューについて）

グローバル・キャッシュとキャッシュ・フュージョンの統計情報の分析

Oracle は、1 つのインスタンス内のバッファ・キャッシュ・レイヤーでグローバル・キャッシュ統計を収集します。この統計情報には、グローバル・リソースに対する要求の件数とタイミングが含まれます。

データ・ブロックに対するグローバル・ロックの要求は、要求元のインスタンスのバッファ・キャッシュで発生します。要求が IDLM に入力される前に、要求の状態を追跡するためのデータ構造が、Oracle によって SGA に割り当てられます。この構造を " ロック要素 " と呼びます。

グローバル・キャッシュ統計を監視するには、次の手順で説明するように、V$SYSSTAT ビューに問合せを実行し、その出力を分析します。

グローバル・キャッシュ統計の監視手順

グローバル・キャッシュ統計を分析するには、次の手順に従います。

1. 次の構文を使用して V$SYSSTAT に問合せを実行します。

```
SELECT * FROM V$SYSSTAT WHERE NAME LIKE 'global cache';
```

次のような Oracle の出力が示されます。

```
NAME                                                              VALUE
----------------------------------------------------------------- ----------
global cache gets                                                    12480
global cache get time                                                  996
global cache converts                                                   21
global cache convert time                                               48
global cache cr blocks received                                          1
global cache cr block receive time                                       1
global cache cr read from disk                                           0
global cache freelist waits                                              0
global cache defers                                                      0
global cache convert timeouts                                            0
global cache cr timeouts                                                 0
global cache fairness down converts                                      0
```

V$SYSSTAT の出力を使用して、この統計グループの残りの手順で説明する計算と分析を実行します。

手順 2 と 3 では、次に示す V$SYSSTAT のキャッシュ・フュージョン統計を使用します。

- global cache cr blocks received
- global cache cr blocks receive time
- global cache cr read from disk
- global cache cr timeouts

- global cache fairness down converts

2. 次の計算式を使用してキャッシュ・フュージョン要求の待ち時間を計算します。

$$\frac{\textit{global cache cr block receive time}}{\textit{global cache cr blocks received}}$$

この結果は一般にシステムの構成とボリュームによって 5 ～ 40ms になり、一貫性読込み要求が要求元のインスタンスから保持側のインスタンスに送られて要求元のインスタンスに戻ってくるまでの、往復の平均待ち時間です。CPU のアイドル時間に制約があり、システムの典型的な処理が実行時間の長い問合せである場合は、待ち時間が長くなります。ただし、インターコネクト・プロトコルがユーザー・モード IPC であれば、平均待ち時間を 1ms 未満にすることができます。

DB_MULTI_BLOCK_READ_COUNT パラメータの値が大きい場合、その値が一貫性読込みサーバー要求の待ち時間に影響することがあります。このパラメータの設定によって、要求元のプロセスがブロックに対して複数の要求を発行することがあるからです。それに応じて、要求元のプロセスの待ち時間は長くなります。

3. 次の計算式を使用して、一貫性読込み要求の合計数を計算します。

global cache cr block received + global cache cr blocks read from disk

ディスクからの読取り許可の割合が高いということは、保持側のインスタンスのバッファ・キャッシュにブロックが蓄えられている期間が非常に短いことを示します。これを解決するには、バッファ・キャッシュのサイズを増やします。

手順 4 と 5 では、たとえば UTLBSTAT や UTLESTAT を使用して統計情報のスナップショットをとる必要があります。

> **関連項目：** UTLBSTAT および UTLESTAT の詳細は、『Oracle8*i* チューニング』を参照してください。

手順 4 では、V$SYSSTAT の次のグローバル・キャッシュ統計を使用します。

- global cache convert time
- global cache converts
- global cache get time
- global cache gets

4. 次の計算式のどちらかを使用して、平均変換時間と平均取得時間を計算します。

$$\frac{\textit{global cache convert time}}{\textit{global cache converts}} \text{ または } \frac{\textit{global cache get time}}{\textit{global cache gets}}$$

平均変換時間または平均取得時間が大きい場合は、過剰な競合が起きています。別の原因としては、システムの全般的な作業負荷または問題のために、ロック・オペレーションの待ち時間が長くなっていることが考えられます。キャッシュ取得の妥当な値は 20 ～ 30ms で、変換の場合は平均 10 ～ 20ms です。

Oracle では、リソースに対する新しいロックをオープンした時点で global cache gets 数が増分されます。変換は、すでにオープンされているロックがあり、それを Oracle が別のモードに変換したときにカウントされます。

したがって、取得の経過時間には、新しいロックの割当てと初期化が含まれます。キャッシュの平均取得時間または平均変換時間の値が大きすぎる場合は、システムでタイムアウトが発生していることがあります。

グローバル・キャッシュ変換時間またはグローバル・キャッシュ取得時間が長い場合は、V$SYSTEM_EVENTS ビューの統計情報を参照して、TIME_WAITED 統計の値が上限の状態でのイベントを識別してください。

5. 'global cache convert timeouts' の値を調べて、ロック変換のタイムアウトを分析します。V$SYSSTAT 出力でこの統計が 0（ゼロ）以外の値を示している場合は、輻輳状態または過剰競合が発生していないか、システムをチェックしてください。通常は、変換タイムアウトは発生しません。変換タイムアウトは、重大なパフォーマンスの問題が存在していることを示します。

6. V$SYSSTAT 出力でこの統計の値を調べて、global cache consistent-read timeouts を分析します。この統計の値は、一貫性読込み要求が完了する前にシステムがタイムアウトしたときに増分されます。この統計が 0（ゼロ）以外の値を示している場合は、一貫性読込み要求の開始後に長時間が経過しており、タイムアウトが発生しています。この状況が起きた場合、通常は一貫性読込み要求が完了するまでの平均時間も増えていることがわかります。タイムアウトが発生していて待ち時間が長い場合は、システムの作業負荷が大きすぎるか、データ・ブロックに対する過剰な競合が起きている可能性があります。

グローバル・ロックの統計情報の分析

グローバル・ロックの統計情報には、PCM および非 PCM ロック・アクティビティの両方の件数とタイミングが含まれます。Oracle では、IDLM API レイヤーからグローバル・ロックの統計情報を収集します。IDLM のすべての Oracle クライアントに対してバッファ・キャッシュは 1 つなので、IDLM のすべての Oracle クライアントはこのレイヤーを介して IDLM に要求を行います。したがって、グローバル・ロックの統計情報にはカーネルのすべてのレイヤーで発行されたロック要求が含まれ、グローバル・ロックの統計情報はバッファ・キャッシュの OPS アクティビティに関連しています。

この項に示す手順に従って V$SYSSTAT のデータを監視し、平均、待ち時間および件数を導出します。それによって、インスタンスで生成される OPS 作業負荷のおおまかな指標を作成できます。

グローバル・ロック統計情報の分析手順

グローバル・ロック処理に関する V$SYSSTAT の統計情報を表示および分析するには、次の手順に従います。

1. 次の構文を使用して V$SYSSTAT に問合せを実行します。

```
SELECT * FROM V$SYSSTAT WHERE NAME LIKE 'global lock';
```

次のような Oracle の出力が示されます。

```
NAME                                                              VALUE
----------------------------------------------------------------- ----------
global lock sync gets                                                    703
global lock async gets                                                 12748
global lock get time                                                    1071
global lock sync converts                                                303
global lock async converts                                                41
global lock convert time                                                  93
global lock releases                                                     573
```

V$SYSSTAT の出力を使用して、この統計グループの残りの手順で説明する計算と分析を実行します。

2. 次の計算式を使用して、平均グローバル・ロック取得時間を計算します。

$$\frac{\textit{global lock get time}}{\textit{(global lock sync gets + global lock async gets)}}$$

結果が 20ms または 30ms を上回った場合は、DESCEND キーワードを使用して V$SYSTEM_EVENTS の TIME_WAITED 列を問い合せ、次の問合せを使用して最も頻繁に待ち状態になってるロック・イベントを識別します。

```
SELECT EVENT_TIME_WAITED, AVERAGE_WAIT
FROM V$SYSTEM_EVENTS
ORDER BY TIME_WAITED DESCEND;
```

Oracle では、リソースに対して新しいロックがオープンされた時点で global lock gets 数が増分されます。変換は、すでにオープンされているロックがあり、それを Oracle が別のモードに変換したときにカウントされます。

したがって、取得の経過時間には、新しいロックの割当てと初期化が含まれます。ロックの平均取得時間または平均変換時間が大きすぎる場合は、システムでタイムアウトが発生していることがあります。

グローバル・ロック変換時間またはグローバル・ロック取得時間が長い場合は、V$SYSTEM_EVENTS ビューの統計情報を参照して、TIME_WAITED 統計の値が大きいイベントを識別します。

3. 次の計算式を使用して、平均グローバル・ロック変換時間を計算します。

$$\frac{\textit{global lock convert time}}{\textit{(global lock sync converts + global lock async converts)}}$$

結果が 20ms を上回っている場合は、DESCEND キーワードを使用して V$SYSTEM_EVENTS の TIME_WAITED 列を問い合せ、遅延の原因になっているイベントを識別します。

4. V$LIBRARYCACHE ビューと V$ROWCACHE ビューを分析して、非 PCM リソースに対する IDLM アクティビティを考察します。これらのビューには、IDLM のリソース使用率を識別する IDLM 固有の列があります。これを行うのは、ライブラリ・キャッシュ確保、エンキューまたは DFS ロック・ハンドルに対する長時間の待機が頻繁に発生する場合です。

IDLM リソース、ロック、メッセージおよびメモリー・リソースの統計情報の分析

Oracle では、IDLM リソース、ロックおよびメッセージの統計情報を IDLM レベルで収集します。この統計情報を使用して、IDLM の待ち時間と作業負荷を監視します。この統計情報が表示されるビューは、V$DLM_CONVERT_LOCAL ビューと V$DLM_CONVERT_REMOTE ビューです。

これらのビューは、ロック要求の種類ごとに、平均変換時間、件数情報および定期的な統計情報を記録します。V$DLM_CONVERT_LOCAL はローカル・ロック・オペレーションの統計情報を示します。V$DLM_CONVERT_REMOTE は、リモート変換の値を示します。これらのビューの平均変換時間は 1 秒の 1/100 の単位で表されます。

注意： これらのビューの件数情報は、インスタンスの存続期間中の累積です。

IDLM の作業負荷によるパフォーマンスへの影響

IDLM の作業負荷は、OPS およびキャッシュ・フュージョンのパフォーマンスの重要な側面です。一貫性読込み要求のたびに、結果としてロック要求が発生するからです。要求発生率が高いために IDLM の作業負荷が大きくなると、パフォーマンスに悪影響を与えます。

IDLM は、ローカル・ロック・オペレーションを完全にローカル・ノード内で実行します。つまり、メッセージを送信せずに実行します。リモート・ロック・オペレーションでは、他のノードにメッセージを送信し、その応答を待つ必要があります。しかし、IDLM ではほとんどの下方変換がローカル操作です。

次に示す IDLM リソース、ロックおよびメッセージの統計情報の分析手順は、2 つのグループに分かれています。最初のグループの手順では、IDLM リソースとロックを監視する方法について説明します。2 番目のグループでは、メッセージの統計情報を監視する方法について説明します。

IDLM リソースおよびロックの統計情報の分析手順

DLM リソース処理に必要な V$DLM_CONVERT_LOCAL と V$DLM_CONVERT_REMOTE の統計情報を取得して分析するには、次の手順に従います。

V$DLM_CONVERT_LOCAL ビューと V$DLM_CONVERT_REMOTE ビューへの移入を行うには、イベント 29700 を使用可能にする必要があります。そのためには、次の構文を入力します。

```
EVENT="29700 TRACE NAME CONTEXT FOREVER"
```

1. 次の構文を使用して V$DLM_CONVERT_LOCAL に問合せを実行します。

```
SELECT CONVERT_TYPE,
AVERAGE_CONVERT_TIME,
CONVERT_COUNT
FROM V$DLM_CONVERT_LOCAL;
```

次のような Oracle の出力が示されます。

```
CONVERT_TYPE                          AVERAGE_CONVERT_TIME CONVERT_COUNT
------------------------------------- -------------------- -------------
NULL -> SS                                              0             0
NULL -> SX                                              0             0
NULL -> S                                               1           146
NULL -> SSX                                             0             0
NULL -> X                                               1            92
SS   -> SX                                              0             0
SS   -> S                                               0             0
SS   -> SSX                                             0             0
SS   -> X                                               0             0
SX   -> S                                               0             0
SX   -> SSX                                             0             0
SX   -> X                                               0             0
S    -> SX                                              0             0
S    -> SSX                                             0             0
S    -> X                                               3            46
SSX  -> X                                               0             0
16 rows selected.
```

2. 次の構文を使用して V$DLM_CONVERT_REMOTE に問合せを実行します。

```
SELECT * FROM V$DLM_CONVERT_REMOTE;
```

V$DLM_CONVERT_LOCAL の出力と同じ形式の Oracle の出力が示されます。

V$DLM_CONVERT_LOCAL と V$DLM_CONVERT_REMOTE の出力を使用して、次の手順で説明する計算を実行します。

3. 次の問合せを使用して、ローカル・ロック・オペレーションとリモート・ロック・オペレーションの比率を計算します。

```
SELECT r.CONVERT_TYPE,
 r.AVERAGE_CONVERT_TIME,
 l.AVERAGE_CONVERT_TIME,
 r.CONVERT_COUNT,
 l.CONVERT_COUNT,
   FROM V$DLM_CONVERT_LOCAL l, V$DLM_CONVERT_REMOTE r
   GROUP BY r.CONVERT_TYPE;
```

4. 作業負荷とロック変換の待ち時間の履歴を維持することは有用です。ロック・オペレーションの要求率が変化していて平均待ち時間は増えていない場合、通常その変化は、アプリケーションにおける作業負荷のパターンの変化によるものです。

   要求率と待ち時間の両方が悪化している場合、それは通常、メッセージの待ち時間、システムの問題、またはタイムアウトによってロック競合の割合または作業負荷が増加していることを示します。LMD プロセスの CPU 消費率が高い、すなわち CPU 消費率が 20% を超えていて、全般的なシステム・リソースの消費率が正常の範囲内である場合、複数の CPU を持つシステムではその LMD プロセスを特定のプロセッサにバインドすることを検討してください。

   待ち時間が増えた場合は、CPU のデータと、"sar"、"vmstat"、"netstat" などのユーティリティを使用して取得できるオペレーティング・システムの統計情報も調べてください。

IDLM メッセージの統計情報

IDLM は、メッセージを直接送信するか、フロー制御を使用して送信します。どちらの方法でも、IDLM は各メッセージに " チケット " というマーカーを付加します。各 IDLM のチケット割当てには限度があります。ただし、IDLM はチケットを無期限に再使用できます。

IDLM は、使用可能なチケットがなくなるまでメッセージを直接送信します。IDLM がチケットを使い果たした場合、未処理メッセージの処理が終了して使用可能なチケットの数が増えるまで、メッセージはフロー制御キューの中で待機する必要があります。フロー制御メッセージ機能は、LMD プロセスが管理します。

チケットの数を制限することによって、負荷の大きいインスタンス間通信中に 1 つのノードから別のノードへ過剰なメッセージ送信が行われることを防止できます。また、リモート一貫性読込みブロック要求の負荷が大きいノードが、負荷の小さい他のノードがあるにもかかわらず、クラスタ全体のメッセージ・リソースの制御権を得ることも防止できます。

V$DLM_MISC ビューには、メッセージ・アクティビティに関する次の統計情報が含まれます。

- DLM messages sent directly（直接送信した DLM メッセージの数）
- DLM messages flow controlled（フロー制御した DLM メッセージの数）
- DLM messages received（受信した DLM メッセージの数）
- DLM total incoming message queue length（DLM 受信メッセージ・キュー長の合計）

IDLM メッセージ統計情報の分析手順

V$DLM_MISC のメッセージ統計情報を取得して分析するには、次の手順に従います。

1. 次の構文を使用して V$DLM_MISC に問合せを実行します。

```
SELECT NAME, VALUE FROM V$DLM_MISC;
```

次のような Oracle の出力が示されます。

```
STATISTIC# NAME                                VALUE
---------- ---------------------------------- ------
         0 dlm messages sent directly          29520

         1 dlm messages flow controlled         1851

         2 dlm messages received               29668

         3 dlm total incoming msg queue length   297

4 rows selected.
```

注意： V$DLM_MISC 出力の情報は、オラクル・カスタマ・サポートでデバッグを行うために必要な場合があります。

V$DLM_MISC の出力を使用して、次の手順を実行します。

2. 次の計算式を使用して、2 つのスナップショットの間で受信キューの長さの平均を計算します。

$$\frac{\textit{total incoming message queue length}}{\textit{messages received}}$$

Oracle では、LMD プロセスのメッセージ・キューに新しい要求が入れられたときに、'total incoming message queue length'（受信メッセージ・キュー長の合計）の値が増分されます。メッセージの処理を開始するためにメッセージが LMD キューから取り出されたときに、Oracle によって 'messages received'（受信したメッセージの数）の値が増分されます。

多数の要求が同時に LMD に届き、キューのサイズが増大することがあります。これが起こるのは、ロック・アクティビティの量が多いとき、または LMD が大量の一貫性読込み要求を処理しているときです。一般的に受信キューの長さの平均は 10 未満です。

OPS の I/O 統計情報の分析

前述のグローバル・キャッシュおよびグローバル・ロックの統計情報に加えて、V$SYSSTAT の統計情報を使用して、グローバル競合の解消に関連した I/O の作業負荷も計測できます。V$SYSSTAT にはこの目的のための 3 種類の重要な統計情報があります。

- DBWR forced writes（DBWR 強制書込み回数）
- Remote instance undo header writes（リモート・インスタンスの UNDO ヘッダー書込み回数）
- Remote instance undo block writes（リモート・インスタンスの UNDO ブロック書込み回数）

DBWR 強制書込みは、データ・ブロックを要求した要求元のノードが要求したブロックを使用する前に Oracle がそのブロックをディスクに書き込み、それによってインスタンス間データ・ブロック競合を解消する場合に発生します。

一貫性読込み要求で、ブロックをロールバックして一貫した読取りを行うために別のインスタンスのキャッシュの情報が必要な場合、Oracle はロールバック・セグメント・ヘッダーとロールバック・セグメント・ブロックもディスクに書き込む必要があります。1 つのインスタンスが UNDO ブロックと UNDO ヘッダーをディスクに書き込む必要があり、別のインスタンスがそれを読み取ります。

キャッシュ・フュージョンによって一貫性読込みのための I/O は最小限で済みます。そのため、各インスタンスが実行する物理書込みが大幅に減少します。キャッシュ・フュージョンを取り入れる前、リモート・インスタンスのデータが関係する一貫性読込みでは、要求された 1 つのブロックに対して最大 3 回の書込みと 3 回の読取り、ロールバック・セグメント・ヘッダー、UNDO セグメント・ブロックおよび複数のロック変換が必要でした。

OPS の I/O 統計情報の分析手順

V$SYSSTAT のメッセージ統計情報を取得して分析するには、次の手順に従います。

1. 次の構文を使用して V$SYSSTAT に問合せを実行します。

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME IN ('DBWR forced writes',
'remote instance undo block writes',
'remote instance undo header writes',
'physical writes');.
```

次のような Oracle の出力が示されます。

```
NAME                                                          VALUE
----------------------------------------------------------- ----------
physical writes                                               41802
DBWR forced writes                                             5403
remote instance undo block writes                                 0
remote instance undo header writes                                2
4 rows selected.
```

V$SYSSTAT の出力を使用して、次の計算を実行します。

2. 次の計算式を使用して、物理 I/O 全体に対する OPS 関連 I/O の割合を計算します。

$$\frac{\textit{DBWR forced writes}}{\textit{physical writes}}$$

この計算とキャッシュ・フュージョン導入前の統計情報では、割合が大幅に減少しています。

3. リモート・インスタンスが、ローカル・インスタンスによって使用されているロールバック・セグメントから読み取る必要がある場合に、ロールバック・セグメントへの書込みがどれだけ発生しているかを、次の計算式を使用して計算します。

$$\frac{\textit{(remote instance undo header writes+ remote instance undo block writes)}}{\textit{DBWR forced writes}}$$

この割合は、ロールバック・セグメントへの書込みに関連するディスク I/O の量を示しています。この割合は、キャッシュ・フュージョンによってかなり小さくなるはずです。

ロック変換のタイプ別分析

この項では、3 つのビューの出力を分析し、ロック変換をタイプ別に定量化する方法を説明します。この項で説明する作業とビューは、次のとおりです。

- V$LOCK_ACTIVITY を使用してロック変換を分析する方法
- V$CLASS_PING を使用して ping をブロック・クラス別に識別する方法
- V$PING を使用してホット・オブジェクトを識別する方法

関連項目： ロック変換の詳細は、第 8 章「統合分散ロック・マネージャ」を参照してください。

V$LOCK_ACTIVITY を使用してロック変換を分析する方法

V$LOCK_ACTIVITY には、インスタンスの存続期間中に発生したロックの上方変換と下方変換の数がまとめて示されます。X-to-N 下方変換は、別のインスタンスがリソースを変更しようとしたためにロックが下方変換された回数を示します。

下方変換のもう 1 つのタイプは、X-to-S です。このタイプの下方変換は、あるインスタンスが読み取ったリソースが、ローカル・インスタンスによって最後に変更されたリソースである場合に発生します。どちらのタイプのロック変換も I/O を伴います。しかし、キャッシュ・フュージョンの場合、X-to-S 下方変換はバッファ・ロックに必要ないため減少します。

V$CLASS_PING を使用して ping をブロック・クラス別に識別する方法

V$CLASS_PING はロック変換のアクティビティをまとめ、次のブロック・クラスでディスク I/O が発生しているかどうかを示します。

- データ・ブロック
- セグメント・ヘッダー
- エクステント・ヘッダー
- UNDO ブロック

すべての X_2_NULL_FORCED_WRITE 変換と X_2_S_FORCED_WRITE 変換は、書込み I/O を伴います。つまり、各ブロック・クラスの列の値には、ディスク I/O の原因を示す標識があります。

V$PING を使用してホット・オブジェクトを識別する方法

V$PING は、" ホット・ブロック " と " ホット・オブジェクト " の識別に役立ちます。FORCED_READS 列と FORCED_WRITES 列の合計は、特定のブロックまたはオブジェクトに関する実際の ping アクティビティを示します。

3 つのビューすべてが、異なるレベルの詳細情報を提供します。パフォーマンスの問題の原因が ping または OPS 自体であると想定される場合は、V$LOCK_ACTIVITY を監視して OPS 全体の作業負荷プロファイルを作成します。V$LOCK_ACTIVITY の情報を使用して、ロック変換の発生率を記録します。

詳細を得るには、V$CLASS_PING を使用してロック変換と ping が発生しているブロックのタイプを識別します。クラスを識別したら、V$PING を使用して、特定の表または索引と、多くのロック変換アクティビティが発生しているファイル番号およびブロック番号の詳細を取得します。

ラッチ、OPS および IDLM 関連の統計情報の分析

ラッチは、SGA データ構造を保護する下位レベルのロック・メカニズムです。ラッチに対する過剰な競合は、パフォーマンスを低下させます。

IDLM 内のラッチ競合を監視するには、V$DLM_LATCH と V$LATCH_MISSES を使用します。これらのビューは、特定のラッチ、そのラッチの統計、およびそのラッチが取得されたコード内の位置に関する情報を示します。

ラッチ、OPS および IDLM 関連の統計情報の分析手順

ラッチ、OPS および IDLM 関連の統計情報を分析するには、次の手順に従います。

1. 次の構文を使用して V$LATCH に問合せを実行します。

```
SELECT * FROM V$LATCH;
```

次のような Oracle の出力が示されます。

```
cache buffer handle              46184          1        1           0           0
cache buffers chained         84139946     296547     .996       29899        .101
cache buffers lru             4760378      11718     .998         227        .019
channel handle pool                 1          0        1           0           0
channel operations                  1          0        1           0           0
dlm ast latch                  542776        494     .999        1658       3.356
dlm cr bast queue               37194          1        1           0           0
dlm deadlock list               32839          0        1           0           0
dlm domain lock la                  1          0        1           0           0
dlm domain lock ta              49164          1        1           0           0
dlm group lock lat                  1          0        1           0           0
dlm group lock tab              25239          1        1           0           0
dlm lock table fre             325306        270     .999         327       1.211
dlm process hash l               6346          0        1           0           0
dlm process table                   2          0        1           0           0
dlm recovery domai               2014          0        1           0           0
dlm resource hash              683031       1709     .997       41342      24.191
dlm resource scan                 188          0        1           0           0
dlm resource table             182093         70        1           2        .029
dlm shared communication       190766        211     .999         313       1.483
dlm timeout list               113294         40        0           3        .075
dml lock allocation               261          0        1           0           0
22 rows selected.
```

注意： この出力例の 5 列の内容は、左から右の順に、gets（取得）、hits（ヒット）、misses（ミス）、sleeps（スリープ）、sleeps-to-misses ratio（スリープ対ミスの比率）です。

---

2. 前の手順の出力で sleeps-to-misses（スリープ対ミス）の比率が高かった場合は、スリープが発生している場所の特定を試みてください。そのためには、V$LATCH_MISSES ビューに対して次の問合せを実行します。

```
SELECT PARENT_NAME, "WHERE", SLEEP_COUNT
FROM V$LATCH_MISSES
ORDER BY SLEEP_COUNT DESCENDING;
```

次のような Oracle の出力が示されます。

```
PARENT_NAME                WHERE                        SLEEP_COUNT
-------------------------  ----------------------      --------------
dlm resource hash list     kjrrmas1:lookup master n              39392
cache buffers chains       kcbgtcr:kslbegin                      27738
library cache              kglhdgn:child:                        15408
shared pool                kghfnd:min scan                        6876
cache buffers chains       kcbrls:kslbegin                        2124
shared pool                kghalo                                 1667
dlm ast latch              kjucll:delete lock from                1464
7 rows selected.
```

V$LATCH と V$LATCH_MISSES の出力を使用して、次の手順を実行します。

3. 次の計算式により、20-24 ページのこのグループの最初の手順に示した V$LATCH 出力を使用して misses（ミス）に対する gets（取得）の比率を計算します。

$$\frac{gets}{misses}$$

misses（ミス）の数値が大きい場合、通常は同一のリソースとロックに対する競合があることを示します。比率の許容範囲は、90 ～ 95% です。

4. 20-25 ページの手順 2 にある V$LATCH_MISSES 出力を使用して、misses（ミス）に対する sleeps（スリープ）の比率を分析します。この比率から、プロセスがすぐにラッチを取得することができず、ラッチを待機するときにスリープ状態になる頻度がわかります。

比率が 2 であれば、ミスのたびに、プロセスはラッチを取得するまで 2 回試行したことを意味します。sleeps / misses（スリープ / ミス）の数値が大きい場合、それは通常、プロセス・スケジュールの遅延またはオペレーティング・システムの作業負荷が大きいことを示します。また、単一のリソースについての内部的な非効率や同時実行性の高さも示します。たとえば、同じリソースに対して多数のロックが同時にオープンされた場合、複数のプロセスがリソースのラッチを待機する必要があります。

V$LATCH_MISSES では、ラッチを取得した関数が WHERE 列に示されます。この情報は、内部のパフォーマンス問題を判断するときに役に立ちます。通常、長時間スリープ状態であったラッチは、V$SESSION_WAIT または V$SYSTEM_EVENT の 'latch free'（ラッチなし）待機イベント・カテゴリに示されます。

次の項では、V$SYSTEM_EVENTS の使用方法についてさらに詳しく説明します。

V$SYSTEM_EVENTS を使用してパフォーマンスの問題を識別する方法

V$SYSTEM_EVENT ビューには、キャッシュ・フュージョンと OPS イベントに関するデータが表示されます。プロセスが最も長く待機したイベントを識別するには、DESCENDING キーワードを使用して V$SYSTEM_EVENT の TIME_WAITED 列を問い合せます。TIME_WAITED 列は、リストされたシステム・イベントごとに合計待機時間を示します。V$SYSTEM_EVENTS の問合せ方法は、20-16 ページのステップ 2 を参照してください。

イベント待機の順序リストを生成すると、パフォーマンスのボトルネックを簡単に特定できます。COUNT はそれぞれ、非強制コンテキスト切替えを表します。TIME_WAIT 値は、プロセスが特定のシステム・イベントを待機した累積時間です。TOTAL_TIMEOUT 列と AVERAGE_WAIT 列の値には、システム効率についての追加情報含まれます。

V$SYSTEM_EVENTS の明確な OPS 関連イベント

V$SYSTEM_EVENT 出力に表示される次のイベントは、OPS イベントに関する待機を表します。

- Global cache cr request
- Library cache pin
- Buffer busy due to global cache
- Global cache lock busy
- Global cache lock open x
- Global cache lock open s
- Global cache lock null to x

- Global cache lock s to x
- Global cache lock null to s

非 PCM リソースに関連するイベント

パフォーマンスの問題は OPS に関連することがあるので、前項で示したイベント以外のイベントも監視できます。それらのイベントは、次のとおりです。

- Row cache locks
- Enqueues
- Library cache pins
- DFS lock handle

まとめ

グローバル・キャッシュ・イベントのために発生した待機時間が他の待機時間と比較して長い場合は、V$SYSSTAT の統計情報とオペレーティング・システムのパフォーマンス・モニターを使用して、待ち時間の増加、競合、またはシステムの過剰な作業負荷がないか確認してください。グローバル・キャッシュがビジーまたはバッファがビジーであるために発生した待機時間の数値が大きい場合、それはバッファ・キャッシュ内で競合が増加していることを示します。

データ・ブロック・アドレスのロックが発生し、競合の度合いが高い OLTP システムでは、グローバル・キャッシュ待機イベントが合計待機時間の総計に対して高い割合を占めることは珍しくありません。

'row cache lock'、'enqueues' および 'library cache pin' の各行の統計情報が示すように、非バッファ・キャッシュ・リソース待機によって大量の待機時間が発生している場合は、V$ROWCACHE ビューと V$LIBRARYCACHE ビューで OPS 関連の問題を監視してください。特に、それぞれのビューの IDLM 列の値に注目してください。

一般的な OPS の問題は、管理が不完全な領域パラメータまたはキャッシュされていない順序によって発生します。そのような場合は、行キャッシュ・ロックとエンキューのための待機がプロセスに発生し、V$ROWCACHE は特定のディクショナリ・キャッシュに対する高い値の競合を示します。

21

データベースのバックアップ

Those behind cried "Forward!"
And those before cried "Back!"

Thomas Babington, Lord Macaulay: *On Frederic The Great*

データをプロテクトするために、オンライン REDO ログ・ファイルをアーカイブし、定期的にデータ・ファイルをバックアップしてください。データベースの制御ファイルおよび各インスタンスのパラメータ・ファイルもバックアップする必要があります。この章では、次の作業の実行方法について、項目別に説明します。

- バックアップ方法の選択
- REDO ログ・ファイルのアーカイブ
- チェックポイントおよびログ・スイッチ
- データベースのバックアップ

Oracle Parallel Server（OPS）は、排他モードでの Oracle のすべてのバックアップ機能をサポートしています。データベース全体または個々の表領域のどちらでも、オープンおよびクローズドの両方の方法でバックアップできます。

バックアップ方法の選択

バックアップおよびリカバリの操作は、次の 2 通りの方法で実行できます。

- Recovery Manager を使用する方法
- オペレーティング・システムを使用する方法

この章に記載する情報は、特に断らない限り両方の方法に適用します。

注意： データ・ファイルおよび表領域のオンラインとオフラインの混同を避けるために、この章では、バックアップ中にデータベースが使用可能であるか使用不能であるかを示すのに、" オープン " および " クローズド " という用語を使用しています。" 全体バックアップ " または " データベース・バックアップ " という用語は、すべてのデータ・ファイルおよび制御ファイルをバックアップすることを意味します。" 全体バックアップ " および " 増分バックアップ " は、Recovery Manager が提供する特定の種類のバックアップのみを指します。

関連項目： 『Oracle8*i* バックアップおよびリカバリ・ガイド』（バックアップおよびリカバリの操作および用語について）

REDO ログ・ファイルのアーカイブ

この項では、パラレル・サーバーの各インスタンス用の REDO ログ・ファイルをアーカイブする方法について説明します。

- アーカイブ・モード
- 自動アーカイブと手動アーカイブ
- アーカイブ・ファイルのフォーマットとアーカイブ先
- 制御ファイルの REDO ログ履歴
- アーカイブ・ログのバックアップ

アーカイブ・モード

Oracle には、2 つのアーカイブ・モードがあります。ARCHIVELOG モードと NOARCHIVELOG モードです。Oracle を ARCHIVELOG モードで運用する場合、REDO ログが上書きされないように、ログへの記入と同時にアーカイブする必要があります。したがって、メディア障害が発生しても Oracle はログ・ファイルを回復できます。ARCHIVELOG モードでは、オープン・バックアップおよびクローズド・バックアップの両方を作成できます。NOARCHIVELOG モードでは、クローズド・バックアップのみを作成できます。

注意： アーカイブとは、次のどちらかの方法で扱われるインスタンスごとの操作です。

- パラレル・サーバー上の各インスタンスはそれ自身の REDO ログ・ファイルをアーカイブできます。
- また、1 つ以上のインスタンスがすべてのインスタンスの REDO ログ・ファイルを手動でアーカイブできます。これについては、次の項で説明します。

関連項目： 21-11 ページの「オープン・データベース・バックアップとクローズド・データベース・バックアップ」

自動アーカイブと手動アーカイブ

アーカイブは、特定のインスタンスに対して自動または手動で実行できます。自動か手動かは、LOG_ARCHIVE_START 初期化パラメータの設定値によって決まります。

- LOG_ARCHIVE_START を TRUE に設定すると、Oracle は REDO ログを自動的にアーカイブしていきます。
- LOG_ARCHIVE_START を FALSE に設定すると、Oracle はユーザーからアーカイブの指示があるまで待機します。

LOG_ARCHIVE_START は、各 OPS インスタンスごとに異なる設定にできます。たとえば、インスタンス 2 で LOG_ARCHIVE_START が FALSE に設定されている場合、手動で SQL コマンドや Server Manager を使用して、インスタンス 1 にインスタンス 2 の REDO ログ・ファイルをアーカイブさせることができます。

自動アーカイブ

LOG_ARCHIVE_START が TRUE に設定されている場合、ARCH バックグラウンド・プロセスは、インスタンス起動時に、自動アーカイブを実行します。自動アーカイブでは、アーカイブを行うインスタンスについてのみ REDO ログ・ファイルをコピーします。

スレッドがクローズされている場合、アクティブなインスタンスのアーカイブ・プロセスでは、ログ・スイッチおよびそのスレッドのアーカイブを実行します。これが行われるのは、使用可能な全スレッド内でのアーカイブ済みログで SCN の範囲をほぼ同じに維持するためにログ・スイッチが全スレッドに対して強制されている場合です。

手動アーカイブ

LOG_ARCHIVE_START を FALSE に設定することにより、次の方法のいずれかによって手動アーカイブを実行できます。

- SQL コマンド ALTER SYSTEM の ARCHIVE LOG 句を使用する。
- SQL コマンド ALTER SYSTEM ARCHIVE LOG START または Server Manager を使用して、自動アーカイブを使用可能にする。

手動アーカイブは、アーカイブ・コマンドを発行するユーザー・プロセスにより実行されます。インスタンスの ARCH プロセスが実行するのではありません。

手動アーカイブ用 ALTER SYSTEM ARCHIVE LOG オプション

ALTER SYSTEM ARCHIVE LOG の手動アーカイブ・オプションには、次のものがあります。

ALL	満杯になっているがアーカイブされていないすべてのオンライン REDO ログ・ファイル
CHANGE	オンライン REDO ログ・ファイル内で最小のシステム変更番号（SCN）
CURRENT	使用可能なすべてのスレッドの現行 REDO ログ
GROUP *integer*	オンライン REDO ログのグループ番号
LOGFILE '*filename*'	スレッド内のオンライン REDO ログ・ファイルのファイル名
NEXT	アーカイブが必要な、次に満杯になる REDO ログ・ファイル
SEQ *integer*	オンライン REDO ログ・ファイルのログ順序番号
THREAD *integer*	アーカイブ対象の REDO ログ・ファイルを含むスレッド（デフォルト値は現行インスタンスに割り当てられたスレッド番号）

ALTER SYSTEM ARCHIVE LOG の THREAD オプションを使用して、現行インスタンス以外のインスタンスと対応付けられたスレッド内の REDO ログ・ファイルをアーカイブできます。

関連項目： 21-10 ページの「ログ・スイッチの強制」（スレッドおよびログ・スイッチについて）。『Oracle8*i* リファレンス・マニュアル』（ALTER SYSTEM ARCHIVE LOG 文の構文について）。『Oracle8*i* バックアップおよびリカバリ・ガイド』および『Oracle8*i* 管理者ガイド』の「REDO 情報のアーカイブ」の章（手動および自動アーカイブについて）

アーカイブ・プロセスの監視

GV$ARCHIVE_PROCESSES ビューと V$ARCHIVE_PROCESSES ビューには、それぞれ、データベースとインスタンスのさまざまな ARCH プロセスの状態に関する情報が含まれています。GV$ARCHIVE_PROCESSES ビューは 10*n 行を表示します（'n' はデータベースのオープン・インスタンスの数）。V$ARCHIVE_PROCESSES ビューは 10 行（各 ARCH プロセスにつき 1 行）を表示します。

関連項目： これらのビューの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

アーカイブ・ファイルのフォーマットとアーカイブ先

アーカイブ済み REDO ログは LOG_ARCHIVE_FORMAT パラメータで指定されるとおり、一意の名前が付けられます。このオペレーティング・システム固有のフォーマットには、テキスト文字列、1 つ以上の変数、ファイル名拡張子を使用できます。
LOG_ARCHIVE_FORMAT には、表 21-1 に示す変数を指定できます。この表の例では、LOG_ARCHIVE_FORMAT= arch%*parameter* であり、すべてのパラメータが 10 文字以下であることが前提となっています。

表 21-1 アーカイブ済み REDO ログ・ファイル名のフォーマット・パラメータ

パラメータ	説明	例
%T	左端に 0 が埋め込まれたスレッド番号	arch0000000001
%t	埋込みなしのスレッド番号	arch1
%S	左端に 0 が埋め込まれたログ順序番号	arch0000000251
%s	埋込みなしのログ順序番号	arch251

スレッドのパラメータ %t および %T は、OPS のみで使用されます。たとえば、REDO スレッド番号 7 に対応するインスタンスで LOG_ARCHIVE_FORMAT を LOG_%s_T%t.ARC に設定した場合、アーカイブ済み REDO ログ・ファイルの名前はそれぞれ次のようになります。

```
LOG_1_T7.ARC
LOG_2_T7.ARC
LOG_3_T7.ARC
...
```

注意： REDO ログ・ファイルを簡単に識別できるように、アーカイブ・ログ・ファイルのフォーマットには常にスレッドおよび順序番号を指定してください。

関連項目： 『Oracle8*i* 管理者ガイド』の「REDO 情報のアーカイブ」の章（アーカイブ済み REDO ログ・ファイル名のフォーマットおよびアーカイブ先の指定について）。『Oracle8*i* 概要』の「回復の構造」の章、システム固有の Oracle マニュアル（デフォルト・ログ・アーカイブのフォーマットおよびアーカイブ先について）

制御ファイルの REDO ログ履歴

CREATE DATABASE コマンドまたは CREATE CONTROLFILE コマンドの MAXLOGHISTORY 句を使用して、パラレル・サーバーが書き込んだ REDO ログ・ファイルの履歴を制御ファイルに保管できます。データベース作成後は、新しい制御ファイルを作成することによってログ履歴を増やすか減らすことだけが可能です。CREATE CONTROLFILE を使用すると、現在の制御ファイル内のログ履歴はすべて破棄されます。

MAXLOGHISTORY オプションでは、アーカイブ履歴に記録可能なエントリ数を指定します。デフォルト値はオペレーティング・システムにより異なります。MAXLOGHISTORY を 0（ゼロ）より大きい値に設定した場合は、インスタンスが 1 つのオンライン REDO ログ・ファイルから他のオンライン REDO ログ・ファイルに切り替わるたびに、そのインスタンスの LGWR プロセスが次のデータを制御ファイルに書き込みます。

- スレッド番号
- ログ順序番号
- 低システム変更番号 (SCN)
- 低 SCN タイムスタンプ
- 次 SCN（つまり、順序の中の次のログの低 SCN）

注意： LGWR が制御ファイルにログ履歴データを書き込むのは、REDO ログ・ファイルのアーカイブ時ではなく、ログ・スイッチ時です。

ログ履歴記録は小さく、ログ履歴が MAXLOGHISTORY で設定された制限を超過すると、古い履歴から順に上書きされます。

回復時に、Server Manager は適切なファイル名を入力するように要求します。Recovery Manager は必要な REDO ログを自動的に復元します。ログ履歴を使用して SCN およびスレッド番号からアーカイブ済みログ・ファイル名を再作成することができ、複数の REDO

スレッドを持つパラレル・サーバーの自動メディア回復を実行します。使用可能なスレッドが 1 つだけの状態で、排他モードでデータベースにアクセスする Oracle インスタンスは、ログ履歴を必要としません。しかし、1 つのスレッドのみがオープンになっている場合でも、複数のスレッドが使用可能であるときはログ履歴がある方が便利です。

V$LOG_HISTORY ビューのログ履歴情報を問い合せることができます。Server Manager を使用している場合は、V$RECOVERY_LOG にはメディア回復に必要なアーカイブ済みログに関する情報も表示されます。この情報はログ履歴の記録が基になっています。

多重 REDO ログ・ファイルには、ログ履歴の複数のエントリは不要です。各エントリが識別するのは、特定のファイル名ではなく、多重 REDO ログ・ファイルのグループです。

関連項目： 22-9 ページの「REDO ログ・ファイルの復元と回復」（回復を実行中の Server Manager プロンプトについて）、システム固有の Oracle マニュアル（MAXLOGHISTORY のデフォルト値について）

アーカイブ・ログのバックアップ

一般的に、アーカイブ・ログにアクセスできるのはそれを作成したノードだけです。OPS には、次の 2 種類のバックアップ・オプションがあります。

- 各ノードでそのノードのアーカイブ・ログをバックアップする
- アーカイブ・ログを 1 つのノードに移動してからバックアップする

どちらの方法もオペレーティング・システム・ユーティリティを使用して手動で実行できます。

ノードとログの親和性

オプションで、RMAN の複数のコピーを各ノードで 1 つずつ実行することによって、各ノードで各ノードのログをバックアップするように設定できます。新しい OEM（Oracle Enterprise Manager）のアーキテクチャでは、1 つのバックアップ・ジョブを作成し、指定した時刻にそれを複数のノードに送ることができます。

Recovery Manager によるアーカイブ・ログのバックアップ

Recovery Manager によって、各ノードでそのノードのアーカイブ・ログを自動的にバックアップできます。ただし、ログを移動する場合は手動で移動を行い、その後で適切な RMAN カタログを使用して、ファイルの移動に合わせてコマンドを変更する必要があります。変更が Recovery Manager に通知されると、Recovery Manager は単一ノードからアーカイブ・ログをバックアップできます。

複数のノードを使用してアーカイブ・ログをバックアップする場合は、Recovery Manager がアーカイブ対象ログのリストを作成するときに、アーカイブ・ログの存在を確認できる必要があります。そのためには、Recovery Manager がすべてのノードのすべてのアーカイブ・ログのヘッダーを読取り可能である必要があります。

次に、各ノードで、そのノードで作成したアーカイブ・ログをバックアップできます。次に示す例では、最初のターゲット・データベースがノード 1（RMAN コマンド行で指定）なので、アーカイブ・ログ（作成されたのはノード 2）のヘッダーが必ずノード 1 から読取り可能である必要があります。

```
RMAN TARGET INTERNAL/KNL@NODE1 RCVCAT RMAN/RMAN@RCAT

RUN {
  ALLOCATE CHANNEL T1 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL T2 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  BACKUP
  FILESPERSET 10
  FORMAT 'AL_%T_%S_%P'
  (ARCHIVELOG UNTIL TIME 'SYSDATE' THREAD 1 DELETE INPUT CHANNEL T1)
  (ARCHIVELOG UNTIL TIME 'SYSDATE' THREAD 2 DELETE INPUT CHANNEL T2);
   )
```

Recovery Manager によるアーカイブ・ログの復元

デフォルトでは、RMAN は接続しているインスタンスの *log_archive_dest* にアーカイブ・ログを復元します。複数のノードを使用して復元と回復を行う場合は、復元 / 回復を実行している任意のノードにアーカイブ・ログが復元されることがあります。復元されたログを実際に読み取ってロールフォワードを実行するノードは、最初に接続された接続先ノードです。ログを回復で使用するためには、必ずそのノードからログが読取り可能であることを確認してください。

ALLOCATE CHANNEL コマンドの CONNECT オプションの使用方法

RMAN ALLOCATE CHANNEL コマンドの CONNECT オプションを使用して、OPS クラスタの任意のノードにチャネルを割り当てることができます。クラスタ内の複数のノードにチャネルを割り当てると、RMAN によりバックアップ処理がそれらのノードの間で自動的に分配されます。

AIX および Pyramid Mesh クラスタでは、RMAN によりディスクとノードの親和性が自動的に検出され、データファイルに最も速くアクセスできるノードにデータファイルがバックアップされます。

複数の OPS ノードにバックアップを作成するには、メディア・マネージャによる次のサポートが必要です。

- メディア・マネージャは、任意のノードからアクセスできる中央カタログを持っている必要がある。
- 使用するすべてのノードにテープ・ドライブを備えるか、メディア・マネージャによりネットワークを介してバックアップ / 復元をサポートする必要がある。

チェックポイントおよびログ・スイッチ

この項では、次の項目について説明します。

- チェックポイント
- チェックポイントの強制
- ログ・スイッチの強制
- クローズ状態のスレッドに対するログ・スイッチの強制

チェックポイント

Oracle8*i* では、チェックポイント処理は一貫して自動的に実行されます。チェックポイント処理では、Oracle によりすべての使用済みバッファをディスクに書き込み、チェックポイントを進める必要があります。

関連項目： チェックポイントおよび Oracle によるチェックポイント処理の制御方法は、『Oracle8*i* チューニング』を参照してください。

チェックポイントの強制

SQL 文の ALTER SYSTEM CHECKPOINT により、現行インスタンスまたは全インスタンスのいずれかに対するチェックポイントが明示的に強制されます。チェックポイントを強制することによって、データベース・バッファの変更がすべてディスク上のデータ・ファイルに書き込まれます。

ALTER SYSTEM CHECKPOINT の GLOBAL オプションは、デフォルトです。このオプションは、データベースをオープンしたインスタンスすべてに強制的にチェックポイントを実行させます。LOCAL オプションは、現行インスタンスにチェックポイントを強制します。

グローバル・チェックポイントは、回復を必要とするすべてのインスタンスが回復するまで終了しません。ただし、グローバル・チェックポイント時にインスタンスに障害が発生すると、そのインスタンスが回復する前にチェックポイントが終了することもあります。

リモート・ノード上で稼働中のインスタンスでチェックポイントを強制するには、Server Manager コマンドの CONNECT を使用して現行インスタンスを変更できます。

注意： チェックポイントを強制するには、ALTER SYSTEM 権限が必要です。

関連項目： 18-16 ページの「インスタンスの指定」（リモート・ノードの指定について）

ログ・スイッチの強制

インスタンスが多くの REDO エントリを生成していない場合、またはインスタンスが停止した場合のどちらかの理由で、ある期間のオンライン REDO ログ・ファイルのアーカイブに失敗した任意のインスタンスに対して、パラレル・サーバーはログ・スイッチを強制できます。これによって、インスタンスの REDO ログ、つまり REDO のスレッドが長い間アーカイブされないままになることを防ぎます。メディア回復が必要な場合、常に最近の REDO エントリが回復処理に使用されます。

たとえば、あるインスタンスが停止すると、そのインスタンスの現行ログ・ファイルがアーカイブされるように、別のインスタンスがそのインスタンスのためにログ・スイッチを強制することができます。

SQL 文の ALTER SYSTEM SWITCH LOGFILE は、現行 REDO ログ・ファイルが満杯であるかどうかにかかわらず、現行インスタンスに新しい REDO ログ・ファイルに書込みを開始するように強制します。

すべてのインスタンスにグローバル・ログ・スイッチの実行を強制します。そのためには、SQL 文の ALTER SYSTEM ARCHIVE LOG CURRENT 文で THREAD キーワードを省略します。この文を発行すると、Oracle は制御をユーザーに戻す前にすべてのオンライン REDO ログ・ファイルがアーカイブされるまで待ちます。単一のインスタンスにログ・スイッチを強制し、そのオンライン REDO ログ・ファイルをアーカイブするには、この ALTER SYSTEM ARCHIVE LOG CURRENT 文で THREAD キーワードを指定します。

Server Manager では、「Instance Force Log Switch」オプションは現行インスタンスにのみ使用してください。Server Manager では、ログ・スイッチを強制するグローバル・オプションはありません。現行 REDO ログ・ファイルをアーカイブ、削除、改名できるようにするために、ログ・スイッチを強制することもあります。

注意： ログ・スイッチを強制するには、ALTER SYSTEM 権限が必要です。

関連項目： 6-3 ページの「REDO ログ・ファイル」（スレッドについて）

クローズ状態のスレッドに対するログ・スイッチの強制

データベースがオープンされている間にログ・スイッチを終了するようクローズ状態のスレッドを強制できます。これは、そのスレッドの現在のログを削除するときに役立ちます。この方法は、スレッドがオープンしているインスタンスが停止した場合でも、オープン・スレッド（現在のスレッドを含む）では有効ではありません。たとえば、スレッドがオープンしている間にインスタンスが異常終了した場合、スレッドのログを切り替えることはできません。

クローズ状態のスレッドにログ・スイッチを強制するには、Server Manager の「Begin Manual Archive」ダイアログ・ボックスを使用するか、SQL コマンドの ALTER SYSTEM コ

マンドで ARCHIVE LOG オプションを指定して、スレッドを手動でアーカイブします。次に例を示します。

```
ALTER SYSTEM ARCHIVE LOG GROUP 2;
```

クローズ状態の REDO ログ・グループを手動でアーカイブすると、ログ・スイッチを強制することになりますが、このためには、SYSOPER 権限または SYSDBA 権限で接続する必要があります。

関連項目：『Oracle8*i* 管理者ガイド』（SYSDBA 権限または SYSOPER 権限での接続について）

データベースのバックアップ

この項では、OPS でのバックアップ操作について説明します。次の項目について説明します。

- オープン・データベース・バックアップとクローズド・データベース・バックアップ
- Recovery Manager でのバックアップ
- オペレーティング・システムでのバックアップ

オープン・データベース・バックアップとクローズド・データベース・バックアップ

すべてのバックアップ動作は、パラレル・サーバーのどのノードからでも実行できます。オープン・バックアップでは、実行中のデータベースの全体または一部をバックアップできます。オープン・バックアップ中、ユーザーはデータベースにアクセスしてデータベースの任意の部分のデータを更新できます。パラレル・サーバーでは、異なるノードから同時に複数の表領域のオープン・バックアップを作成できます。オープン・バックアップには、1 つ以上のデータ・ファイルと現在の制御ファイルのコピーが含まれています。また、メディア障害発生時点までの回復ができるように、それ以降のアーカイブ REDO ログ・ファイルまたは増分バックアップも必要です。

オペレーティング・システムを使用する場合、クローズド・バックアップはデータベースがクローズされた状態のときに行います。Recovery Manager を使用する場合、クローズド・バックアップを実行するには、インスタンスが起動済みかつマウント済みでオープンしていない状態になっている必要があります。クローズド・バックアップを作成するには、その前にパラレル・サーバーのすべてのインスタンスをシャットダウンしてください。データベースがクローズ状態のときは、異なるノードから並列にファイルをバックアップできます。クローズド・データベースの全体バックアップには、すべてのデータ・ファイルおよび現在の制御ファイルのコピーが含まれています。

REDO ログ・ファイルをアーカイブする場合、クローズド・バックアップを使用することにより、メディア障害発生時点までの回復が可能です。クローズド・バックアップではバック

アップ時点までのデータベースしか回復できないので、NOARCHIVELOG モードでの完全な回復は不可能です。

警告 : 全体クローズド・バックアップを実行する場合以外は、オペレーティング・システムのユーティリティを使用した ARCHIVELOG モードでの制御ファイルのバックアップは行わないでください。

別の完全なオープンまたはクローズド・バックアップ（できれば 2 組の完全バックアップ）を作成するまでは、アーカイブ済み REDO ログ・ファイルを消去、再使用または破棄しないでください。

> **関連項目 :** 『Oracle8*i* バックアップおよびリカバリ・ガイド』『Oracle8*i* 概要』の「データベースのバックアップ」および「データベースの回復」

Recovery Manager でのバックアップ

Recovery Manager でのスナップショット制御ファイルの準備

OPS では、Recovery Manager によるバックアップを実行する前に、スナップショット制御ファイルの準備をする必要があります。

バックアップを作成するすべてのノードで、スナップショット制御ファイルを作成する必要があります。したがって、バックアップに使用するすべてのノードで、スナップショット制御ファイルを書き込むディレクトリが存在していることを確認してください。

たとえば、スナップショット制御ファイルを /oracle/db_files/snapshot/snap_prod.cf ファイルに書き込む場合は、次のように入力します。

```
SET SNAPSHOT CONTROLFILE TO '/ORACLE/DB_FILES/SNAPSHOT/SNAP_PROD.CF';
```

次に、バックアップを実行するすべてのノードにディレクトリ **/oracle/db_files/snapshot** が存在することを確認します。

スナップショット制御ファイルの書込み先としてロー・デバイスを指定することもできます。その場合、スナップショット制御ファイルは OPS の他のデータ・ファイルと同様に、クラスタ内の全ノードによって共有されます。

Recovery Manager を使用したオープン・バックアップの実行

Recovery Manager を使用したオープン・バックアップの詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

アーカイブ・ログもバックアップする場合は、バックアップ完了後に ALTER SYSTEM ARCHIVE LOG CURRENT 文を発行します。この文によって、バックアップしたファイルを一貫性のあるものにするためのすべての REDO を確実に獲得できます。

次のサンプル・スクリプトでは、パラレル・サーバー環境の 2 つのインスタンスにデータ・ファイルおよびアーカイブ・ログ・バックアップを分散しています。次のことを想定しています。

- データベースには 20 ファイル以上のファイルがある
- 1 ノードに 2 台ずつ、計 4 台のテープ・ドライブがある
- スレッド 2 が作成したアーカイブ・ログ・ファイルはノード 1 から読込み可能である

サンプル・スクリプトを次に示します。

```
RUN {
  ALLOCATE CHANNEL NODE1_T1 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL NODE1_T2 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL NODE2_T3 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  ALLOCATE CHANNEL NODE2_T4 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  BACKUP
   FILESPERSET 6
   FORMAT 'DF_%T_%S_%P'
   (DATABASE);
  SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
  BACKUP
    FILESPERSET 10
    FORMAT 'AL_%T_%S_%P'
    (ARCHIVELOG UNTIL TIME 'SYSDATE' LIKE 'node1_archivelog_dest%' DELETE
     INPUT CHANNEL NODE1_T1)
    (ARCHIVELOG UNTIL TIME 'SYSDATE' LIKE 'node2_archivelog_dest%' DELETE
     INPUT CHANNEL NODE2_T3);
```

ノードの親和性の認識

一部のクラスタ・プラットフォームでは、クラスタの特定のノードが一部のデータファイルにアクセスするとき、他のデータファイルよりも高速にアクセスできます。そうした親和性は、RMAN によって自動的に検出されます。特定のデータファイルをバックアップするチャネルを決定するとき、RMAN は、そのデータファイルへの親和性を持つノードに割り当てられたチャネルに優先度を与えます。この機能を使用するには、バックアップするデータファイルへの親和性を持つ複数のノードに RMAN のチャネルを割り当てます。

次に例を示します。

```
RUN
{
  ALLOCATE CHANNEL CH1 TYPE 'SBT_TAPE' CONNECT '@INST1';
  ALLOCATE CHANNEL CH2 TYPE 'SBT_TAPE' CONNECT '@INST2';
  ...
}
```

注意： ALLOCATE コマンドの CONNECT オペランドの詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

オペレーティング・システムでのバックアップ

この項では、次の項目について説明します。

- オペレーティング・システム・ユーティリティを使用したオープン・バックアップの開始と終了
- オペレーティング・システム・ユーティリティを使用したオープン・バックアップの実行

オペレーティング・システム・ユーティリティを使用したオープン・バックアップの開始と終了

オペレーティング・システムの方法を使用する場合、表領域のオープン・バックアップは 1 つのインスタンスから開始でき、同じインスタンスでも別のインスタンスでもそのバックアップを終了できます。次に例を示します。

```
ALTER TABLESPACE TABLESPACE BEGIN BACKUP;/* INSTANCE X */
Statement processed.

....OPERATING SYSTEM COMMANDS TO COPY DATAFILES...
....COPY COMPLETED...

ALTER TABLESPACE TABLESPACE END BACKUP;/* INSTANCE Y */
Statement processed.
```

注意： オペレーティング・システムによる表領域のバックアップを開始するまでに ALTER TABLESPACE ... BEGIN BACKUP コマンドを発行しなかった場合、またはこのコマンドの実行が完了していなかった場合は、バックアップしたデータファイルはそれ以降の回復処理では使用できません。このようなバックアップを回復しようとすると危険です。これはエラーの原因となり、データに不整合が生じる可能性があります。

オープン・バックアップ作成時には、これらの文を必ず発行する必要がありますが、どのインスタンスがどの文を発行するかは問題ではありません。BEGIN BACKUP オプションは、ユーザーの表領域へのアクセスには影響を与えません。

完全なメディア回復または不完全なメディア回復に使用できるオープン・バックアップを作成するには、BEGIN BACKUP コマンドの実行から回復の終了時点までの間のアーカイブ済み REDO ログをすべて保持する必要があります。

オープン・バックアップを作成した後、ALTER SYSTEM ARCHIVE LOG CURRENT を使用してグローバル・ログ・スイッチを強制できます。これには、現行 REDO ログ・ファイルをアーカイブしないで停止した任意のインスタンスの、すべての使用可能スレッドとクローズ状態のスレッドの現行オンライン REDO ログ・ファイルが含まれます。

関連項目：『Oracle8*i* SQL リファレンス』（ALTER TABLESPACE コマンドの BEGIN BACKUP 句および END BACKUP 句について）

オペレーティング・システム・ユーティリティを使用したオープン・バックアップの実行

OPS でオペレーティング・システム・ユーティリティを使用したオープン・バックアップを実行する場合は、次の手順に従うことをお薦めします。

1. オープン・バックアップを開始する前に、ALTER SYSTEM ARCHIVE LOG CURRENT コマンドを発行します。

   これによって、OPS のすべてのスレッドの現行 REDO ログ・ファイルが切り替えられ、アーカイブされます。現在使用可能でないスレッドも対象となります。

2. ALTER TABLESPACE *tablespace* BEGIN BACKUP コマンドを発行します。
3. ALTER TABLESPACE コマンドが正常に終了するまで待ちます。
4. オペレーティング・システム環境で、適切なコマンドを発行して表領域のデータ・ファイルをバックアップします。
5. オペレーティング・システムのバックアップが正常に終了するまで待ちます。
6. ALTER TABLESPACE *tablespace* END BACKUP コマンドを発行します。
7. ALTER DATABASE BACKUP CONTROLFILE TO *filename* を使用して制御ファイルをバックアップします。

念のため、ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS コマンドを使用して、制御ファイルのバックアップをトレース・ファイルに格納します。その後、そのトレース・ファイルを識別してバックアップします。

アーカイブ・ログもバックアップする場合は、END BACKUP の後に ALTER SYSTEM ARCHIVE LOG CURRENT 文を発行します。この文によって、END BACKUP 時点まで戻るためのすべての REDO を確実に獲得できます。

22

データベースの回復

この章では、パラレル・サーバーでの Oracle の回復機能について説明します。次の項目について説明します。

- 概要
- インスタンス障害の回復
- メディア障害の回復
- パラレル回復

概要

この章では、次の 3 種類の回復方法について説明します。

表 22-1 回復のタイプ

回復のタイプ	定義
インスタンス障害	ソフトウェアまたはハードウェアの問題により、インスタンスが作業を継続できない場合に発生します。
メディア障害	Oracle ファイルの記憶域メディアが破損した場合に発生します。メディア障害が起こると、通常、Oracle によるデータの読込みまたは書込みができなくなります。
パラレル回復	Recovery Manager の場合、チャネル割当てを使用して回復と増分バックアップの適用がパラレル化されます。 REDO の適用（Recovery Manager によって行われるか Server Manager によって行われるかにかかわらず）は、RECOVERY_PARALLELISM パラメータによって決まります。

インスタンス障害の回復

次の項では、共有モードでデータベースにアクセスしているインスタンスに障害が発生した後で行われる回復について説明します。

- 単一ノード障害
- 複数ノード障害
- ファースト・スタート・チェックポイント処理
- ファースト・スタート・ロールバック
- インスタンス回復のためのデータ・ファイルへのアクセス
- インスタンス回復のためのデータベースの凍結
- Oracle インスタンス回復のフェーズ

インスタンス障害の発生後、Oracle はオンライン REDO ログ・ファイルを使用してデータベースの自動回復を実行します。排他モードで稼動している単一インスタンスの場合、障害発生後または異常停止後にそのインスタンスを再起動するとすぐに、インスタンスの回復が実行されます。

共有モードでデータベースにアクセスしているインスタンスで障害が発生すると、オンラインのインスタンス回復が自動的に実行されます。他のノード上で稼働し続けているインスタンスは、バッファ・キャッシュから読込みを行っている限り、その影響は受けません。インスタンスが書込みを実行しようとすると、そのトランザクションは停止します。障害インスタンスのキャッシュ回復が終了するまで、そのデータベースに対する操作はすべて保留にされます。

関連項目：『Oracle8*i* バックアップおよびリカバリ・ガイド』

単一ノード障害

Oracle Parallel Server（OPS）は、他の稼働中インスタンスの SMON プロセスを通して回復操作を調整することによって、インスタンスの回復を実行します。1 つのインスタンスで障害が発生すると、他のインスタンスの SMON プロセスが障害を検出し、障害インスタンスに対してインスタンス回復を自動的に実行します。

インスタンス回復では、障害インスタンスの再起動、またはそのインスタンスで実行中のアプリケーションの再起動は行われません。実行中であったアプリケーションは、フェイルオーバーによって実行を継続できます。詳細は、22-2 ページの「インスタンス障害の回復」を参照してください。

あるインスタンスが、障害が発生している別のインスタンスの回復を実行すると、回復を行うインスタンスは、障害インスタンスが作成した REDO ログ・エントリを読み込み、その情報を使用して、コミットされたトランザクションがすべてデータベース内で反映されるようにします。コミットされたトランザクションのデータが失われることはありません。回復

実行中のインスタンスは、障害発生時にアクティブであったトランザクションをすべてロールバックし、そのトランザクションが使用していたリソースを解放します。

複数ノード障害

1 つのインスタンスが稼働し続けているかぎり、SMON プロセスは、パラレル・サーバーのその他の障害インスタンスに対してインスタンス回復を実行します。

パラレル・サーバー内にあるすべてのインスタンスで障害が発生した場合、次回のあるインスタンスがデータベースをオープンしたときに、インスタンス回復が自動的に実行されます。そのインスタンスは、障害を起こしたインスタンスの 1 つである必要はありません。また、そのインスタンスは、パラレル・サーバーのどのノードからでも、共有モードまたは排他モードでデータベースをマウントできます。この回復手順は、共有モードで実行している Oracle の場合も排他モードで実行している Oracle の場合も同じです。ただし、共有モードで実行している Oracle では、障害が発生したすべてのインスタンスの回復を 1 つのインスタンスが実行します。

ファースト・スタート・チェックポイント処理

ファースト・スタート・チェックポイント処理は、Oracle8*i* のファースト・スタート障害回復の基礎になる処理です。ファースト・スタート・チェックポイント処理は連続して発生し、Oracle によってブロックがディスクに書き込まれたときに、チェックポイントを進めます。ファースト・スタート・チェックポイント処理では必ず最初に一番古い変更済みブロックを書き込み、すべての書込みでチェックポイント時間を進められるようにします。これにより、従来のチェックポイント処理で発生する大量の書込みとそれに伴う I/O の突出がなくなり、継続して一定の効率的なパフォーマンスを得ることができます。

ファースト・スタート・チェックポイント処理のロールフォワード・フェーズにかかる時間の長さについて、制限を指定できます。Oracle では、指定されたロールフォワードの制限に応じてチェックポイントの書込み率を自動的に調整し、発行する書込みの数を最小限に抑えます。これがどのように行われるかの詳細は、『Oracle8*i* チューニング』を参照してください。

ファースト・スタート・ロールバック

Oracle8*i* では、システム障害回復のロールバック・フェーズで " 非ブロック化 " ロールバック・テクノロジを採用しています。そのため、ロールフォワード完了直後に新しいトランザクションを開始できます。新しいトランザクションがアクセスする行が、使用不能なトランザクションによってロックされている場合、新しいトランザクションは、新しいトランザクションの進行を妨げる変更のみをロールバックします。新しいトランザクションは使用不能なトランザクション全体が Oracle によってロールバックされるまで待つ必要がないので、実行時間の長いトランザクションが回復時間に影響することはありません。ファースト・スタート・テクノロジにより、データの可用性を最大化し、回復時間を確実に予測できます。

さらに、データベース・サーバーは使用不能なトランザクションをパラレルにロールバックできます。この技術は行に対して使用され、新しいトランザクションをブロックしません。また、使用不能なトランザクションのロールバックをパラレルに実行する方が、シリアルに実行するよりもコストが小さい場合にのみ使用されます。

関連項目：『Oracle8*i* 概要』

インスタンス回復のためのデータ・ファイルへのアクセス

他のインスタンスに対して回復を実行するインスタンスは、障害インスタンスがアクセスしていたすべてのオンライン・データ・ファイルに対してアクセス権を持っている必要があります。データ・ファイルが検査で不合格となったためにインスタンス回復が失敗した場合、回復を実行しようとしたインスタンスには障害は起こりませんが、ALERT ファイルにメッセージが書き込まれます。

データベース・ファイルへのアクセスを妨げていた問題を解決したら、SQL 文の ALTER SYSTEM CHECK DATAFILES を使用して、インスタンスを使用できるようにしてください。

関連項目： 6-2 ページの「データ・ファイル」

インスタンス回復のためのデータベースの凍結

OPS では、インスタンス回復中のデータベースの凍結を制御する動的パラメータ FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY を使用できます。このパラメータは、すべてのインスタンスで同じ値にする必要があります。

このパラメータを TRUE に設定すると、インスタンス回復中に Oracle はデータベース全体を凍結します。データベース全体を凍結する利点は、インスタンス回復以外のディスク・アクティビティが停止されることです。これにより、インスタンス回復をより速く完了することができます。データベース全体を凍結した場合の欠点は、インスタンス回復中にデータベースが使用不可能になることです。

このパラメータを FALSE に設定すると、Oracle はデータベースを凍結しないので、インスタンスの回復中であっても凍結による影響を受けていないデータベース部分にアクセスできます。

システムは、適切なデフォルト値をインテリジェントに選択しようとします。

- すべてのオンライン・データ・ファイルがハッシュ・ロックを使用している場合、このパラメータのデフォルト値は FALSE です。その理由は、ハッシュ・ロックを使用している場合、インスタンスの回復中でもユーザーはデータベースの大部分にアクセスできるからです。
- データ・ファイルがファイン・グレイン・ロックを使用している場合、デフォルトは TRUE です。ファイン・グレイン・ロックを使用している場合、データベースの大部分がインスタンスの使用不能による影響を受ける可能性があります。インスタンスが回復した後でなければ、影響を受けたデータへのアクセスはできません。したがってこの場

合は、このパラメータを TRUE に設定した方が、影響を受けたデータベース部分が短時間で使用可能になる可能性があります。

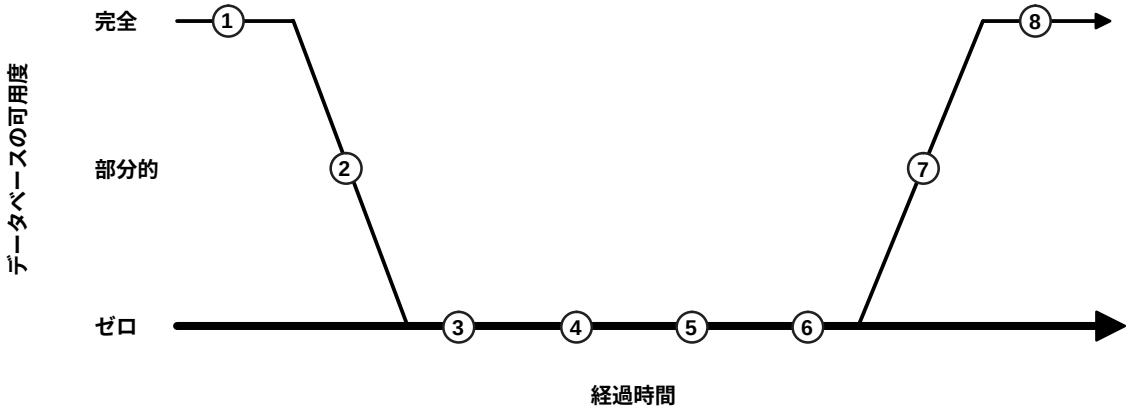
インスタンスの起動後にインスタンス回復のためにデータベース全体が凍結された回数を知るには、V$SYSSTAT の "instance recovery database freeze count" 値を調べます。

関連項目：『Oracle8*i* リファレンス・マニュアル』

Oracle インスタンス回復のフェーズ

図 22-1 に、Oracle インスタンス回復の各フェーズにおけるデータベースの可用度を示します。

図 22-1 Oracle インスタンス回復のフェーズ



回復フェーズは次のとおりです。

1. OPS が複数のノード上で稼働中です。
2. ノード障害が検出されます。
3. LM が再構成され、リソースおよびロック管理が、障害を起こしていないノードの集合に再分散されます。1 回のコールで持続リソースを獲得できます。ロック値ブロックには、排他モードまたは書込み禁止モードに保持されているロックの不確定マークが付きます。ロック要求は、待ち行列に入ります。
4. LCK*n* プロセスが、無効なロック要素すべてのリストを作成します。
5. ロール・フォワードします。デッド・スレッドの REDO ログがデータベースに適用されます。
6. LCK*n* プロセスが、無効なロック要素をすべて有効にします。

7. ロールバックします。コミットしていないトランザクションすべてについて、ロールバック・セグメントがデータベースに適用されます。
8. インスタンス回復が完了し、すべてのデータはアクセス可能になります。

5 番目のフェーズ（REDO ログのフォワード適用）中、データベース・アクセスは、バッファ・キャッシュの転送状態により制限を受けます。ハッシュ・ロックを使用している場合でも、ファイン・グレイン・ロックを使用している場合でも、また特定の機能を使用している場合でも、すべてのデータ・ファイルにあるすべてのユーザー・データには、次のデータ・アクセス上の制約があります。

- このアクセスが制限されている間は、残っているバッファ・キャッシュには書込みできません。
- バッファ・キャッシュおよびダイレクト・パスを介して行われるディスク I/O は、どの種類の I/O であっても、残っているインスタンスからは実行できません。
- IDLM のユーザー・データに対するロック要求は行われません。

正しいグローバル・ロックを持つキャッシュの中にすでに存在しているバッファは読込み可能です。これは、これらのバッファが I/O もロック・オペレーションも必要としないからです。

バッファ・キャッシュの転送状態は、最初のロック走査フェーズが終了すると開始します。この時点で、インスタンス回復は、デッド REDO スレッドがあるかどうか走査することにより始められます。新しい " デッド " スレッドが見つかると、次のロック走査が実行されます。この状態は、REDO ログが適用されている間（キャッシュ回復中）継続し、REDO ログが適用され、ファイル・ヘッダーが更新されると終了します。キャッシュ回復動作は、無効ロックの妥当性走査で終了します。この妥当性走査は、バッファ・キャッシュの状態が正規化されてから行われます。

メディア障害の回復

メディア障害が発生して 1 つ以上のデータベース・ファイルが損失した場合は、データ・ファイルのバックアップを使用してデータベースを回復してください。

Recovery Manager を使用する場合は、増分バックアップやアーカイブ済み REDO ログ・ファイル、制御ファイルのバックアップを適用することが必要な場合があります。

オペレーティング・システムのユーティリティを使用する場合は、アーカイブした REDO ログ・ファイルをデータベースに適用し、制御ファイルのバックアップを使用することが必要な場合があります。

この項では、次の項目について説明します。

- 完全メディア回復
- 不完全メディア回復
- REDO ログ・ファイルの復元と回復

- 災害時回復

関連項目：『Oracle8*i* バックアップおよびリカバリ・ガイド』（さまざまなメディア障害の回復手順について）

完全メディア回復

完全メディア回復は、排他モードと共有モードのどちらでも実行できます。表 22-2 は、特定のデータベース・オブジェクトを回復するときに必要なデータベースの状態を示しています。

表 22-2 メディア回復に必要なデータベースの状態

回復対象	データベースの状態
データベース全体または SYSTEM 表領域	データベースはマウントされる必要がありますが、どのインスタンスもオープンしてはなりません。
SYSTEM 表領域以外の表領域	データベースは、回復を行うインスタンスによってオープンされており、表領域はオフラインになっている必要があります。
データ・ファイル	データベースは、データ・ファイルがオフラインの状態でオープンしているか、またはデータベースがマウントされていて、どのインスタンスからもオープンされていない状態でもかまいません。（SYSTEM 表領域中のデータ・ファイルの場合、データベースはマウントされている必要がありますが、オープンしないでください。）

同時に複数のインスタンス上の複数のデータ・ファイルまたは表領域を回復できます。

オペレーティング・システム・ユーティリティを使用した完全メディア回復

オペレーティング・システム・ユーティリティを使用すると、表領域またはデータ・ファイルのオープン・データベース回復を共有モードで実行できます。そのためには、Server Manager コマンドの RECOVER TABLESPACE または RECOVER DATAFILE を使用します。

Server Manager の RECOVER DATABASE コマンドを使用して、共有モードでマウントされた、オープンしていないデータベースを回復できます。OPS では、1 つのインスタンスだけがこのコマンドを発行できます。

注意： データベースの回復には、Server Manager を使用する方法をお薦めします。SQL コマンドの ALTER DATABASE RECOVER を直接使用する方法はお薦めしません。

Recovery Manager を使用した完全メディア回復

Recovery Manager を使用すると、次の文を実行してファイルの復元と回復を行うことができます。

- RESTORE DATABASE
- RESTORE TABLESPACE
- RESTORE DATAFILE
- RECOVER DATABASE
- RECOVER TABLESPACE
- RECOVER DATAFILE

OPS の Recovery Manager で使用するコマンドは、単一インスタンス環境を回復するときに使用するコマンドと同じです。

> **関連項目：** 詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

不完全メディア回復

不完全メディア回復は、データベースを共有モードまたは排他モードでマウントして、どのインスタンスからもオープンしていない場合に実行できます。この回復には、次のデータベース回復オプションを使用します。

Recovery Manager の場合、復元と回復の前に SET コマンドで次のいずれかのオプションを使用します。

- UNTIL CHANGE *integer*
- UNTIL TIME *date*
- UNTIL LOGSEQ *integer* THREAD *integer*

オペレーティング・システム・ユーティリティの場合は、適切なバックアップを復元してから RECOVER DATABASE コマンドで次のいずれかのオプションを使用します。

- UNTIL CANCEL
- UNTIL CHANGE *integer*
- UNTIL TIME *date*

> **関連項目：** 『Oracle8*i* バックアップおよびリカバリ・ガイド』

REDO ログ・ファイルの復元と回復

OPS がアクセスしたデータベースのメディア回復を行うために、複数のアーカイブ済みログ・ファイルを同時にオープンすることが必要な場合があります。各インスタンスは REDO ログ・データを別々の REDO スレッドに書き込むので、回復のために、それぞれのスレッドに 1 つずつアーカイブ済みログ・ファイルが必要な場合があります。

ただし、スレッドのオンライン REDO ログに十分な回復情報が含まれている場合には、そのスレッドのためにアーカイブ済みログ・ファイルを復元する必要はありません。

Recovery Manager を使用した回復

Recovery Manager は、必要なアーカイブ・ログを自動的に復元して適用します。デフォルトでは、Recovery Manager は接続しているインスタンスの LOG_ARCHIVE_DEST ディレクトリにアーカイブ・ログを復元します。これは、複数のノードを使用して復元と回復を行う場合に、復元 / 回復を実行している任意のノードにアーカイブ・ログが復元されることを意味します。

復元されたログを実際に読み取ってロールフォワードを実行するノードは、最初に行われた接続の接続先ノードです。そのノードからログが読込み可能であることを確認してください。

> **関連項目：**『Oracle8*i* バックアップおよびリカバリ・ガイド』（Recovery Manager がアーカイブ・ログを復元する位置の変更について）

オペレーティング・システム・ユーティリティを使用した回復

回復中にアーカイブ済みログ・ファイルが必要になると、ファイル名の入力を求める Oracle のプロンプトが表示されます。必要なファイルについての情報を示すメッセージが表示され、ファイル名の入力を求める Oracle のプロンプトが表示されます。

たとえば、ログ履歴が使用可能であり、ファイル名フォーマットが LOG_T%t_SEQ%s の場合（ここで %t はスレッド、%s はログ順序番号を示します）、スレッド 8 内の SCN 9523 を使用して回復を開始するように、次のメッセージが表示されることがあります。

```
ORA-00279: Change 9523 generated at 27/09/91 11:42:54 needed for thread 8
ORA-00289: Suggestion : LOG_T8_SEQ438
ORA-00280: Change 9523 for thread 8 is in sequence 438
Specify log: {<RET> = suggested | filename | AUTO | FROM | CANCEL}
```

Server Manager を使用しないで RECOVER 句を指定した ALTER DATABASE 文を使用すると、これらのメッセージが表示されますが、プロンプトは表示されません。OPS 内の使用可能な各スレッドに、REDO ログ・ファイルが必要な場合もあります。ログ・ファイルが不要になると Oracle のメッセージが表示されます。そのスレッドが使用禁止にされたか、回復が終了していない限り、そのスレッドの次のログ・ファイルが要求されます。

追加スレッドが使用可能となる時点まで回復が進むと、Oracle はそのスレッドに対してだけアーカイブ済みログ・ファイルを要求します。インスタンスがスレッドを使用可能にしたと

きは必ず、変更を記録する REDO エントリを書き込みます。したがって、スレッドについての必要な情報はすべて、回復処理中に REDO ログ・ファイルから使用できます。

あるスレッドが使用禁止となった時点まで回復が進むと、Oracle は、そのスレッドのログ・ファイルが必要なくなったことをユーザーに知らせ、そのスレッドのログ・ファイルをそれ以上要求しません。

注意： Oracle がアーカイブ済み REDO ログ・ファイルの名前を再作成する場合、回復を行っているインスタンス用に LOG_ARCHIVE_FORMAT で指定されているフォーマットは、ファイルをアーカイブしたインスタンス用に指定されているフォーマットと同じにする必要があります。OPS で使用する LOG_ARCHIVE_FORMAT の値は、全インスタンスに同一です。また回復を実行しているインスタンスも、その値を使用します。アーカイブ済み REDO ログ・ファイルが元のアーカイブ先にない場合は、回復実行中に LOG_ARCHIVE_DEST の値を変えて指定できます。

災害時回復

この項では、Recovery Manager を使用した災害時回復およびオペレーティング・システム・ユーティリティを使用した災害時回復について説明します。災害時回復は、障害によってサイト全体が使用不能となったときに行われます。この場合、かわりのサイトで、データベースのオープン・バックアップまたはクローズド・バックアップを使用して回復を行うことができます。

注意： 最新時点までの回復を行うには、リモート・サイトですべてのログが使用可能でなければなりません。そうでないと、コミットされたトランザクションの一部が失われることがあります。

Recovery Manager を使用した災害時回復

次のシナリオを想定します。

- データベース全体とすべての制御ファイル、およびオンライン REDO ログが失われました。
- 2 つのノードに分散して回復します。
- 磁気テープ装置が 4 台（各ノードに 2 台ずつ）あります。
- リカバリ・カタログを使用します。

注意： データベース・リセット・ログをオープンした後、ただちにデータベースのバックアップをとることを強くお薦めします。それ以前のバックアップはすべて無効になるからです。このステップは例には示されていません。

また、最新のバックアップでデータベースの構造が変更されている場合、その時点まで回復するには SET UNTIL コマンドを使用します。そうすると、指定の時点におけるデータベース構造と同じ構造を持つデータベースが、Recovery Manager によって復元されます。

作業開始前 : データベースの復元を開始する前に、次のことを行う必要があります。

- 最新のバックアップから初期設定ファイルおよびリカバリ・カタログを復元します。
- ディスク上にあってリカバリ・カタログに登録されていないアーカイブ・ログ、またはデータ・ファイル・コピー、バックアップ・コピー・セットを登録します。

  復元されるログ順序番号までのアーカイブ・ログが、必ずリカバリ・カタログに登録されている必要があります。そうでない場合、Recovery Manager にはアーカイブ・ログの場所がわかりません。

  リカバリ・カタログの再同期を頻繁に実行しており、復元に用いた最新のコピーがある場合は、登録する必要があるアーカイブ・ログの数は多くありません。

注意 : このステップを実行する必要があるのは、すでに復元してその時点までの回復を実行済みであるリカバリ・カタログを失った場合のみです。リカバリ・カタログが正常なまま残っている場合、このステップは不要です。ただし正常なカタログが残っていても、少数のアーカイブ済みログをカタログに登録する必要が生じることがありますが、最後の " カタログの再同期化 " 以降に作成されたログを再作成するだけです。" カタログの再同期化 " とは、バックアップ、コピーおよびアーカイブログの情報を rman によってターゲット・データベースの制御ファイルからリカバリ・カタログにコピーする処理です。

サンプル・スクリプトの実行内容 : 次のスクリプトでは、データベースを使用可能な最新のアーカイブ済みログ（ログ 124 スレッド 1）の状態まで回復しています。実行内容は次のとおりです。

- データベース NOMOUNT を起動し、接続を DBA ユーザーだけに制限します。
- 指定された位置に制御ファイルを復元します。
- CONTROL_FILES 初期化パラメータで指定した他のすべての位置にこの制御ファイルをコピー（またはレプリケート）します。
- 制御ファイルをマウントします。
- リカバリ・カタログにないアーカイブ・ログを登録します。
- データベース・ファイルを（元の位置に）復元します。

  ボリューム名が変更されている場合は、復元の前に SET NEWNAME FOR... 文を実行し、復元後に切り替える必要があります。これにより、制御ファイルがデータファイルの新しい位置で更新されます。

- 増分バックアップと REDO を組み合せて使用するか REDO だけを使用して、データ・ファイルを回復します。

  指定したログ順序番号に達すると、Recovery Manager は回復を完了します。

- データベース・リセット・ログをオープンします。

> **注意：** 適用できるアーカイブ・ログが他にないことが確実な場合にだけ、次の手順を実行してください。

- リセット・ログの後にデータベースのバックアップをとることをお薦めします。これは例には示されていません。

復元 / 回復のサンプル・スクリプト：

DBA は次のようにして Server Manager を起動します。

```
CONNECT SCOTT/TIGER AS SYSDBA
```

Oracle から次の応答が表示されます。

```
Connected.
```

次の STARTUP 構文を入力します。

```
STARTUP NOMOUNT RESTRICT
```

DBA は Recovery Manager を起動してスクリプトを実行します。

> **注意：** ターゲット・パラメータで指定するユーザーは、SYSDBA 権限を持っている必要があります。

```
RMAN TARGET SCOTT/TIGER@NODE1 RCVCAT RMAN/RMAN@RCAT
RUN {
  SET UNTIL LOGSEQ 124 THREAD 1;
  ALLOCATE CHANNEL T1 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL T2 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL T3 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  ALLOCATE CHANNEL T4 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  ALLOCATE CHANNEL D1 TYPE DISK;
  RESTORE CONTROLFILE;
  ALTER DATABASE MOUNT;
  CATALOG ARCHIVELOG '/ORACLE/DB_FILES/NODE1/ARCH/ARCH_1_123.RDO';
  CATALOG ARCHIVELOG '/ORACLE/DB_FILES/NODE1/ARCH/ARCH_1_124.RDO';
  RESTORE DATABASE;
  RECOVER DATABASE;
```

```
SQL 'ALTER DATABASE OPEN RESETLOGS';
}
```

オペレーティング・システム・ユーティリティを使用した災害時回復

この回復を実行するには、次の手順に従ってください。

1. 『Oracle8*i* バックアップおよびリカバリ・ガイド』で説明しているとおり、最新の全体バックアップをかわりのサイトで復元します。
2. Server Manager を起動します。
3. SYSDBA で接続します。
4. STARTUP MOUNT 文でデータベースを起動し、マウントします。
5. 適切な UNTIL オプションを指定した RECOVER コマンドを使用して、不完全回復を開始します。

   コマンドの例を示します。

   ```
   RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL
   ```

6. 特定のスレッドに対して REDO ログ・ファイル名が提示された場合は、そのファイル名を使用してください。

   提示されたアーカイブ・ログがアーカイブ・ディレクトリにない場合、そのファイルがどこにあるのかを指定します。スレッドに REDO 情報が必要で、ファイル名が指示されていない場合は、当該のスレッドのアーカイブ・ログ・ファイルを使用してみてください。
7. すべてのアーカイブ・ログ・ファイルが適用されるまで、手順 6 を繰り返します。
8. CANCEL コマンドを使用した回復操作の停止
9. ALTER DATABASE OPEN RESETLOGS 文を実行します。

注意： 分散データベース・アクションを使用する場合は、調整された分散データベース回復が回復手順で必要かどうかを確認してください。そうしないと、分散データが論理的に壊れる可能性があります。

パラレル回復

パラレル回復機能の目的は、計算と I/O の並列性を使用して、クラッシュ回復、単一インスタンス回復またはメディア回復の実行に必要な経過時間を短縮することです。パラレル回復は、複数のディスク上の複数のデータ・ファイルが同時に回復されているときの回復時間の削減に最も有効です。

Recovery Manager を使用したパラレル回復

Recovery Manager の RESTORE コマンドおよび RECOVER コマンドを使用すると、Oracle で回復の 3 つの段階すべてを自動的にパラレル化できます。

データ・ファイルの復元 : データ・ファイルを復元する場合、Recovery Manager 回復スクリプトで割り当てたチャネルの数によって、Recovery Manager が使用する並列性が実質的に設定されます。たとえば、5 つのチャネルを割り当てた場合、最大 5 つのパラレル・ストリームでデータ・ファイルを復元できます。

増分バックアップの適用 : 同様に、増分バックアップを適用する場合は、割り当てるチャネルの数によって潜在的な並列性が決まります。

REDO ログの適用 : Oracle は、RECOVERY_PARALLELISM パラメータの指定に従って REDO ログをパラレルで適用します。

RECOVERY_PARALLELISM 初期化パラメータは、インスタンス回復またはメディア回復に関係する REDO アプリケーション・サーバー・プロセスの数を指定します。1 つのプロセスがログ・ファイルを順次読み込んで、REDO 情報を複数の回復プロセスに送ります。これらの回復プロセスでは、ログ・ファイルの変更がデータ・ファイルに適用されます。値が 0 または 1 の場合は、回復が 1 つのプロセスにより直列に実行されることを示します。このパラメータの値は、PARALLEL_MAX_SERVERS パラメータの値を超えることはできません。

オペレーティング・システム・ユーティリティを使用したパラレル回復

インスタンス回復およびメディア回復を並行して実行するには、次の 2 つの方法があります。

- RECOVERY_PARALLELISM パラメータの設定
- RECOVER コマンド・オプションの指定

Oracle Server は、1 つのプロセスを使用して、ログ・ファイルを連続して読み込み、REDO 情報を複数の回復プロセスに送り出して、ログ・ファイルからの変更をデータ・ファイルに適用することができます。回復プロセスは Oracle によって自動的に起動されるので、回復を行うために複数のセッションを使用する必要はありません。

RECOVERY_PARALLELISM パラメータの設定

RECOVERY_PARALLELISM 初期化パラメータは、インスタンス回復またはメディア回復に関係する REDO アプリケーション・サーバー・プロセスの数を指定します。1 つのプロセス

がログ・ファイルを順次読み込んで、REDO 情報を複数の回復プロセスに送ります。これらの回復プロセスでは、ログ・ファイルの変更がデータファイルに適用されます。値が 0 または 1 の場合は、回復が 1 つのプロセスにより直列に実行されることを示します。このパラメータの値は、PARALLEL_MAX_SERVERS パラメータの値を超えることはできません。

RECOVER コマンド・オプションの指定

RECOVER コマンドを使用してインスタンス回復とメディア回復をパラレル化する場合、インスタンスへの回復プロセスの割当ては、オペレーティング・システムにより異なります。PARALLEL 句の DEGREE キーワードは、パラレル・サーバーの各インスタンスでのプロセス数、または全インスタンスにわたるプロセス数を指定できます。

関連項目：『Oracle8*i* 概要』（ファースト・スタート・パラレル・ロールバックについて）、システム固有の Oracle マニュアル（回復プロセスのインスタンスへの割当てについて）

OPS でのファースト・スタート・パラレル・ロールバック

INIT.ORA パラメータの FAST_START_PARALLEL_ROLLBACK を LOW または HIGH に設定すると、ファースト・スタート・パラレル・ロールバックが使用可能になります。このパラメータは、ファースト・スタート・パラレル・ロールバックに関係するサーバー・プロセスの最大数を決定するのに役立ちます。値が FALSE になっていると、ファースト・スタート・パラレル・ロールバックは使用できません。

FAST_START_PARALLEL_ROLLBACK の値が LOW の場合、ファースト・スタート・ロールバックに使用するプロセスの数は CPU_COUNT の値の 2 倍になります。値が HIGH の場合、ファースト・スタート・パラレル・ロールバックに使用するロールバック・サーバーの最大数は、CPU_COUNT の値の 4 倍です。

OPS では、複数のパラレル回復プロセスが、それらのプロセスを生成したインスタンスによって所有され、そのインスタンス内でのみ操作されます。
FAST_START_PARALLEL_ROLLBACK の正確な設定を判断するには、
V$FAST_START_SERVERS と V$FAST_START_TRANSACTIONS の内容を調べてください。

ファースト・スタート・パラレル・ロールバックでは、インスタンス間ロールバックは実行されません。しかし、複数のインスタンスがある単一データベースのロールバック・セグメント処理は改善されます。理由は、各インスタンスが、そのインスタンス専用の回復プロセス・グループを作成できるからです。

Managed Standby およびスタンバイ・データベース

スタンバイ・データベースを使用することで、OPS システムを障害から守ることができます。スタンバイ・データベースの管理を簡略化するには、Managed Standby 機能の使用を検討してください。Managed Standby データベース機能の詳細は、『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照してください。

23

単一インスタンスからパラレル・サーバーへの移行

この章では、データベースの変換、つまり Oracle Parallel Server（OPS）のオプションを使用して単一インスタンスの Oracle データベースから複数インスタンスの Oracle データベースに変換する方法について説明します。

この章では、次の項目について説明します。

- 概要
- 変換の決定
- 変換の準備
- 単一インスタンスから複数インスタンスへのデータベースの変換
- 変換のトラブルシューティング

概要

この章では、使用しているデータベースの構造で複数インスタンスをサポートする方法について説明します。また、単一インスタンスの Oracle データベースからプロジェクトを開始して、複数インスタンスのパラレル・サーバーに移行する方法についても説明します。さらに、既存の OPS 構成を拡張してノードを追加する方法も説明します。

注意： この章で複数インスタンスのデータベースに変換する前に、『Oracle8*i* 移行ガイド』を参照して必要な Oracle Server のアップグレードを行ってください。このマニュアルでは、レプリケート・システムでのアップグレードおよびダウングレードについても説明しています。

変換の決定

この項では、次の項目について説明します。

- 変換する理由
- 変換しない理由

変換する理由

次の理由により、複数インスタンス・データベースに変換する場合があります。

- 単一ノードからクラスタへ移行する（OPS を念頭に置いてアプリケーションを設計した場合）。

アプリケーションがすでに OPS 用に設計されていても、現在のデータベース設計を利用するにはより多くのインスタンスが必要な場合。十分なノードがあっても、オフラインのノードをオンラインにする必要がある。すでに OPS を使用している場合でも、ノードを追加する必要が生じることがあります。

変換しない理由

次のような状況では、OPS に変換しないでください。

- 共有されていないファイル・システムを使用している。
- アプリケーションがパラレル処理用に設計されていない。（使用しているアプリケーションについて詳しく調べる必要があります。）
- サポートされた構成（共有ディスクなど）を使用していない。

変換の準備

この項では、次の項目について説明します。

- ハードウェア要件とソフトウェア要件
- 単一インスタンスから複数インスタンスへのアプリケーションの変換
- 管理上の問題

ハードウェア要件とソフトウェア要件

OPS に変換するための要件は、次のとおりです。

- サポートされているハードウェアおよび OS ソフトウェア構成
- Oracle Parallel Server のライセンス
- すべてのノードで稼働する Oracle Server
- 使用しているシステムに OPS がリンクされていること

単一インスタンスから複数インスタンスへのアプリケーションの変換

データベースをパラレルで実行しても、OPS を効果的に実装したことにはなりません。既存のデータベースを単一インスタンスの Oracle から複数インスタンスの Oracle に移行するだけでなく、単一インスタンスの Oracle 用に設計された既存のアプリケーションを移行することも必要です。アプリケーションを複数インスタンス・データベースで使用できるようにするには、アプリケーションの分割や物理スキーマの変更も必要になります。

関連項目： 第 12 章「アプリケーションの分析」（詳細について）

管理上の問題

変換を行うときには、次の管理上の問題に注意してください。

- 単一インスタンスの Oracle から Oracle Parallel Server に変換する前に、適切なバックアップ手順を実行してください。
- OPS 環境では、アーカイブに関して追加の考慮事項があります。特に、アーカイブ・ファイル・フォーマットにはスレッド番号が必要です。さらに、メディア回復には、全ノードのアーカイブ済みログが必要です。ファイルにアーカイブする場合、ファイル・システムを共有できないシステムでは、アーカイブ・ログにアクセスするためになんらかの手段が必要です。

関連項目： 第 21 章「データベースのバックアップ」

単一インスタンスから複数インスタンスへのデータベースの変換

既存のデータベースを単一インスタンスから複数インスタンスに変換する方法を次の手順に示します。アプリケーションも単一インスタンスから複数インスタンスに変換する必要があることに注意してください。

1. アプリケーションを OPS で使用できるように修正します。
2. 必要なファイルがすべてノード間で共有されていることを確認してください。

   OPS は、複数のインスタンス間でディスクが共有されていて、各インスタンスがすべてのログ・ファイル、制御ファイルおよびデータベース・ファイルにアクセスできると想定します。ディスクはほとんどのクラスタでロー・デバイスを通して共有されるため、これらのファイルは通常、ロー・デバイス上にあります。

   注意： OPS のファイルを共有するために NFS は使用できません。NFS は十分な可用性を備えていません。ノードが停止すると NFS も停止し、ファイルにアクセスできなくなります。また、NFS は一貫性が不十分です。書込みはキャッシュに置かれ、すぐにディスクに書き込まれないことがあります。

3. 単一インスタンスの MAXINSTANCES をチェックしてください。

   MAXINSTANCES パラメータはデータベース作成時に、通常はデフォルト値 1 に設定されます。MAXINSTANCES が 1 に設定されていると、データベース上で稼働できるインスタンスは 1 つだけで、このデータベースはパラレル・サーバー・モードでは稼働できません。V$THREAD 内の行の数は、作成されるスレッドあたり 1 です。MAXINSTANCES 値はそれよりもっと大きくできます。V$ACTIVE_INSTANCES をチェックしてこの値を確認できます。

   MAXINSTANCES の値をチェックするには、V$ACTIVE_INSTANCES を問い合せます。また、次の SQL 文を入力して、制御ファイルのダンプをトレース・ファイルに出力できます。

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

   トレース・ファイルは、次のようになっています。

```
Dump file /mf1/qjones/qj1/rdbms/log/ora_20016.trc
Oracle8 Server Release 8.0.3
With the distributed, replication, parallel query and
   Parallel Server options
PL/SQL Release 3.0
ORACLE_HOME = /mf1/qjones/qj1
ORACLE_SID = mf1qj1
Oracle process number: 19        Unix process id: 20016
System name:    mf1seq
```

```
Node name:      mf1seq
Release:        3.2.0
Version:        V2.1.1
Machine:        i386
Wed Feb 22 14:30:22 1997
Wed Feb 22 14:30:23 1997
*** SESSION ID:(18.1)
# The following commands will create a new control file and
# use it to open the database.
# No data other than log history will be lost. Additional logs
# may be required for media recovery of offline data files.
# Use this only if the current version of all online logs are
# available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "TPCC" NORESETLOGS
NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 2
    MAXDATAFILES 62
     MAXINSTANCES 1
    MAXLOGHISTORY 100
LOGFILE
  GROUP 1 '/dev/rvol/v-qj80W-log11'  SIZE 200M,
  GROUP 2 '/dev/rvol/v-qj80W-log12'  SIZE 200M
DATAFILE
  '/dev/rvol/v-qj80W-sys',
  '/dev/rvol/v-qj80W-temp',
  '/dev/rvol/v-qj80W-cust1',
.
.
.
;
# Recovery is required if any of the datafiles are restored
# backups, or if the last shutdown was not normal or
# immediate.
RECOVER DATABASE
# Database can now be opened normally.
ALTER DATABASE OPEN;
```

4. 制御ファイル・スクリプトを編集して MAXINSTANCES 値を大きくします。

CREATE CONTROLFILE 文の生成に必要な SQL コマンドだけがトレース・ファイルに含まれるように、トレース・ファイルを編集します。次の変更を行います。

- 正しい初期化ファイルを指すように PFILE を設定します。
- サポートする Oracle インスタンスの数と同じになるように MAXINSTANCES パラメータの値を大きくします。

- MAXLOGHISTORY パラメータに大きめの値を使用します。

その結果得られる制御ファイルは、必要に応じてデータベースを回復し、回復したデータベースを再オープンするスクリプトです。

この SQL ファイルを実行する前に、現行制御ファイルがバックアップ・ディレクトリに入っていることを確認してください。

スクリプトの例を次に示します。

```
STARTUP NOMOUNT PFILE=$HOME/perf/tkvc/admin/tkvcrun.ora
CREATE CONTROLFILE REUSE DATABASE "TPCC" NORESETLOGS
NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 2
    MAXDATAFILES 62
    MAXINSTANCES 1
    MAXLOGHISTORY 100
LOGFILE
  GROUP 1 '/dev/rvol/v-qj80W-log11'  SIZE 200M,
  GROUP 2 '/dev/rvol/v-qj80W-log12'  SIZE 200M
DATAFILE
  '/dev/rvol/v-qj80W-sys',
  '/dev/rvol/v-qj80W-temp',
  '/dev/rvol/v-qj80W-cust1',
.
.
.
;
# Recovery is required if any of the datafiles are restored
# backups, or if the last shutdown was not normal or
# immediate.
RECOVER DATABASE
# Database can now be opened normally.
ALTER DATABASE OPEN;
```

5. 変換後すぐに新しい制御ファイルをバックアップしてください。また、データベースのバックアップ手順を実行することをお薦めします。

6. 初期化パラメータ・ファイルの管理方法を決めます。

   各インスタンスには、プライベート初期化パラメータがあります。ただし、このパラメータの一部は各インスタンスで同じ値である必要があります。これを管理するには、2 通りの方法があります。

   1 つの方法は、各インスタンスが、インスタンス間で共有される共通パラメータ・ファイルを含むプライベート・パラメータ・ファイルを持つことです。この共通パラメータ・ファイルは、全ノードがアクセスできる共有デバイス上にある必要があります。こ

の方法では、共通初期化パラメータの 1 つに汎用的な変更を加える必要があるときは、全ノードではなく 1 つのノードにその変更を加えるだけで十分です。

または、このパラメータ・ファイルのコピーを複数とり、その 1 つを、OPS 環境の各ノードのプライベート・ディスク上に置くことができます。この場合、汎用的な変更を加えるたびに、すべてのパラメータ・ファイルを更新する必要があります。

7. インスタンス固有の初期化パラメータ・ファイルで次のパラメータを編集します。

   a. このインスタンスの INSTANCE_NUMBER を指定します。各インスタンスは、起動時に番号が付けられます。インスタンス番号は空きリスト・グループの割当て時に使用されます。INSTANCE_NUMBER を指定しない場合、Oracle は起動順序に基づいて番号を割り当てます。

   b. ROLLBACK_SEGMENTS を指定します。各インスタンスには、プライベート・ロールバック・セグメントの集合があります。

   c. 初期化パラメータ・ファイルの THREAD パラメータを指定すると、インスタンスはいつも同じ REDO ログファイルの集合を使用して起動します。起動時にスレッド番号が割り当てられ、インスタンスはその番号のスレッドのログ・ファイルに関連付けられます。デフォルトではこの値は 0 です。最初のインスタンスには、この値に 1 を設定できます。

   d. DB_NAME パラメータを初期化パラメータに追加します。

8. 次の共通初期化パラメータが全インスタンスに対して同一値をとることを確認してください。

   CONTROL_FILES
   DB_BLOCK_SIZE
   DB_FILES
   DB_NAME
   GC_FILES_TO_LOCKS
   GC_ROLLBACK_LOCKS
   LM_LOCKS（同一値をお薦めします。）
   LM_PROCS（同一値をお薦めします。）
   LM_RESS（同一値をお薦めします。）
   MAX_COMMIT_PROPAGATION_DELAY
   ROW_LOCKING
   SINGLE_PROCESS

9. Oracle 実行可能プログラムが OPS のオプションとリンクされていることと、各ノードが同じバージョンの実行可能プログラムを実行中であることを確認してください。接続時の画面の見出しには "Parallel Server" という文字が表示されます。

注意： 1 つのノードが共有モードでデータベースをオープンし、他のノードが排他モードでオープンすると、データベースが壊れることがあります。

---

10. データベースの通常の停止を行います。
11. オペレーティング・システム・コマンドを使用して制御ファイルをバックアップします。
12. バックアップを残したうえで、制御ファイルを削除します。
13. 新規作成したスクリプトを実行します。このスクリプトは、一部のデータベース・オブジェクトの構造を大きくした新しいデータを使用して、古い制御ファイルを再作成します。
14. ロールバック・セグメントを追加します。
15. スレッドを追加します。
16. データベースを停止します。
17. データベースを共有モードで起動します。最初のインスタンスが起動されます。
18. 18-14 ページの「共有モードでの起動」で示している基本的な手順を使用して、2 番目のインスタンスを共有モードで追加します。（2 番目のインスタンスは、最初のインスタンスが共有モードである場合にのみ成功します。）REDO ログ・ファイル、ロールバック・セグメントなどを追加します。
19. パフォーマンスの最適化のために、GC_* パラメータおよび LM_* パラメータをチューニングします。

変換のトラブルシューティング

この項では、一般的なエラーの解決方法について説明します。

- 変換後のデータベース回復
- ロールバック・セグメント表領域の損失
- パラメータ・ファイルの不適切な NFS マウント

変換後のデータベース回復

単一インスタンスの Oracle から OPS への変換後、消失したデータベース・ファイルまたは Oracle8 ファイルがある場合には、コールド・バックアップを復元し、REDO ログの変更をすべて適用してください。この場合、古い制御ファイルが使用され、変換を行わなかったのと同じになります。OPS に移行する場合は、新しい制御ファイルを再作成する必要があります。

ロールバック・セグメント表領域の損失

プライベート・ロールバック・セグメント用の表領域を作成し、作成した表領域を起動時に特定のインスタンスに割り当てた場合、次の問題が起きる可能性があります。また、ロールバック・セグメントを含むファイルが失われた場合にも、この問題が起きることがあります。

1 つのロールバック・セグメント表領域、またはロールバック・セグメントを含むファイルがメディア障害によって失われた場合、すべてのインスタンスに障害が発生します。回復するには、全インスタンスをシャットダウンします。回復対象のロールバック・セグメントをオフラインにするために、その他すべてのロールバック・セグメントをオフラインの状態にしておく必要があります。

パラメータ・ファイルの不適切な NFS マウント

前述したように、NFS で共通パラメータ・ファイル（または任意の Oracle ファイルあるいは実行可能ファイル）にアクセスするのは望ましくありません。NFS ディスクが停止した場合、他のインスタンスはいずれも起動できません。制御ファイルおよびデータ・ファイルへのアクセスは NFS ではサポートされていません。

第Ｖ部

参照情報

A

以前のバージョンとの相違点

この付録では、Oracle Parallel Server Option のリリースによる違いを説明します。

- 8.0.4 と 8.1 の相違点
- リリース 8.0.3 とリリース 8.0.4 の相違点
- リリース 7.3 とリリース 8.0.3 の相違点
- リリース 7.2 とリリース 7.3 の相違点
- リリース 7.1 とリリース 7.2 の相違点
- リリース 7.0 とリリース 7.1 の相違点
- バージョン 6 とリリース 7.0 の相違点

関連項目：『Oracle8*i* 移行ガイド』（データベースのアップグレードについて）

8.0.4 と 8.1 の相違点

キャッシュ・フュージョンによるアーキテクチャの変更

あるインスタンスが保持しているブロックに対して別のインスタンスが一貫性読込み（CR）を要求した場合、キャッシュ・フュージョン処理によって、要求されたブロックの CR コピーがインターコネクトを介して要求元のインスタンスに直接送信されます。これにより、read / write 競合時の複数インスタンス間でのキャッシュ一貫性矛盾が大幅に減少します。

キャッシュ・フュージョンのインプリメンテーションでは、特定のバックグラウンド・プロセスとフォアグラウンド・プロセス（LMON と LCK）が、インターコネクトを介してインスタンス間で直接通信することが要求されます。新しいプロセスであるブロック・サーバー・プロセス（BSP）は、コミットされていないトランザクションをロールバックし、要求元インスタンスへ送信するために CR サーバー・ブロックをコピーします。これにより

キャッシュの一貫性を維持するために必要な ping が減少し、パフォーマンスが大幅に向上します。

キャッシュ・フュージョンによって、OLTP アプリケーションおよびハイブリッド・アプリケーションで OPS を開発しやすくなります。これまで、ランダムな変更が発生するデータベースはパラレル・サーバーに向いているとはみなされませんでした。キャッシュ・フュージョンと高度なインスタンス間インターコネクト・テクノロジが出現したことで、OLTP アプリケーションおよびハイブリッド・アプリケーションの拡張性が向上します。たとえば、あるインスタンスが変更した表を別のインスタンスが読み取る場合などは特に拡張性が向上します。

新しいビュー

V$DLM_ALL_LOCKS ビューが新規に追加されました。このビューは、ブロックする側のロックか、ブロックされる側のロックかにかかわらず、またその他すべてのロック・タイプのロックに関する統計情報を示します。

V$DLM_RESS ビューが新規に追加されました。このビューは、ロック・タイプに応じて、ロックに対応付けられたすべてのリソースを示します。

V$DLM_CONVERT_LOCAL ビューが新規に追加されました。このビューは、ローカル・ノードでオープンされたロックのロック変換統計情報を示します。

V$DLM_CONVERT_REMOTE ビューが新規に追加されました。このビューは、リモート・ノードでオープンされたロックのロック変換統計情報を示します。

V$DLM_MISC ビューが新規に追加されました。このビューは、DLM メッセージ情報を示します。

GMS の削除

8.1 では、GMS（グループ・メンバーシップ・サービス）の機能が GMS モジュールからベンダー固有の Cluster Manager（CM）と Oracle データベース・カーネルに移されています。8.1 では、Oracle ユーザーが個別の GMS モジュールを見ることはできません。

この変更により、ベンダー・ハードウェアの Oracle との互換性が大きく向上します。また、ユーザーにとっては、CM の使用方法とメンテナンスが簡単になります。CM はインスタンスの起動時に自動的に起動します。メンバー・サービスの起動と停止を手動で行う必要がなくなりました。

パラレル・トランザクション回復から " ファースト・スタート・パラレル・ロールバック " へ

" パラレル・トランザクション回復 " 機能の名前が、" ファースト・スタート・パラレル・ロールバック " に変更されました。名前の変更だけでなく、8.0 ではロールバック・セグメントの回復を SMON によってシリアルに処理していました。この方法ではロールバック回復時間が長くなります。8.1 ではファースト・スタート・パラレル・ロールバックにより回

復時間が短縮されており、それだけ早くデータベースが使用可能になります。パラメータ FAST_START_PARALLEL_ROLLBACK（以前の名前は PARALLEL_TRANSACTION_RECOVERY）の値が 1 より大きい場合、パラレル・ロールバックは複数のプロセスを使用してロールバック・セグメントを回復します。

このパラメータのデフォルトは LOW であり、パラレル回復の実行で使用する SMON 以外のプロセスの数は、CPU_COUNT 数の 2 倍を超えないことを示します。

より正確な設定値を決定するには、新しい 2 つの表 V$FAST_START_SERVERS と V$FAST_START_TRANSACTIONS の内容を調べてください。また、トランザクションの回復に必要な平均時間と希望回復時間も考慮してください。FAST_START_PARALLEL_ROLLBACK を 1 より大きい値に設定すると、SMON は複数の回復プロセスを起動して、ロールバック・ファイル内の未回復のロールバック・セグメントの量に対応します。SMON が起動するプロセスの数は、FAST_START_PARALLEL_ROLLBACK の値によって制限されています。

インスタンス登録の変更

サービスを識別するために以前使用していた単一名（SID）が、3 つのレベルの指定に変わりました。インスタンス登録のための新しいパラメータは、次のとおりです。

SERVICE_NAME	サービスの最上位ビューの名前であり、TNSNAMES.ORA で指定します。複数のインスタンスまたはノードにまたがってかまいません。
SERVICE_NAMES	サービスのインスタンス名であり、複数のノードにまたがることができます。このパラメータは、INIT.ORA で指定します。
INSTANCE_NAME	サービスの中間層の名前です。インスタンスの ORACLE_SID に対応します。

クライアントは、必要なハンドラやインスタンスを指定しなくてもサービスに接続できます。したがって、最適なインスタンスの最適なハンドラを選択する自動ロード・バランシングが可能です。ロード・バランシングについては、次の項で説明します。

リスナーのロード・バランシング

TNS リスナーは、複数のノードにまたがる分散サービスを介してロード・バランシングを実行します。サービス、インスタンス、およびハンドラの名前を使用してロード・バランシングの動作を決定します。

1. クライアント・プログラムが、接続するサービスの名前を指定します。
2. リスナーが、サービス内で最少ロード回数のインスタンスを検索します。
3. リスナーが、インスタンス内で最少ロード回数のハンドラを検索します。
4. リスナーが、クライアントを最適なハンドラにリダイレクトします。

診断プログラムの拡張

Oradebug は、コンサルティングおよびサポート要員が問題のあるシステムにランタイムで診断とトラブルシューティングを行うときに使用するユーティリティです。Oradebug 機能は、Oracle Parallel Server 用に拡張されています。

Oracle Parallel Server Management（OPSM）

OPSM は、パラレル・サーバーの管理を簡単にするオプションです。OPSM 8.1 の拡張機能は、任意のプラットフォームでパラレル・サーバーを管理するための単一の汎用インタフェースを備えています。

OPSM の詳細は、『Oracle Parallel Server Management ユーザーズ・ガイド』を参照してください。

パラレル・サーバーのインストールとデータベース構成

Oracle Universal Installer と Oracle Database Configuration Assistant は、どちらもクラスタを認識します。リリース 8.1 では、単一の Installer セッションだけで OPS をインストールできます。Installer は、ユーザーからノード情報を収集し、必要な Oracle 製品を指定されたノードに配布し、OPS Assistant を起動してインスタンスを設定し、データベースを作成します。

OPS Assistant がこのプロセスを終了すると、すべてのノードでパラレル・サーバーが使用可能になり、パラレル・サーバーの構成情報が保存されます。これにより、OPSM はこの情報を使用して新しいパラレル・サーバーを管理できます。

ジョブとインスタンスの親和性

ジョブとインスタンスの親和性とは、インスタンスに対するジョブの対応付けです。新しい DBMS_JOB パッケージを使用すると、特定のインスタンスまたは任意のインスタンスが、OPS 環境でユーザーが発行したジョブを実行できるかどうかを指定できます。

リリース 8.1 のこの機能を使用して、ロード・バランシングを改善し、ブロックの ping を制限できます。たとえば、クラスタ化された環境にあるすべてのインスタンスが遅延トランザクション・キューからトランザクションを伝播しようとする場合、OPS とレプリケーションを同時に使用すると、遅延トランザクション・キューでブロックの ping の問題が発生することがあります。表に対するアクティビティをパラレル・サーバーのクラスタ内にある 1 つのインスタンスだけに制限することで、ping を制限できます。詳細は、『Oracle8*i* パッケージ・プロシージャ リファレンス』を参照してください。

廃止されたパラメータ

次のパラメータは、リリース 8.1 で廃止されました。

GC_LCK_PROCS

GC_LATCHES

PARALLEL_DEFAULT_MAX_INSTANCES

LOG_FILES

OPS_ADMIN_GROUP

CACHE_SIZE_THRESHOLD

OGMS_HOME

ALLOW_PARTIAL_SN_RESULTS

SEQUENCE_CACHE_ENTRIES

リリース 8.0.3 とリリース 8.0.4 の相違点

新しい初期化パラメータ

次の初期化パラメータが、Oracle Parallel Server に追加されました。

OGMS_HOME
GC_LATCHES
PARALLEL_SERVER

廃止された初期化パラメータ

次の初期化パラメータは廃止されました。

MTS_LISTENER_ADDRESS
MTS_MULTIPLE_LISTENERS

廃止された起動パラメータ

PARALLEL
EXCLUSIVE

動的パフォーマンス・ビュー

次のビューが変更になりました。

V$DLM_LOCKS

グループ・メンバーシップ・サービス

OGMSCTL コマンドに新しいオプションが追加されました。

リリース 7.3 とリリース 8.0.3 の相違点

新しい初期化パラメータ

次のパラメータが、Oracle Parallel Server に追加されました。

FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY
LM_LOCKS
LM_PROCS
LM_RESS
INSTANCE_GROUPS
PARALLEL_INSTANCE_GROUP
OPS_ADMIN_GROUP
ALLOW_PARTIAL_SN_RESULTS

関連項目： 18-11 ページの「必要なロックの量の決定および LM_* パラメータの設定」

廃止された GC_* パラメータ

次のグローバル・キャッシュ・ロック初期化パラメータは廃止されました。

GC_DB_LOCKS パラメータ
GC_FREELIST_GROUPS パラメータ
GC_ROLLBACK_SEGMENTS パラメータ
GC_SAVE_ROLLBACK_LOCKS パラメータ
GC_SEGMENTS パラメータ
GC_TABLESPACES パラメータ

関連項目： 18-8 ページの「GC_* グローバル・キャッシュ・パラメータ」

変更された GC_* パラメータ

GC_* パラメータで設定した値は、素数に調整されるのではなく、入力したとおりの値になります。

次のパラメータが変更されました。

GC_FILES_TO_LOCKS
GC_ROLLBACK_LOCKS
GC_RELEASABLE_LOCKS

関連項目： 9-12 ページの「GC_* 初期化パラメータ」

動的パフォーマンス・ビュー

次のビューが新規に追加されました。

V$RESOURCE_LIMIT
V$DLM_CONVERT_LOCAL
V$DLM_CONVERT_REMOTE
V$DLM_LATCH
V$DLM_MISC
V$FILE_PING
V$CLASS_PING

次のビューが変更になりました。

V$BH
V$SESSIONS
V$SYSSTAT

グローバル動的パフォーマンス・ビュー

V$ROLLNAME を除く個々の V$ ビューに対応したグローバル動的パフォーマンス・ビュー（GV$ 固定ビュー）が追加されました。

関連項目： 20-10 ページの「グローバル動的パフォーマンス・ビュー」

統合分散ロック・マネージャ

Oracle Parallel Server リリース 8.0 は、外部分散ロック・マネージャに依存しません。ロック管理機能は、Oracle の内部機能になりました。統合分散ロック・マネージャは、外部ノード・モニターに依存します。

LMON プロセスおよび LMD*n* プロセスが追加されました。

関連項目： 第 8 章「統合分散ロック・マネージャ」

インスタンス・グループ

関連するすべてのインスタンスをまとめて操作するためにインスタンスを論理的にグループ化する機能が追加されました。

関連項目： 18-21 ページの「インスタンス・グループの指定方法」

グループ・メンバーシップ・サービス

グループ・メンバーシップ・サービス（GMS）は、インスタンス間の初期化および調整を行うために、ロック・マネージャ（LM）および Oracle の他のコンポーネントによって使用されます。

関連項目： 18-20 ページの「Cluster Manager」

ファイン・グレイン・ロック

Oracle Parallel Server リリース 8.0 では、すべてのプラットフォームでファイン・グレイン・ロックが利用できます。ファイン・グレイン・ロックはデフォルトで使用可能になっています。

クライアント側のアプリケーション・フェイルオーバー

Oracle8 では、データベースとの接続が途切れた場合に、アプリケーションが自動的に再接続できる機能をサポートしています。

関連項目： 22-2 ページの「インスタンス障害の回復」

Recovery Manager

Recovery Manager が、メディア障害の回復手段として推奨される手段となりました。

関連項目： 22-6 ページの「メディア障害の回復」

リリース 7.2 とリリース 7.3 の相違点

初期化パラメータ

次の初期化パラメータが、Parallel Server Option に追加されました。

CLEANUP_ROLLBACK_ENTRIES
DELAYED_LOGGING_BLOCK_CLEANOUTS
GC_FREELIST_GROUPS
GC_RELEASABLE_LOCKS

データ・ディクショナリ・ビュー

次のビューが、Parallel Server Option に追加されました。

FILE_LOCK

動的パフォーマンス・ビュー

次のビューが変更されました。

V$BH

次のビューが追加されました。

V$SORT_SEGMENT
V$ACTIVE_INSTANCES

空きリスト・グループ

表およびクラスタだけでなく索引にも空きリスト・グループが設定できるようになりました。

ファイン・グレイン・ロック

Oracle Parallel Server リリース 7.3 の PCM ロックにはファイン・グレイン・ロックを使用して構成するためのオプションが追加されています。この変更は、分散パラレル・サーバー・キャッシュ内でデータベース・ブロックをプロテクトするためのロックを決定する各パラメータの解釈に影響を与えます。

ファイン・グレイン・ロックは、複数のノード構成でロックを提供するより効率の高い方法です。特に、MPP システムでは、この方法により、ロック衝突率が低下し、ロック管理に必要な領域を低減します。この機能は、ハードウェアおよびオペレーティング・システムのプラットフォームの機能に依存しているため、すべてのプラットフォームで使用できるわけではありません。

ファイン・グレイン・ロックについては、9-14 ページの「PCM ロックの 2 つの方法 : 固定および解放可能なロック」で説明しています。

インスタンス登録

この機能により、各インスタンスは、インスタンスおよび属性のある部分を登録することができ、また他のインスタンスと接点を確立できます。インスタンス登録は、パラレル・サーバーのリモート・インスタンス上でパラレル問合せ障害が発生した場合を除いて、ユーザーからは透過的になっています。パラレル問合せがリモート・インスタンス上のエラーにより停止した場合、その障害インスタンスはエラー・メッセージの中で識別できるようになりました。

ソートの向上

このリリースでは、ソート一時領域の割当て方法が以前より効率良く行われるため、これにより直列化やインスタンス間の ping が減少します。この機能を正しくセットアップすることにより、特にパラレル・モードにおける OPS パフォーマンスを向上させることができます。

最善の結果を得るためには、安定したソート領域を確立してください。ソート領域はインスタンス内にキャッシュされることに注意してください。あるインスタンスが領域を解放するのは、領域が不足している他のインスタンスが、領域を解放するように最初のインスタンスに呼出しを発行する場合です。これは多くの資源を使用する直列化されたプロセスであり、パフォーマンスを損なうことになります。使用しているシステムのソート領域が恒久的に不足している場合、領域を多めに割り当てるか、単に一時表領域を使用しない方がよいでしょう。

ソート領域の安定度を知るには、V$SORT_SEGMENT ビューをチェックします。この新しいビューは、すべてのインスタンスのソート履歴を示すものです。FREED_EXTENTS 列および ADDED_EXTENTS 列が、割当て / 割当て解除アクティビティが過剰であることを示している場合は、該当する表領域に領域の追加を検討する必要があります。また、ソート領域全体でインスタンス間の競合があるかどうかを判別するためには、FREE_REQUESTS の値をチェックします。

割当ておよび割当て解除が過剰になるもう 1 つの理由として、ソート操作の規模が大きすぎるという場合もあります。膨大なソートを必要とする処理に対して異なる一時表領域を割り当ててみるのもよいでしょう。MAX_SORT_SIZE の値は、これらの大きなソートが起こっているかどうかを判断する要因になります。

関連項目 : 『Oracle8*i* チューニング』（ソートの機能強化について）

XA パフォーマンスの向上

拡張性およびスループットの面におけるさまざまな改善は、XA トランザクションに影響を及ぼします。これらの変更を行っても、パフォーマンスが向上する以外に、目立った影響はありません。

次に示す 3 つのラッチのパフォーマンスが大きく向上することにより、拡張性が強化されています。

- グローバル・トランザクション・マッピング表ラッチ
- エンキュー・ラッチ
- セッション切替えラッチ

ほとんどの共通 XA コールでコード・パスが減り、データベースへの往復が減るため、トランザクション・スループットが強化されます。

XA 回復機能の強化

OPS では、XA インタフェースを使用して TP モニターから送られる分散トランザクションの回復が完全にサポートされています。

XA_RECOVER コールが強化されました。これにより、他のインスタンスの障害が発生したトランザクションからあるインスタンスの適切、かつ完全な回復を保護します。

XA_RECOVER コールをインスタンス回復まで待機させるオプションが追加されています。この機能により、ある Oracle インスタンスで、障害が発生した Oracle インスタンスに代わって回復を実行できます（この 2 つのインスタンスが同じ OPS クラスタに属している場合）。

XA_INFO 文字列には OPS_FAILOVER と呼ばれる新しい句があります。これが一定の XA リソース・マネージャ接続に対して TRUE に設定されると、その接続から発行される XA_RECOVER コールはいずれも、必要なインスタンス回復が終了するまで待機します。構文は、次のとおりです。

OPS_FAILOVER=T

大文字または小文字（T または t）を使用できます。OPS_FAILOVER のデフォルト値は FALSE（F または f）です。

これまでは、XA_RECOVER コールが、障害インスタンスからインダウト・トランザクションのリストを戻すことは保証されていませんでした。OPS_FAILOVER=T を設定することにより、確実にこのリストが戻るようになります。

OPS_FAILOVER を TRUE に設定すると、XA_RECOVER コールは、SMON がキャッシュ回復を終了してインダウト・トランザクションを識別し、それらをインダウト・トランザクションのリストを持つ PENDING_TRANS$ 表に追加するまで待機します。

遅延トランザクション回復

トランザクション回復動作が変更され、次のことが可能になりました。

- 起動時のデータベース可用性の向上
- トランザクションのパラレルでの回復（必要な場合）
- 短いトランザクションの回復を中断しない、長いトランザクションの回復

高速ウォーム・スタート

以前のリリースでは、障害発生後は完全にトランザクションが回復されるまでデータベースはオープンできませんでした。リリース 7.3 では、キャッシュが回復するとすぐに、データベースをオープンして接続できます。（これは、OPS 環境におけるフェイルオーバーの実行ではなく、データベースをオープンするときにだけ該当します。）インスタンス障害が発生した場合は、稼働中の他のインスタンスからデータベースに接続できます。

これは、その障害が発生した時点でアクティブであったトランザクションがまだロールバックされていないことを意味します。そのトランザクションは、システムのユーザーからはアクティブ（行ロックが保持されている）に見えます。さらに、障害発生時点でアクティブであったシステム上のトランザクションはすべて DEAD とマークされ、これらのトランザクションを含むロールバック・セグメントは、PARTIALLY AVAILABLE とマークされます。このトランザクションの回復は、SMON 回復の一部としてバックグラウンドで行われるか、またはこのトランザクションに遭遇したフォアグラウンド・プロセスによって行われます。これについては、次の項で説明します。このロールバック・セグメントは、オンライン化できます。

トランザクションの回復

高速ウォームスタート機能があるため、全トランザクションの回復に要する時間がデータベースの一般的利用を制限することはありません。そのため、ユーザーは、未回復のトランザクションがロックしている部分以外のすべてのデータを使用できます。ただし、OLTP 作業負荷があると、データベースまたはインスタンスが停止したときにアクティブであった要求はすべて、すぐに再投入されると考えられます。これらの要求は、きわめて高い確率で、未回復のトランザクションが保持するロックに遭遇します。したがって、これらのトランザクションの回復に必要な時間は、ロックされたデータにアクセスするには依然重要です。この問題の軽減を図るため、トランザクション回復は必要に応じて並列に実行できるようになっています。次の操作で回復を実行できます。

フォアグラウンド・プロセスによる回復 まだ回復されていないトランザクションによって行がロックされている場合があります。そのような行に遭遇したフォアグラウンド・プロセスは、自力でトランザクションを回復できるようになっています。現在の SMON による回復も実行されます。それによって、最終的にトランザクション全体を完全に回復できるようになっています。ただし、フォアグラウンド・プロセスが行ロックに遭遇した場合は、ロックを保持しているトランザクションを速やかに回復し、実行を継続できます。このように、回復操作は必要ベースでパラレル化されています。デッド・トランザクションはアクティブ・トランザクションの妨げにはなりません。以前のリリースでは、アクティブ・トランザ

クションは、SMON がデッド・トランザクションを回復するまで待機する必要がありました。

回復は、ロールバック・セグメントごとのベースで実行されます。これにより、異なるインスタンスにある複数のフォアグラウンド・プロセスが、同じロールバック・セグメント内のトランザクションを回復することはありません。したがって、ping は起こりません。フォアグラウンド・プロセスは、そのトランザクションを完全に回復します。これまでは、待機するしかありませんでした。また、このプロセスにより、ロールバック・セグメント全体を探索して、未回復トランザクションをすべて部分的に回復します。そして、各トランザクションに構成可能な変更（取消し記録）の数を適用します。これで、短いトランザクションを速やかに回復することができます。このとき、長いトランザクションの回復終了を待つ必要はありません。初期化パラメータ CLEANUP_ROLLBACK_ENTRIES は、適用する変更の数を指定します。

SMON による回復 SMON トランザクション回復処理はほとんど変更されていません。SMON は、そのインスタンス内で DEAD とマークされたトランザクションの回復、起動時のトランザクション回復、およびインスタンス回復を受け持っています。唯一の変更点は、SMON が全トランザクションを何度か探索し、その探索ごとに各トランザクションについて指定数の取消し記録だけを適用することです。これにより、短いトランザクションは、長いトランザクションの回復終了まで待機することがなくなります。

ロールバック・セグメントのオンライン化による回復 ロールバック・セグメントをオンライン化することで、そのロールバック・セグメントに含まれるすべてのトランザクションが完全に回復できるようになりました。これまでは、オンライン化プロセスが、回復を行うよう SMON に通知していました。ロールバック・セグメントを、ウォームスタートやインスタンス起動の一部として暗黙にオンライン化しても、全トランザクションを回復することにはならず、それらを DEAD とマークするだけであることに注意してください。

接続時のロード・バランシング

標準の Oracle では、ロード・バランシングによって、SQL*Net 接続時に複数のリスナーと複数のインスタンスの負荷バランスがとれるようになりました。これによって、複数のリスナーが 1 つの Oracle インスタンスをリスニングすることができ、Oracle ディスパッチャが複数のリスナーに登録されます。SQL*Net クライアント・レイヤーは、DESCRIPTION_LIST 機能を介して複数のリスナーをランダム化します。

接続時のロード・バランシングの詳細は、Oracle7 Server リリース 7.3 の SQL*Net マニュアルを参照してください。

ソート処理におけるキャッシュのバイパス

SORT_DIRECT_WRITES 初期化パラメータのデフォルト値は AUTO に設定されています。このパラメータは、ソート領域が一定のサイズ以上の場合は、オンになります。これによりパフォーマンスが向上します。詳細は、『Oracle8*i* チューニング』を参照してください。

遅延ロギング・ブロック・クリーンアウト

Oracle7 Server リリース 7.3 では、遅延ブロック・クリーンアウトのパフォーマンスが向上しており、関連する ping が減少します。特に、Oracle Parallel Server を使用している場合は、これらの機能強化は有用です。

Oracle7 Server リリース 7.3 では、DELAYED_LOGGING_BLOCK_CLEANOUTS という新しい初期化パラメータを提供しています。このパラメータのデフォルト値は TRUE です。

Oracle がトランザクションをコミットすると、そのトランザクションが変更した各ブロックはそのコミット時間をすぐにマークされるわけではありません。これは後で、そのブロックの読込み時または更新時に、要求に基づいて実行されます。これを、ブロック・クリーンアウトと呼びます。現行ブロックを更新中にブロック・クリーンアウトが実行されると、そのクリーンアウトは変更され、REDO レコードには、この更新レコードが付加されます。以前のリリースでは、現行ブロックへの読込み中にブロック・クリーンアウトを行う必要がある場合は、余分のクリーンアウト REDO レコードが生成され、そのブロックのクリーン度が損われました。この点が変更されています。

リリース 7.3 では、トランザクションをコミットすると、そのトランザクションによって変更されたキャッシュ内のすべてのブロックが、ただちにクリーンアウトされます。コミット時に実行されるこのクリーンアウトは、REDO ログ・レコードを生成しない " 高速バージョン " であり、そのブロックを再固定しません。このように、長時間実行しているトランザクションにより変更されたブロックを除いて、ほとんどのブロックについてクリーンアウトが実行されます。

したがって、問合せ中は、データ・ブロックのトランザクション情報は通常最新となっており、ブロック・クリーンアウトが必要とされる頻度が、大幅に減少します。トランザクションがまだアクティブになっているブロックを問い合せるとき、またコミット中にクリーンアウトされていないブロックを問い合せるときには、定期的なブロック・クリーンアウトが必要です。

変更時（INSERT、DELETE、UPDATE）には、クリーンアウト REDO ログ・レコードが生成され、そのレコードには変更の REDO が付加されます。

パラレル問合せプロセッサの親和性

Oracle7 Server リリース 7.3 は、パラレル問合せオプション用のインスタンス間でサーバーを割り当てるデフォルトの方法が改善されています。その結果、ユーザーはヒントを与えなくても並列性を指定できます。

パラレル問合せのスレーブは、ユーザー問合せ時のディスク転送速度や CPU 処理速度に基づいて割り当てられるようになっています。資源を消費するリモート・ディスクへのアクセスよりもローカル・ディスクへのアクセスの方が望ましいため、ローカル・ディスクにアクセスするスレーブに問い合せるように作業が割り当てられます。こうして、データのローカル性を高めたため、パラレル問合せのパフォーマンスが向上します。

最善の結果を得るためには、パラレル・サーバーのインスタンスとノード間でデータを均等に配分するようにしてください。これは、特に、処理を大きく左右する中型から大型の表についてあてはまります。各ディスク上、またはすべてのノードにわたってデータを均等に分散する必要があります。非常に小さな表の場合、これは必要ありません。

たとえばノードが 2 つある場合は、表の 90% を 1 つのノードに置き、残りの 10% を別のノードに置くようなアンバランスな方法で配分しないようにします。これと同様に、ディスクが 4 つある場合、1 つのディスクにデータの 90% を、残り 3 つのディスクに 10% を置くということもすべきではありません。データは、使用可能なノードおよびディスクに均等に分散させるべきです。ディスク・ストライプ化を使用するとこれを自動的に行うことができます。ディスク・ストライプ化を使用しないで、最適パフォーマンスを望む場合は、ディスクを手動でストライプ化する必要があります。

リリース 7.1 とリリース 7.2 の相違点

不要になった領域の事前割当て

パラレル・サーバーの大部分の構成では、空きリスト・グループ間のデータの分割を維持するために、データ・ブロックを事前に割り当てる必要がなくなりました。行を挿入すると、インスタンスの該当する空きリスト・グループにデータ・ブロックのグループが割り当てられます。

データ・ディクショナリ・ビュー

パラレル・サーバー・オプションには、特に次のビューが追加されました。

FILE_LOCK
FILE_PING

動的パフォーマンス・ビュー

次のビューが変更になりました。

V$BH
V$CACHE
V$PING
V$LOCK_ACTIVITY

次のビューが追加されました。

V$FALSE_PING
V$LOCKS_WITH_COLLISIONS
V$LOCK_ELEMENT

空きリスト・グループ

次のコマンドを使用して、セッションから特定のインスタンスを指定できるようになったため、空きリスト・グループも指定できるようになりました。

ALTER SESSION SET INSTANCE = *instance_number*

表ロック

次のコマンドを使用して、ユーザーが表をロックする機能を使用禁止にできるようになりました。

ALTER TABLE *table_name* DISABLE TABLE LOCK

表ロックは、次のコマンドによって再び使用可能になります。

ALTER TABLE *table_name* ENABLE TABLE LOCK

ロック・プロセス

障害が発生しているインスタンスが保持している PCM ロックは、そのインスタンスを回復する別のインスタンスのロック・プロセスで回復できるようになりました。

リリース 7.0 とリリース 7.1 の相違点

初期化パラメータ

CACHE_SIZE_THRESHOLD が新しく追加されました。

動的パフォーマンス・ビュー

次のビューが変更になりました。

V$BH
V$CACHE
V$PING
V$LOCK_ACTIVITY

バージョン 6 とリリース 7.0 の相違点

この項では、Oracle バージョン 6 とリリース 7.0 の違いを説明します。

バージョン間の互換性

Parallel Server Option のバージョン 6 は、Oracle7 と上位互換性があります。ただし、例外が 1 つあります。バージョン 6 では、すべてのインスタンスが同じ REDO ログ・ファイルの集合を共有しますが、Oracle7 では各インスタンスが固有の REDO ログ・ファイルの集合を所有します。Oracle7 への移行の詳細は、『Oracle8*i* 移行ガイド』で説明しています。Oracle7 で動作できるようにデータベースをアップグレードすると、そのデータベースは、Oracle バージョン 6 では起動できません。Oracle7 上で実行するアプリケーションは、Oracle バージョン 6 上で実行できない場合があります。

ファイル操作

Oracle バージョン 6 では、次のファイル操作はデータベースが排他モードでマウントされている場合しかサポートされていませんでした。Oracle7 では、パラレル・モードでもサポートされています。

- データ・ファイルの追加、改名または削除
- データ・ファイルのオフラインまたはオンライン
- 表領域の作成、変更または削除
- 表領域のオフライン化またはオンライン化

これらの操作を実行するインスタンスは、データベースをオープンしていても、マウントしていてもかまいません。

表 A-1 に、データベースをパラレル・モードでマウントした Oracle バージョン 6 では実行できないファイル操作とそれに対応する SQL 文を示します。

表 A-1 Oracle7 でサポートされるようになった SQL 文

操作	SQL 文
表領域の作成	CREATE TABLESPACE tablespace
表領域の削除	DROP TABLESPACE tablespace
表領域のオフライン化またはオンライン化	ALTER TABLESPACE tablespace OFFLINE ALTER TABLESPACE tablespace ONLINE
データ・ファイルの追加	ALTER TABLESPACE tablespace ADD DATAFILE
データ・ファイルの改名	ALTER TABLESPACE tablespace RENAME DATAFILE
データ・ファイル・ログ・ファイルの改名	ALTER TABLESPACE tablespace RENAME FILE
REDO ログ・ファイルの追加	ALTER DATABASE dbname ADD LOGFILE
REDO ログ・ファイルの削除	ALTER DATABASE dbname DROP LOGFILE
データ・ファイルのオフラインまたはオンライン	ALTER DATABASE dbname DATAFILE OFFLINE ALTER DATABASE dbname DATAFILE ONLINE

Oracle7 では、上に示したすべてのファイル操作が、データベースを共有モードでマウントしているときに実行できます。

アクティブな REDO ログ・ファイルは削除できません。また、削除した結果、スレッドのグループ数が 2 未満に減ってしまう場合も削除できません。Oracle7 でデータ・ファイルをオンラインまたはオフラインにする場合、インスタンスは、データベースをオープンかクローズにして、マウントしていてもかまいません。他のインスタンスがデータベースをオープンしている場合は、ファイルをオンラインまたはオフラインにするインスタンスも、そのデータベースをオープンしている必要があります。

注意： データ・ファイルの追加、表領域の作成または表領域やそのデータ・ファイルの削除を行った場合は、Oracle をパラレル・モードで再起動する前に、必要に応じて GC_FILES_TO_LOCKS および GC_DB_LOCKS の値を調整してください。この調整を行わないと、新しいファイルを対象とするロックの数が足りなくなる場合があります。

遅延ロールバック・セグメント

グローバル定数パラメータ GC_SAVE_ROLLBACK_LOCKS は、遅延ロールバック・セグメント用に分散ロックを確保します。これらのセグメントには、オフラインにされた表領域でのトランザクションのロールバック・エントリが含まれています。

バージョン 6 では、パラレル・モードで表領域のオフライン化はサポートされていないため、初期化パラメータ GC_SAVE_ROLLBACK_LOCKS は Oracle バージョン 6 では必要ありません。Oracle7 では、このパラメータは遅延ロールバック・セグメント用に必要です。

REDO ログ

Oracle バージョン 6 では、全インスタンスが、同じオンライン REDO ログ・ファイルの集合を共有し、各インスタンスは現行 REDO ログ・ファイル内でそれに割り当てられた表領域に書込みを行います。

Oracle7 では、各インスタンスはそれ自身の REDO ログ・ファイルの集合があります。REDO ログ・ファイルの集合は、REDO スレッドと呼ばれます。スレッド番号は、ファイルがデータベースに追加されたときに REDO ログ・ファイルと対応付けられ、各インスタンスは、起動時にスレッド番号を取得します。

Oracle7 では、インスタンス単位でログ・スイッチが行われます。Oracle バージョン 6 では、複数のインスタンスが REDO ログ・ファイルを共有しているため、ログ・スイッチは全インスタンスに適用されます。

Oracle7 では、オンライン REDO ログ・ファイルのミラー化を導入しています。ミラー化の度合は、インスタンス単位で決定されます。これにより、各インスタンス上で実行されるアプリケーションの要件に従ってミラー化を指定できます。

ALTER SYSTEM SWITCH LOGFILE

Oracle バージョン 6 では、全インスタンスがオンライン REDO ログ・ファイルの集合を共有していました。したがって、ALTER SYSTEM SWITCH LOGFILE 文により、全インスタンスで新しい REDO ログ・ファイルへのログ・スイッチが強制的に行われました。

Oracle7 では、この SQL 文のグローバル・オプションはありませんが、ALTER SYSTEM ARCHIVE LOG CURRENT 文を使用して、全インスタンスで強制的にログ・ファイルを切り替える（切替えが行われるまでのオンライン・ログ・ファイルをすべてアーカイブする）ことができます。

初期化パラメータ

Oracle バージョン 6 の LOG_ALLOCATION パラメータは、Oracle7 では廃止します。Oracle7 では、新規初期化パラメータ THREAD を使用します。このパラメータは、起動時に REDO ログ・ファイルをまとめて特定のインスタンスに対応付けます。

空き領域リスト

この項では、空き領域リストに関する変更について説明します。

削除および更新により使用可能となった領域

Oracle バージョン 6 では、削除または行を縮小する更新によって使用可能となったブロックは、空き領域の共通プールに追加されます。Oracle7 では、ブロックは、ブロックを削除したプロセスの空きリストおよび空きリスト・グループへ戻されます。

クラスタ用空きリスト

Oracle バージョン 6 では、記憶領域オプション FREELISTS および FREELIST GROUPS は CREATE CLUSTER 文には使用できません。また、ALLOCATE EXTENT 句は ALTER CLUSTER 文には使用できません。

Oracle7 では、クラスタ（大部分のハッシュ・クラスタを除く）が、複数の空きリストを使用できます。それには、CREATE CLUSTER の FREELISTS および FREELIST GROUPS 記憶領域オプションを指定し、ALTER CLUSTER ALLOCATE EXTENT (INSTANCE *n*) 文で、インスタンスにエクステントを割り当てます。

Oracle7 では、ハッシュ関数用のユーザー定義キーを使用してハッシュ・クラスタを作成した場合、そのハッシュ・クラスタは空きリストと空きリスト・グループを持つことができます。このキーは、インスタンスによって分割されます。

初期化パラメータ

FREELISTS および FREELIST GROUPS 記憶領域オプションは、Oracle バージョン 6 の初期化パラメータ FREE_LIST_INST および FREE_LIST_PROC にかわって使用されます。

インポート / エクスポート

Oracle バージョン 6 のエクスポートでは、空きリスト情報をエクスポートしません。Oracle7 のエクスポートおよびインポートでは、FREELISTS および FREELIST GROUPS を扱うことができます。

SQL*DBA

バージョン 6 では、接続していないときに STARTUP および SHUTDOWN を実行する必要があります。Oracle7 リリース 7.0 では、INTERNAL として、または SYSDBA か SYSOPER として接続しているときに STARTUP および SHUTDOWN を発行する必要があります。

Oracle7 では、コマンドまたは SQL*DBA メニュー・インタフェースのどちらを使用しても操作を実行できます。詳細は、『Oracle8*i* ユーティリティ・ガイド』で説明しています。

初期化パラメータ

この項では、新しいパラメータと廃止されたパラメータを示します。

新規パラメータ

新規初期化パラメータ THREAD は、起動時に REDO ログ・ファイルを特定のインスタンスに対応付けます。

新規パラメータの完全なリストは、『Oracle8*i* リファレンス・マニュアル』を参照してください。

廃止されたパラメータ

次に示す初期化パラメータは、以前のバージョンの Parallel Server Option で使用され、Oracle7 で廃止された初期化パラメータです。

ENQUEUE_DEBUG_MULTI_INSTANCE
FREE_LIST_INST
FREE_LIST_PROC
GC_SORT_LOCKS
INSTANCES
LANGUAGE
LOG_ALLOCATION
LOG_DEBUG_MULTI_INSTANCE
MI_BG_PROCS (renamed to GC_LCK_PROCS)
ROW_CACHE_ENQUEUE
ROW_CACHE_MULTI_INSTANCE

廃止されたパラメータの完全なリストは、『Oracle8*i* 移行ガイド』を参照してください。

アーカイブ

Oracle バージョン 6 では、全インスタンスが同じ REDO ログ・ファイルを共有しているため、各インスタンスがパラレル・サーバー全体のオンライン REDO ログ・ファイルをアーカイブします。したがって、記憶メディアへのアクセスが最も簡単なインスタンスに自動アーカイブを使用させ、他のインスタンスには手動でアーカイブさせることが可能です。

Oracle7 では、各インスタンスが固有のオンライン REDO ログ・ファイルの集合をもっているため、自動アーカイブでは現行インスタンスのオンライン REDO ログ・ファイルだけがアーカイブされます。また、Oracle7 では、クローズ状態のスレッドもアーカイブします。手動アーカイブを行えば、全インスタンスに対するオンライン REDO ログ・ファイルをアーカイブできます。ALTER SYSTEM ARCHIVE LOG 文の THREAD オプションを使用すると、任意の特定インスタンスの REDO ログ・ファイルをアーカイブできます。

Oracle7 では、アーカイブ済み REDO ログ・ファイルのファイル名には、スレッド番号およびログ順序番号を指定できます。

新規初期化パラメータ LOG_ARCHIVE_FORMAT は、アーカイブ済みファイル名のフォーマットを指定します。CREATE DATABASE 文内の新規データベース・パラメータ MAXLOGHISTORY は、制御ファイル内のアーカイブ履歴を保持するために指定できます。

メディア回復

Oracle7 では、メディア障害からのオンライン回復は、データベースがパラレル・モードまたは排他モードのどちらかでマウントした場合にサポートされます。

どちらのモードでも、回復中のデータベースまたはオブジェクトは使用できません。

- データベース全体を回復するには、マウントされている必要があるが、オープンしない。
- 表領域を回復するには、データベースはオープンされていて、表領域はオフラインになっている必要がある。
- データ・ファイル（SYSTEM 表領域内のファイルを除く）を回復するには、データベースはクローズまたはオープンされており、データ・ファイルがオフラインになってる必要がある。

B

制約

この付録では、Oracle Parallel Server の互換性の問題および制約についてまとめています。

- 互換性
- 制約

互換性

次の項では、パラレル・サーバーの共有モードと排他モード間の互換性の側面について説明します。

- Export ユーティリティと Import ユーティリティ
- 共有モードと排他モード間の互換性

Export ユーティリティと Import ユーティリティ

Export ユーティリティは、Oracle データベースのデータをオペレーティング・システムのファイルに書き込みます。Import ユーティリティは、オペレーティング・システムのファイルのデータを Oracle データベースに読み込みます。この Oracle の機能は、共有モードと排他モードで共通です。

関連項目：『Oracle8*i* ユーティリティ・ガイド』（Import および Export ユーティリティについて）

共有モードと排他モード間の互換性

OPS は、排他モードで作成されたどの Oracle データベースでも作動します。各インスタンスは、それぞれ専用の REDO ログの集合を持っている必要があります。

排他モードの Oracle は、OPS が作成または変更したデータベースにアクセスできます。

OPS が特定のインスタンスに空き領域を割り当てた場合、排他モードでは別のインスタンスがその領域に挿入することはできません。割当て済みエクステント内のデータはすべて、いつでも使用できます。

制約

次の各項では、制約について説明します。

- 1 回に割り当てられる最大ブロック数
- 共有モードでの制約

1 回に割り当てられる最大ブロック数

GC_FILES_TO_LOCKS パラメータの *!blocks* オプションによって、空きリスト・グループ内で使用可能なブロック数を制御できます。*!blocks* を使用すると、1 つのエクステント内に割り当てるブロックの割合を指定できます。最大で 1 回に 255 ブロックまで指定できます。

共有モードでの制約

複数のインスタンスを共有モードで実行する場合、Oracle は、次の各見出しで挙げる場合以外は、排他モードでの Oracle の機能をすべてサポートしています。

SQL 文の制約

共有モードでは、次の操作はサポートされていません。

- データベースの作成（CREATE DATABASE）
- 制御ファイルの作成（CREATE CONTROLFILE）
- データベースのアーカイブ・モードの切替え（ALTER DATABASE の ARCHIVELOG オプションおよび NOARCHIVELOG オプション）

これらの操作を行うには、18-12 ページの「インスタンスの起動」で説明しているように、すべてのインスタンスを停止してから、1 つのインスタンスを排他モードで起動してください。

データ・ファイルの最大数

Oracle でサポートするデータ・ファイルの数は、オペレーティング・システム固有のものです。この制限内で、許される最大数は、CREATE DATABASE コマンドで使用される値によって決まります。次に、この値は、制御ファイルの物理サイズによって制限されます。この制限は共有モードでも排他モードでも同じですが、OPS ではインスタンスが追加されるため、単一インスタンスのシステムの場合よりもファイルの最大数に制限があります。詳細は、『Oracle8*i* SQL リファレンス』およびオペレーティング・システム固有の Oracle マニュアルを参照してください。

順序番号ジェネレータ

OPS は、共有モードの順序番号ジェネレータの CACHE オプションおよび ORDER オプションの組合せをサポートしていません。これらのオプションを両方使用して作成された順序は、順序が付けられますが、パラレル・サーバーで実行中の場合はキャッシュに入れられません。

Import ユーティリティおよび Export ユーティリティでの空きリスト

Export ユーティリティで、複数の空きリストおよび空きリスト・グループについての情報は保存されません。複数のインスタンスからデータをエクスポートし、次に単一ノードからデータをファイルにインポートする場合、そのデータは、最初とまったく同じように複数のエクステントに分散されない場合があります。そのデータがインポートされる表のメタデータには、その空きリストおよび空きリスト・グループの情報が含まれており、これらの情報はその後、データ・ブロックと対応付けされます。

したがって、Export および Import を使用してデータのバックアップおよび復元を行う場合は、再度分割できるようにデータをインポートすることは困難です。

索引

数字

2 フェーズ・コミット，1-19

A

ADD LOGFILE 句
THREAD 句，6-3
必須スレッド，14-8
ADDED_EXTENTS，A-10
ALERT ファイル，6-2，22-4
ALL_TABLES 表，16-7
ALLOCATE EXTENT オプション
DATAFILE オプション，11-11，17-11
INSTANCE オプション，11-11，17-11
SIZE オプション，11-11，17-11
インスタンス番号，11-15，17-12
エクステントの事前割当て，17-13
使用不能，A-20
排他モードの，17-11
ALLOW_PARTIAL_SN_RESULTS パラメータ
8.1 で廃止，A-5
ALL オプション，21-4
ALTER CLUSTER 文，A-20
ALLOCATE EXTENT オプション，17-10
エクステントの割当て，17-13
ALTER DATABASE OPEN RESETLOGS 文，22-13
ALTER DATABASE 文
ADD LOGFILE，6-3
CLOSE オプション，18-16
DATAFILE OFFLINE オプションと DATAFILE ONLINE オプション，A-18
DATFILE RESIZE，15-9
DISABLE，14-8
ENABLE THREAD，14-8
MOUNT オプション，18-12
OPEN オプション，18-13
RECOVER，22-7
RECOVER オプション，22-7
REDO のスレッド，14-8
THREAD，14-8
ファイルの改名，A-18
ログ・ファイルの追加または削除，A-18
ログ・モードの設定，14-2，14-8，B-2
ALTER INDEX 文
DEALLOCATE UNUSED オプション，17-15
ALTER ROLLBACK SEGMENT コマンド，6-10
ALTER SESSION 文
SET INSTANCE オプション，11-11，17-9
ALTER SYSTEM
コマンド、パラレル実行用のインスタンスの制限，18-25
ALTER SYSTEM ARCHIVE LOG 文，18-17，21-15
CURRENT オプション，21-10，A-19
THREAD オプション，18-17，21-4
グローバル・ログ・スイッチ，21-10，21-15
ALTER SYSTEM CHECK DATAFILES 文，6-2
インスタンスの回復，22-4
ALTER SYSTEM CHECKPOINT 文，21-9
インスタンスの指定，18-16
グローバル対ローカル，18-16
ALTER SYSTEM SWITCH LOGFILE 文，18-17，21-10，A-19
DBA 権限，21-10，A-19
ALTER SYSTEM 権限，21-9，21-10
ALTER TABLESPACE 文
ADD DATAFILE，15-9
ADD DATAFILE オプション，A-18
BACKUP オプション，21-14
OFFLINE オプションと ONLINE オプション，A-18

READ ONLY オプション，12-2
データ・ファイルの改名，A-18
ALTER TABLE 文
ALLOCATE EXTENT オプション，17-10
DISABLE TABLE LOCK オプション，7-6，10-3，16-7
ENABLE TABLE LOCK オプション，7-6，10-3，16-7
MAXEXTENTS オプション，17-14
エクステントの割当て，17-13，17-14
ARCHIVE LOG 句
CURRENT オプション，21-10，21-15，A-19
THREAD オプション，21-4
インスタンスの指定，18-19
グローバル・ログ・スイッチ，21-10，21-15
手動アーカイブ，21-4
ARCHIVE LOG コマンド，21-3
ARCHIVELOG モード，14-8
オンラインおよびオフラインのバックアップ，4-7，21-3
自動アーカイブ，4-7，21-3
データベースの作成，14-2
モードの変更，14-2，14-8，B-2
ARCH プロセス，5-5，21-4
AST，8-2
AUTOEXTEND，15-9

B

BEGIN BACKUP オプション，21-14
BSP
定義，1-16，5-6

C

CACHE_SIZE_THRESHOLD パラメータ，A-17
8.1 で廃止，A-5
CACHE オプション、CREATE SEQUENCE，6-6
CACHE キーワード，18-25
CATPARR.SQL スクリプト，15-12，20-10
CHECK DATAFILES 句，6-2
インスタンスの回復，22-4
CHECKPOINT_PROCESS パラメータ，18-9
CKPT プロセス，5-5
CLEANUP_ROLLBACK_ENTRIES パラメータ，A-9，A-13
Cluster Manager（CM）ソフトウェア，18-12，18-13
用途，18-20
CONNECT
RMAN ALLOCATE CHANNEL コマンド，21-8
CONNECT コマンド，14-6，18-17，18-19
Net8，18-16
SYSDBA オプション，18-12，18-24
チェックポイントの強制，21-9
CONTROL_FILES パラメータ，18-10，22-11
すべてのインスタンスで同じ値，18-7，18-10
CPU_COUNT パラメータ，18-10
CPU 使用率，19-2
CREATE CLUSTER 文，17-7，A-20
FREELIST GROUPS オプション，17-6
FREELISTS オプション，17-6
CREATE CONTROLFILE 文
MAXLOGHISTORY，21-6
データベース・オプションの変更，14-10
排他モード，B-2
複数インスタンスへの変換，23-5
CREATE DATABASE 文，14-2
MAXINSTANCES，11-15，14-3
MAXLOGFILES，14-3
MAXLOGHISTORY，6-5，14-4，21-6
MAXLOGMEMBERS，14-3
オプション，14-3
スレッド番号 1，14-8
排他モード，B-2
ログ・モードの設定，14-2，14-8
CREATE INDEX 文
FREELISTS オプション，17-8
CREATE ROLLBACK SEGMENT 文，14-5
CREATE SEQUENCE 文，6-7
CACHE オプション，6-6，6-7
CYCLE オプション，6-6
ORDER オプション，6-6，6-7
説明，6-6
CREATE TABLESPACE 文，A-18
CREATE TABLE 文
FREELIST GROUPS オプション，11-11
FREELISTS オプション，11-11，17-6
クラスタ化された表，17-7
初期記憶領域，17-12，17-13
例，17-13
CR サーバー
8.1 での変更の概要，A-1
CURRENT オプション，21-4
Oracle7 で新しく追加された，A-19

グローバル・ログ・スイッチ，21-15
グローバル・ログ・スイッチの強制，21-10
チェックポイント，21-10
CYCLE オプション、CREATE SEQUENCE，6-6

D

DATAFILE オプション
表，17-13
表領域，A-18
DB_BLOCK_BUFFERS パラメータ，9-17
GC_RELEASABLE_LOCKS，15-13
LM ロック容量の確保，16-6
DB_BLOCK_SIZE パラメータ
すべてのインスタンスで同じ値，18-10
DB_FILES パラメータ
LM ロック容量の確保，16-6
すべてのインスタンスで同じ値，18-10
非 PCM リソース数の計算，16-4
DB_NAME パラメータ
すべてのインスタンスで同じ値，18-10
複数インスタンスへの変換，23-7
DBA_ROLLBACK_SEGS ビュー，6-8，14-6
パブリック・ロールバック・セグメント，14-5
DBA_SEGMENTS ビュー，14-6
DBA_TABLES 表，16-7
DBMS_JOB パッケージ
インスタンスの親和性，4-10
DBMS_SPACE パッケージ，17-15
DBMSUTIL.SQL スクリプト，17-15
DBWR プロセス，5-5，7-6
パラレル・サーバー，5-2
DDL
コマンド，16-6
ロック，7-5
DELAYED_LOGGING_BLOCK_CLEANOUTS パラメータ，18-9，A-9，A-14
DESCRIPTION_LIST 機能，A-13
DISABLE TABLE LOCK オプション，16-7
DISABLE THREAD 句，14-8
DISCONNECT コマンド，18-19
DML_LOCKS パラメータ，7-6，10-3，18-9，18-10
IDLM ロック容量の確保，16-6
パフォーマンス，16-7
非 PCM リソース数の計算，16-4
DML ロック，7-3，7-5
DM、データベース・マウント，10-5
DROP TABLESPACE 文，A-18
DROP TABLE コマンド，4-17
DSS アプリケーション，1-5，1-12，1-21，2-7，3-2

E

ENABLE TABLE LOCK オプション，16-7
END BACKUP オプション，21-14
ENQUEUE_DEBUG_MULTI_INSTANCE パラメータ（Oracle バージョン 6），A-21
ENQUEUE_RESOURCES パラメータ
非 PCM リソース数の計算，16-4
EXCLUSIVE オプション，18-12
EXCLUSIVE パラメータ
8.0.4 で廃止，A-5
Export ユーティリティ
空きリスト，11-12，A-20，B-3
互換性，B-1
データのバックアップ，B-1
EXT_TO_OBJ 表，15-12，20-10

F

false ping，9-17，15-15
FAST_START_PARALLEL_ROLLBACK，A-3
FILE_LOCK ビュー，9-21，15-11，A-9
FORCE パラメータ
ジョブのインスタンスとの親和性，4-10
FREE_LIST_INST パラメータ（Oracle バージョン 6），A-20，A-21
FREE_LIST_PROC パラメータ（Oracle バージョン 6），A-20，A-21
FREED_EXTENTS，A-10
FREELIST GROUPS 記憶領域オプション，17-6，17-13，18-13
インスタンス番号，11-15
クラスタ化された表，A-20
FREELISTS
記憶領域オプション，17-6
パラメータ，11-5
FREELISTS 記憶領域オプション
クラスタ化された表，A-20
最大値，17-6
索引，17-8
FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY パラメータ，22-4，A-6

G

GC_DB_LOCKS パラメータ，A-6
　ファイル操作後の調整，A-18
GC_FILES_TO_LOCKS パラメータ，9-3，9-4，9-6，9-13，9-23，14-9，15-17，18-10
　false ping の減少，15-16
　PCM ロックとエクステントの対応付け，11-16，17-10
　ガイドライン，15-9
　構文，15-6
　索引データ，15-5
　成長の余地，15-10
　設定，15-6
　データ・ファイルの追加，15-12
　デフォルト・バケット，15-7
　ハッシュの例，15-8
　ファイル操作後の調整，15-6，A-18
　ファイン・グレインの例，15-8
GC_FREELIST_GROUPS パラメータ，A-6，A-9
GC_LATCHES パラメータ，A-5
　8.1 で廃止，A-5
GC_LCK_PROCS パラメータ
　8.1 で廃止，A-5
GC_RELEASABLE_LOCKS パラメータ，9-13，15-17，A-9
　デフォルト，15-13
GC_ROLLBACK_LOCKS パラメータ，6-9，9-13，9-14，15-7，15-13，15-17，18-10
GC_ROLLBACK_SEGMENTS パラメータ，A-6
　分散ロックの数，6-9
GC_SAVE_ROLLBACK_LOCKS パラメータ，6-9，A-6，A-19
GC_SEGMENTS パラメータ，A-6
GC_SORT_LOCKS パラメータ，A-21
GC_TABLESPACES パラメータ，A-6
GLOBAL オプション
　チェックポイントの強制，18-16，21-9
　ファイルへのアクセスの検証，6-2
GLOBAL ヒント，20-11
GMS
　8.1 では削除，A-2
GROUP オプション，21-4
GV$BH ビュー，19-5
GV$CACHE ビュー，19-5，20-10
GV$CLASS_PING ビュー，20-10
GV$FILE_PING ビュー，20-10
GV$PARAMETER ビュー，18-22
GV$PING ビュー，19-5，20-10
GV$ ビュー，18-22，20-10，A-7

H

HOST コマンド，18-17

I

IDLM，8-1
IDLM パラメータ，18-11
IFILE パラメータ，18-5
　値の上書き，18-7
　同一パラメータの指定方法，18-6
　複数ファイル，18-7
Import ユーティリティ
　空きリスト，A-20
　互換性，B-1
　データの復元，B-1
INITIAL 記憶領域パラメータ
　最小値，17-12
　ロールバック・セグメント，6-8
INSTANCE NAME パラメータ，18-6
INSTANCE_GROUPS パラメータ，18-21
INSTANCE_ID 列，20-11
INSTANCE_NUMBER パラメータ，17-9，18-5
　SQL オプション，11-11
　インスタンスの一意の値，18-9
　インスタンスの一意の番号，18-15
　インスタンスへの空きリストの割当て，11-13
　個別パラメータ・ファイル，18-5
　設定，17-12
　排他モード，18-13
　排他モードまたは共有モード，18-15
　複数インスタンスへの変換，23-7
　未指定，18-15
INSTANCES キーワード，18-25
INSTANCES パラメータ（Oracle バージョン 6），A-21
INSTANCE オプション
　SET INSTANCE コマンド，17-9，18-16
　SHOW INSTANCE コマンド，18-18
　割当て，17-13
INTERNAL オプション
　インスタンス停止，18-24
I/O
　拡張性，2-3

最少化，1-16，4-6，4-13，15-5
ディスクの競合，4-6
割込み，2-5
IPC
キャッシュ・フュージョン，20-6

L

Lamport SCN 生成，4-8
LANGUAGE パラメータ（Oracle バージョン 6），A-21
LCKn プロセス，5-5，9-5
回復におけるロール，22-5
複数インスタンス・データベース，1-15
LGWR プロセス，5-5，7-6
ログ履歴，21-6
LM_LOCKS パラメータ，15-17，15-18，16-2，18-10，18-11，A-6
LM_PROCS パラメータ，18-10，18-11，A-6
LM_RESS パラメータ，15-17，15-18，16-2，16-4，18-10，18-11，A-6
LMDn プロセス，5-5，A-7
定義，7-7
LMON プロセス，5-5，A-7
定義，7-7
LOAD_BALANCE パラメータ
クライアント・ロード・バランシングの構成，18-29
LOCAL オプション
チェックポイントの強制，18-16，21-9
ファイルへのアクセスの検証，6-2
LOG_ALLOCATION パラメータ（Oracle バージョン 6），A-19，A-21
LOG_ARCHIVE_DEST パラメータ，22-9，22-10
回復に関する指定，22-10
LOG_ARCHIVE_FORMAT パラメータ，18-9，21-5，22-10
回復での使用，22-10
すべてのインスタンスで同じ，22-10
LOG_ARCHIVE_START パラメータ
自動アーカイブ，18-7，21-3
データベースの作成，14-2
LOG_CHECKPOINT_TIMEOUT パラメータ
アクティブでないインスタンス，21-10
LOG_DEBUG_MULTI_INSTANCE パラメータ（Oracle バージョン 6），A-21
LOG_FILES パラメータ
8.1 で廃止，A-5
LRU リスト
ロック要素，9-19

M

MAX_COMMIT_PROPAGATION_DELAY パラメータ，4-8，10-4，18-9，18-10
MAX_SORT_SIZE，A-10
MAXDATAFILES オプション，14-10
MAXEXTENTS 記憶領域パラメータ
エクステントの事前割当て，17-13
自動割当て，11-17，17-12
MAXINSTANCES オプション，11-15，14-3
変更，14-10
MAXINSTANCES パラメータ，11-13，23-4
インスタンスへの空きリストの割当て，11-13，11-15
非 PCM リソース数の計算，16-4
複数インスタンスへの変換，23-4，23-5
MAXLOGFILES オプション，14-3，14-10
MAXLOGHISTORY オプション，6-5，14-4，21-6
CREATE CONTROLFILE，21-6
変更，14-10
ログ履歴，21-6
MAXLOGHISTORY パラメータ，23-6
MAXLOGMEMBERS
オプション，14-3，14-10
MINEXTENTS 記憶領域パラメータ
自動割当て，11-17，17-12，17-13
デフォルト，17-12
MONITOR コマンド
インスタンスの指定，18-19
表示画面におけるデフォルト・インスタンス，18-17
MOUNT オプション，18-12
MPP システム，1-5
MTS_LISTENER_ADDRESS パラメータ
8.0.4 で廃止，A-5
MTS_MULTIPLE_LISTENERS パラメータ
8.0.4 で廃止，A-5

N

Net8
CONNECT 用，18-16
SET INSTANCE 用，18-16
クライアント / サーバー構成，1-20

接続文字列，18-18，18-19
分散データベース・システム，1-18
NEXT 記憶領域パラメータ，6-8，21-4
NFS，5-8，23-4，23-9
NLS_* パラメータ，18-9
NOARCHIVELOG モード，14-8
オフライン・バックアップ，4-7
オフライン・バックアップを必要とする，21-3
データベースの作成，14-2，14-8
モードの変更，14-2，14-8，B-2
NOMOUNT オプション，14-3，22-11
NOORDER オプション、CREATE SEQUENCE，6-7
NSTANCE_GROUPS パラメータ，18-22
NULL ロック・モード，4-16

O

OGMS_HOME パラメータ，A-5
8.1 で廃止，A-5
OLTP アプリケーション，1-3，1-5，1-8，1-12，2-7，2-8，3-2，A-2
OPEN オプション，18-12，18-13
OPS_ADMIN_GROUP パラメータ，A-6
8.1 で廃止，A-5
OPS_FAILOVER 句，A-11
Oracle
MPP ノードのインスタンス，3-7
移行，A-17
インスタンスの回復，22-3
共有モード，4-2
構成，1-14
互換性，17-11，B-2
使用不能な空き領域，17-11
実行可能ファイル，23-7
すべてのノードでのバージョン，5-8
制限，6-7，A-17，A-19
単一インスタンス，7-3
データ・ディクショナリ，6-6
データ・ファイルの互換性，6-2
廃止されたパラメータ，A-21
排他モード，4-2，18-12
バックアップ，4-7，21-1
バックグラウンド・プロセス，5-5
パフォーマンス機能，4-6
複数インスタンス，7-3
Oracle Parallel Server Management（OPSM），A-4
oracle_pid，11-13
Oradebug，A-4
ORDER オプション，6-6，6-7

P

PARALLEL_DEFAULT_MAX_INSTANCES パラメータ
8.1 で廃止，A-5
PARALLEL_DEFAULT_MAXSCANS パラメータ，18-10
PARALLEL_INSTANCE_GROUP パラメータ，18-22
PARALLEL_MAX_INSTANCES、初期化パラメータ，18-25
PARALLEL_MAX_SERVERS パラメータ，18-25，22-14，22-15
LM ロック容量の確保，16-6
非 PCM リソース数の計算，16-4
PARALLEL_SERVER パラメータ，18-9，18-12，18-13，18-14
8.0 で新規追加，A-5
PARALLEL_TRANSACTION_RECOVERY パラメータ
8.1 での変更，A-3
PARALLEL オプション，18-12
PARALLEL パラメータ
8.0.4 で廃止，A-5
PCM ロック
解放可能なハッシュ，9-4
解放可能なファイン・グレイン，9-3
競合，11-16，15-5，17-10
共有，9-24，15-5
計画，15-2
計算，15-17
固定ハッシュ，9-3
固定ファイン・グレイン，9-4
合計数の指定，14-9
索引データ，15-5
データ・ファイルの追加，15-12
排他，9-24
必要数の見積り，15-3
ファイルの集合，9-7
ブロックのマップ，9-6，11-16，17-10
変換時間，15-16
待っているセッション，15-17
有効な数のチェック，15-11，15-13
有効なロック割当て，15-11
リソース，15-17
ワークシート，15-4

割当て，15-2
PCTFREE，11-5，19-6
PCTINCREASE パラメータ
表エクステント，17-11
PCTUSED，11-5
PFILE オプション，18-6，18-8
複数インスタンスへの変換，23-5
ping，9-8，9-9，15-14，15-16
false，9-17
定義，9-3
率，15-15
ping / 書込み率，19-4
PL / SQL 共有メモリー領域，5-5
PMON プロセス，5-5
PROCESSES パラメータ，18-9
LM ロック容量の確保，16-6
非 PCM リソース数の計算，16-4
PUBLIC スレッド，14-8

R

reader / writer 競合
キャッシュ・フュージョン，20-1
RECOVER DATABASE 文，22-7，22-8
RECOVER DATAFILE 文，22-7，22-8
RECOVER TABLESPACE 文，22-7，22-8
Recovery Manager，22-6
アーカイブ・ログ・バックアップ，21-7
災害時回復，22-10
不完全メディア回復，22-8
メディア回復，22-8
RECOVERY_PARALLELISM パラメータ，18-9，22-1，22-14，22-15
RECOVER コマンド，18-17，22-7，22-13，22-15
RECO プロセス，1-17，5-5
REDO スレッド，21-4
REDO ログ
REDO ログ・バッファ，5-5
アーカイブ・モード，21-3
インスタンスの回復，22-3
再構成，14-9
ログ履歴，21-6
REDO ログ・ファイル
REDO のスレッド，6-3
アーカイブ，4-7，14-8，21-1，21-3，21-10
アクセス可能性，5-8
上書き，4-7，21-3
改名，21-10，A-18
削除，21-10，A-18
制御ファイル内で識別される，6-5
多重，21-7
追加，A-18
バックアップ，21-11
ログ順序番号，21-5
ログ履歴，21-6
REDO ログ・ファイルのアーカイブ，21-1
オンライン・アーカイブ，4-7，21-3
自動と手動，21-3
制御ファイル内で識別される，6-5
データベースの作成，14-2
複数インスタンスへの変換，23-3
履歴，21-6
ログ順序番号，21-5
ログ・スイッチの強制，21-10
REDO ログ・ファイルの削除，A-18
手動アーカイブ，14-8
制限，14-9
ログ・スイッチ，21-10
REMOTE_LOGIN_PASSWORDFILE パラメータ，18-23
RESTORE DATABASE 文，22-8
RESTORE DATAFILE 文，22-8
RESTORE TABLESPACE 文，22-8
RETRY オプション，18-14
STARTUP PARALLEL コマンド，18-14
RMAN
ALLOCATE CHANNEL コマンド，21-8
ROLLBACK_SEGMENTS パラメータ，6-9，6-10，18-9，18-10
複数インスタンスへの変換，23-7
プライベート・セグメントおよびパブリック・セグメント，14-5
ROW_CACHE_MULTI_INSTANCE パラメータ（Oracle バージョン 6），A-21
ROW_LOCKING パラメータ，18-10

S

SCN，4-5
システム変更番号，10-4
SCSI，3-3
SEQUENCE_CACHE_ENTRIES パラメータ
8.1 で廃止，A-5
SERIALIZABLE パラメータ，18-10

Server Manager
　権限付きのコマンド，18-18
SERVICE_NAMES パラメータ
　インスタンス登録，18-28
SESSIONS パラメータ
　LM ロック容量の確保，16-6
SET INSTANCE コマンド，18-6，18-17，18-18
　インスタンス起動，18-6，18-18
　必要な Net8，18-16
SET UNTIL コマンド，22-11
SHARED オプション，18-12
SHOW INSTANCE コマンド，18-17，18-18
SHOW PARAMETERS コマンド，18-17，18-19
　インスタンス番号，18-15
SHOW SGA コマンド，18-17，18-19
SHUTDOWN コマンド
　ABORT オプション，18-24
　IMMEDIATE オプション，18-24
　インスタンスの指定，18-18
SID パラメータ，18-6
SINGLE_PROCESS パラメータ，18-14
SIZE オプション
　エクステントの割当て，17-13
SMON プロセス，5-6
　SHUTDOWN ABORT 実行後の回復，18-24
　インスタンスの回復，22-2，22-3
　トランザクション回復，A-13
SMP，1-14
SORT MERGE JOIN，12-3
SORT_DIRECT_WRITES パラメータ，A-14
SQL 文
　インスタンス固有の，18-16
　制限，B-2
SQL 領域
　共有，12-6
STARTUP コマンド，14-2，18-6，18-12
　MOUNT オプション，22-13
　OPEN オプション，18-12
　PFILE オプション，18-6，18-8
　インスタンスの指定，18-18
SYSDBA，18-12，18-19，18-24，21-11
SYSOPER，18-12，18-19，18-24，21-11
SYSTEM 表領域，14-4，14-5
SYSTEM ロールバック・セグメント，6-8

T

TABLE_LOCK 列，16-7
THREAD オプション，18-17，21-4，21-10
　スレッドの使用禁止，14-8
　パブリック・スレッドの作成，6-3
　必要な場合，14-8
　プライベート・スレッドの作成，6-3
THREAD パラメータ，14-7，18-5，18-9
　インスタンス獲得スレッド，6-3
　個別パラメータ・ファイル，18-5
　複数インスタンスへの変換，23-7
TM、DML エンキュー，10-3
TNSNAMES.ORA
　インスタンス登録用の構成，18-26
TP モニター，A-11
TRANSACTIONS_PER_ROLLBACK パラメータ，6-10
TRANSACTIONS パラメータ，6-10
　LM ロック容量の確保，16-6
　非 PCM リソース数の計算，16-4
TX、トランザクション，10-3

U

USER_TABLES 表，16-7

V

V$ACTIVE_INSTANCES ビュー，23-4，A-9
V$BH ビュー，9-10，19-5，A-7，A-9，A-17
V$CACHE_LOCK ビュー，20-10
V$CACHE ビュー，19-5，A-17
V$CLASS_PING ビュー，A-7
V$DATAFILE ビュー，6-3，15-11
V$DLM_ALL_LOCKS
　8.1 の新しいビュー，A-2
V$DLM_ALL_LOCKS ビュー，7-5，8-11
V$DLM_CONVERT_LOCAL ビュー，8-11，A-7
V$DLM_CONVERT_REMOTE ビュー，8-11，A-7
V$DLM_LATCH ビュー，A-7
V$DLM_LOCKS ビュー，7-5，8-11
　8.0.4 で変更，A-6
V$DLM_MISC ビュー，8-11，A-7
V$DLM_RESS
　新しいビュー，A-2
V$DLM_RESS ビュー，7-5，8-11
V$FAST_START_SERVERS

ビュー，22-15，A-3
V$FAST_START_TRANSACTIONS
ビュー，22-15，A-3
V$FILE_PING ビュー，A-7
V$LE 表，9-19
V$LOCK_ACTIVITY ビュー，20-10，A-17
ロック変換の検出，19-3
V$LOCK_ELEMENT ビュー，7-9，9-19
V$LOCKS_WITH_COLLISIONS ビュー，20-10，A-16
V$LOCK ビュー，7-8
V$LOG_HISTORY ビュー，21-7
V$LOGFILE ビュー，6-5
V$PING ビュー，19-5，19-6，A-17
V$RECOVERY_LOG ビュー，21-7
V$RESOURCE_LIMIT ビュー，16-3，A-7
V$ROLLNAME ビュー，20-10
V$SESSION_WAIT ビュー，15-17
V$SORT_SEGMENT ビュー，A-9，A-10
V$SYSSTAT ビュー，15-13，22-5，A-7
ロック変換の検出，19-3
V$SYSTEM_EVENT ビュー，15-16
V$THREAD ビュー，23-4
V$WAITSTAT ビュー，19-5
VIA
インターコネクト・プロトコル，20-6

X

XA_RECOVER コール，A-11
XA インタフェース
回復機能の強化，A-11
パフォーマンス強化，A-11
ライブラリ，8-10

あ

アーカイブ・モードの切替え，14-2，14-8，B-2
アーカイブ履歴の使用禁止，14-4
アーカイブ・ログ
バックアップ，21-7
アーキテクチャ
Oracle インスタンス，5-1
Oracle データベース，6-1
ハードウェア，3-1
複数インスタンス，5-4
空きリスト，11-15
PCM ロック，11-16，17-10
インスタンスに割り当てられた，11-13
エクステント，11-12
および Export ユーティリティ，11-12，B-3
競合，19-5
クラスタ，17-7，A-20
索引，17-8
定義，11-5
データの分割，11-11，18-15
トランザクション，11-4
同時挿入，4-8，11-14
排他モード，17-6，17-11，B-2
ハッシュ・クラスタ，17-7
分割，11-14
未使用領域，17-15
リスト・グループ，11-12
リストの数，17-6
空きリスト・グループ
!blocks の設定，15-8
インスタンスに割り当てられた，11-13，11-14
高パフォーマンス機能，4-9
使用済み領域，17-15
セッションへの割当て，17-9
定義，11-5
未使用領域，17-15
リリース 7.3 用に強化，A-9
アップグレード
Oracle，23-1
レプリケート・システム，23-1
アプリケーション
DSS，1-5，1-12，1-21，2-7
OLTP，1-5，1-12，2-7
移植性，1-5
拡張性，2-2，2-6
可用性，22-2
業務機能，12-2
計算集中型，1-21
挿入集約型，11-12
設計，13-1
調整，12-1
問合せ集中処理型，1-17，2-7
ノード，5-4
パフォーマンス，11-12
パフォーマンスの調整，1-14，11-12
パラレル処理のための再設計，1-15
フェイルオーバー，4-12
複数インスタンスへの変換，23-3
部門別処理，2-8

分割データ，1-17，2-7
分析，12-1
プロファイル，3-2，12-2
ランダム・アクセス，2-8

い

移行
今後のための計画，1-14
データの移行，B-1
排他モードへの復帰，17-11
一貫性
複数バージョン読込み，4-7
ロールバック情報，6-8
インスタンス
PCM ロックの所有権，9-6
空きリスト，11-14，17-11
インスタンスの追加，14-3，17-12，23-2
インスタンス番号，11-15，17-12
エクステントに関連付けられた，17-9
回復，14-3，18-24，22-2
起動順序，18-15
グループ、指定，18-21
グループ、使用，18-22
現行，18-18，18-19，21-9
現行インスタンスの変更，18-19
最大数，6-8，11-15，14-3
障害，22-3
親和性、ジョブ，4-10，A-4
スレッド番号，6-3，14-7，18-14
データ・ファイルに関連付けられた，17-12
データ・ブロックとの関連付け，11-11
デフォルトの変更，18-18
バックグラウンド・プロセス，4-14，5-4，5-5
番号，17-9
パラレル・サーバーの特性，5-1
必要なロールバック・セグメント，6-8
リモート，18-6，18-8，18-18
インスタンス登録，A-10
用途，18-26
インスタンスの回復
異常停止，18-24
グローバル・チェックポイント，21-9
ファイルへのアクセス，6-2，22-4
複数の障害，22-3
別のインスタンスの起動，14-3
ロールバック・セグメント，6-8
インスタンスのシャットダウン，18-24
REDO ログ・ファイルのアーカイブ，21-10
アーカイブされていないログ・ファイル，21-4
異常停止，18-24
失われた順序番号，6-7
起動順序の変更，18-15
ログ・スイッチの強制，21-10
インスタンスの接続の切断，18-18，18-19
複数のセッション，18-24
ユーザー・プロセス，18-24
インスタンス番号，11-13
インスタンス・ロック，7-2
タイプ，7-5
定義，7-2，7-4
バックグラウンド・プロセスによる取得，7-6
フォアグラウンド・プロセスによる取得，7-6
インターコネクト，3-3
OPS 用のプロトコル，20-6
拡張性，2-2
高速，1-9

え

エクステント
PCM ロックの割当て，11-16，17-10
インスタンスに割り当てられていない，11-6，
11-17，17-11
インスタンスへの割当て，17-9，17-13，18-15
サイズ，6-8，14-7，17-11
初期割当て，17-12
定義，11-3
ファイルの指定，17-11
ロールバック・セグメント，6-8，14-7
エクステントの事前割当て，11-17
エラー・メッセージ
記憶領域オプション，17-6
パラメータ値，18-15
ロールバック・セグメント，14-5
エンキュー，7-3
OPS コンテキスト，7-5
V$LOCK，7-8
グローバル，7-3
ローカル，7-3
エンキュー・ロック
非 PCM リソース数の計算，16-4

お

応答時間，1-9
オーバーヘッド
　非 PCM リソース数の計算，16-4
オフライン・データ・ファイル，A-18
オフライン・バックアップ，4-7，21-1
　REDO ログ・ファイル，21-11
　パラレル，21-11
オフライン表領域
　制限，6-8，A-18
　遅延ロールバック・セグメント，A-19
オペレーティング・システム
　エクスポート・ファイル，B-1
　拡張性，2-5
　統合分散ロック・マネージャ，9-8
　特権，18-18
オンライン REDO ログ・ファイル
　REDO のスレッド，6-3，18-14
　アーカイブ，21-1，21-6
　アーカイブ・ログ・モード，14-8
　ログ・スイッチ，21-6，21-10
オンライン回復，6-2，22-2，22-4，22-7，A-22
オンライン・データ・ファイル
　サポートされている操作，A-18
オンライン・トランザクション処理，1-3
オンライン・バックアップ，4-7，21-1
　REDO ログ・ファイル，21-11
　手順，21-15
　パラレル，21-11
　ログ・ファイルのアーカイブ，21-15

か

回復，22-1
　FREEZE_DB_FOR_INSTANCE_RECOVERY，22-4
　PARALLEL_MAX_SERVERS パラメータ，22-14，22-15
　Recovery Manager，22-8
　REDO ログの使用，21-11
　SHUTDOWN ABORT 実行後，18-24
　アーカイブ履歴，14-4
　インスタンス，14-3，18-24，22-2
　インスタンス障害，22-1
　インスタンスの回復，22-1
　エラーの検出，19-8
　オフライン・バックアップ，22-10
　オンライン，22-2
　オンライン・バックアップ，22-10
　回復時間，21-10
　グローバル・チェックポイント，21-9
　災害時，22-10，22-13
　自動，21-7
　制限，A-22
　単一ノード障害，22-2
　遅延トランザクション，A-12
　定義，22-1
　パラレル，22-14，22-15
　ファイルへのアクセス，6-2，22-4
　フェーズ，22-5
　不完全メディア，22-8
　複数インスタンスへの変換，23-9
　複数ノード障害の，22-3
　並列性の設定，22-14，22-15
　別のインスタンスの起動，14-3
　メディア障害，6-2，21-7，21-10，22-6，22-7，A-22
　ロールバック，6-8
　ログ履歴，21-7，22-9
解放可能な空きリスト待ち，15-13
解放可能なハッシュ PCM ロック，9-4，15-7，15-8
拡張性
　4 つのレベル，2-1
　アプリケーション，2-2，2-6，2-12
　オペレーティング・システム，2-5
　可能性，1-12
　関連，2-8
　共有メモリー・システム，2-5
　決定要因，1-13
　定義，1-8
　ディスク入出力，2-3
　データベース，2-5
　ネットワーク，2-6
　ハードウェア，2-2
　リリース 7.3 における機能強化，A-11
獲得 AST，8-2，8-4
仮想メモリーの使用率，19-2
可用性，12-1
　インターコネクト，3-3
　回復フェーズ，22-5
　共有ディスク・システム，3-6
　単一ノード障害，2-12，22-2
　データ・ファイル，6-2，22-4
　パラレル・データベースの利点，1-13
　複数データベース，1-19

き

記憶領域オプション
　エクステント・サイズ，6-8，17-11，17-12，17-13
　クラスタ化された表，17-6，A-20
　索引，17-8
　表，17-6
　ロールバック・セグメント，6-8
起動
　インスタンスの回復時，14-3
　起動順序，18-15
　共有モード，18-14，A-18
　グローバル定数パラメータ，6-9，18-8
　データベースの作成および，14-2
　排他モード，17-12，18-11
　ファイル操作後，15-6，A-18
　ファイルへのアクセスの検証，6-2
　リモート・インスタンス，18-6，18-7，18-8，18-18
　ロールバック・セグメント，6-8，14-5
キャッシュ
　一貫性，4-13，9-2
　一貫性、MPP，3-8
　回復，22-6
　管理について、非 PCM，4-17
　行キャッシュ，6-6
　順序キャッシュ，4-8，6-7
　ディクショナリ・キャッシュ，4-14，5-5，6-6
　ディクショナリのフラッシュ，4-17
　バッファ・キャッシュ，4-6，5-5
　パラレル・キャッシュ管理，4-14
キャッシュ・フュージョン
　8.1 での変更の概要，A-1
　アーキテクチャ，5-6
　キャッシュ一貫性矛盾，20-2
　説明，20-3
　調整，20-1
　定義，4-7，5-6
　パフォーマンス，20-1
　利点，20-4
競合
　1 つのブロックまたは行に対する，2-9
　SYSTEM 表領域，14-4
　空きリスト，19-5
　空き領域，4-8，11-2，11-14
　索引，19-6
　順序番号，4-8，6-6
　セグメント・ヘッダー，11-14，19-5
　ディスク，6-2，6-9
　表データ，6-2，6-8，17-12
　ブロック，6-9，11-16，15-5，17-10，17-12
　分散ロック，15-5
　ロールバック・セグメント，6-8，6-9
共有 SQL 領域，5-5，12-6
共有ディスク・システム
　拡張性，2-3
　実現，3-3
　非共有システムと，3-9
　利点，3-6
共有排他モード，8-7
共有メモリー・システム
　拡張性，2-5
　緊密結合，3-3
共有モード
　インスタンスの回復，22-2
　インスタンス番号，18-15
　回復制限，22-7
　起動，18-14
　データ・ファイル，6-2
　データベース・アクセス，4-2
　ファイル操作の制限，A-18
許可キュー，8-3，8-4
緊密結合システム
　実現，3-2
　疎結合クラスタ，3-5
　ハードウェア・アーキテクチャ，3-3，3-5
　パフォーマンス，3-4
行キャッシュ，6-6
行レベル・ロック，7-2
　DML ロック，10-3
　PCM ロックから独立した，4-16
　リソース共有システム，4-7，5-4

く

クライアント / サーバー構成，5-4
　Net8，1-20
　説明，1-20
クライアント・ロード・バランシング，18-29
クラスタ
　空きリスト，11-11，17-7
　空きリスト・グループ，17-11
　エクステントの割当て，17-13
　実現，2-3
　ハッシュ・クラスタ，11-11，17-7，A-20

バージョン間の互換性，18-20
パフォーマンス，3-2
クローズ状態のスレッド，21-4，21-15
グループ
MAXLOGFILES，14-3
REDO ログ・ファイル，6-4，14-3，14-9
V$LOGFILE，6-5
空きリスト，11-12
一意の番号，6-5
グループ・ベースのロック，8-9，8-10
グループ・メンバーシップ・サービス，A-8
8.1 では削除，A-2
グローバル定数パラメータ，18-10
すべてのインスタンスで同じ値，18-8，18-10
制御ファイル，6-1
説明，9-12
非 PCM ロック，4-13
リスト，18-8
ロールバック・セグメント，6-9
グローバル動的パフォーマンス・ビュー，18-22，20-10，A-7

け

形式、ロック名，7-8
権限
ALTER SYSTEM，21-9，21-10
現行インスタンス，18-19
指定，18-16
チェックポイント，21-9
ログ・スイッチ，21-10

こ

更新
PCM ロック，4-16
空きリスト，11-14，18-15
インスタンス・ロック，9-8
異なるタイミング，2-7
同時，4-7，11-14
パフォーマンス，11-12
高水位標，11-17
移動，11-18，11-19
定義，11-3，11-18
構成
Oracle の概要，1-14
REDO ログにおける変更，14-9
クライアント / サーバー，1-20
単一インスタンス・データベース・システム，1-15
パラレル・サーバーについてのガイドライン，5-8
複数インスタンス・データベース・システム，1-15
高速インターコネクト，12-1
高速バス，3-5，3-6
固定ハッシュ PCM ロック，9-3
固定モード、ロック要素，9-19
コミットされたデータ
インスタンス障害，22-3
順序番号，6-7
チェックポイント，21-9
互換性
共有モードおよび排他モード，6-2，17-11

さ

サービス接続
クライアント，18-26
サービスへのクライアント接続，18-26
災害時回復，22-10，22-13
作業負荷
均衡化，1-6
混在，1-5
スケールアップ，1-12
タイプ，1-5，1-12
分割，1-21
索引
FREELISTS オプション，17-8
PCM ロック，15-5
競合，19-6
作成，17-8
データの分割，11-11，15-5

し

識別子、ロック，7-8
システム・グローバル領域（SGA）
SHOW SGA コマンド，18-19
インスタンス，5-4
行キャッシュ，6-6
順序キャッシュ，6-6
統計，18-19
パラメータ・ファイル，18-3
パラレル・サーバー，5-5
システム固有の Oracle マニュアル
MAXLOGHISTORY のデフォルト値，21-6

Net8 接続文字列，18-16
Oracle 構成をサポートする，1-14
REDO ログのアーカイブ先，21-6
REDO ログのアーカイブ・フォーマット，21-6
アーカイブ済み REDO ログの名前，21-5
空きリストのオーバーヘッド，11-5
インスタンス番号範囲，17-11
回復プロセスの割当て，22-15
クライアント / サーバー処理，1-20
システム変更番号
Lamport，4-8
SCN，4-5
データ・ファイル、最大数，B-3
パスワード・ファイル名，18-23
システム変更番号（SCN），10-4
Lamport，4-8
REDO ログの履歴，21-6
REDO ログ・ファイルのアーカイブ，21-4
アーカイブ・ファイルのフォーマット，21-5
非 PCM ロック，7-5
増分処理，4-5
手動アーカイブ，21-3，21-4
REDO ログ・ファイルの削除，14-8
障害
ALERT ファイル，6-2
MPP ノード，3-7
インスタンスの回復，22-4
ノード，1-19，22-2
ファイルへのアクセス，22-4
メディア，22-1，22-7，A-22
初期化パラメータ
LM 容量の計画，16-2，16-6
MAX_DEFAULT_PROPAGATION_DELAY，18-10
PARALLEL_DEFAULT_MAXSCANS，18-10
PCM ロックの制御，9-12
アーカイブ，21-3
値の表示，18-15，18-19
ガイドライン，5-8
グローバル定数，6-1，18-8
重複値，18-7
すべてのインスタンスで同じ値，18-10
デフォルト値の使用，18-7
統合分散ロック・マネージャ，18-11
廃止，A-21
ブロックの制御，9-12
親和性
ディスク，4-9，18-21
認識，21-13
パラレル・プロセッサ，A-15
持続リソース，8-10
自動アーカイブ，21-3
自動回復，21-7
順次処理，1-2，1-4
順序
キャッシュされない，6-7，B-3
タイムスタンプ，6-7
データ・ディクショナリ・キャッシュ，4-8，6-7
ログ順序番号，21-5，21-6
順序番号ジェネレータ
LM ロック，4-8
アプリケーションの拡張性，2-6
競合，2-9
順序番号のスキップ，6-7
制限，6-7，B-3
パラレル・サーバー，6-6
分散ロック，6-6
ジョブ
インスタンスの親和性，4-10

す

垂直方向の分割，2-10
水平方向の分割，2-12
スタンバイ・データベース，18-30，22-15
ストアド・プロシージャ，7-8
スピードアップ
定義，1-6
バッチ処理，1-12
スピードダウン，1-8，1-12
スレッド
REDO ログ・ファイルのアーカイブ，21-4，21-10
アーカイブ・ファイルのフォーマット，21-5
インスタンスに関連付けられた，14-7
オープン，21-7，21-15
強制されたログ・スイッチ，21-10
クローズ状態，21-15
グループの数，6-4
使用可能，21-7，21-15，22-9
使用禁止，14-8
単一，4-9
排他モード，18-14
パブリック，14-7，14-8
例，6-3
ログ履歴，21-7

せ

制御ファイル，6-1
　MAXLOGHISTORY，6-5
　アクセス可能性，5-8
　共有，5-4，18-7
　作成，14-10
　データ・ファイル，17-11
　バックアップ，21-1
　パラメータ値，18-8
　複数インスタンスへの変換，23-6
　ログ履歴，14-4，21-6
制限
　REDO ログの変更，14-9
　オフライン表領域，6-8，A-18
　回復，A-22
　キャッシュされた順序，6-7
　遅延ロールバック・セグメント，A-19
　ファイル操作，A-17，A-19，B-2
成長、余地，15-10
静的ハッシング
　ロック・マスター・スキーム，8-9
セグメント
　ID 番号，14-5，14-6
　サイズ，14-7
　定義，11-3
　名前，14-6
　ヘッダー，7-9
　ヘッダー、競合，11-6
　ヘッダー競合，11-6，19-5
　ヘッダー・ブロック，11-14，14-7
　ロールバック・セグメント，6-8
セッション
　PCM ロックの変換の待機，15-17
　複数，18-7，18-19，18-24
接続
　クライアントおよびサービス，18-26
　複数ノード間でのロード・バランシング，18-29
接続時フェイルオーバー，18-29
接続文字列，18-16，18-18，18-19
絶対ファイル番号，6-3

そ

相対ファイル番号，6-3
挿入
　空きリスト，11-14，18-15
　使用不能な空き領域，17-11
　同時，4-8，11-14，17-6
　パフォーマンス，11-12
ソートの強化，A-10
ソート領域，A-10
疎結合システム
　キャッシュの一貫性，3-8
　緊密結合ノード，3-5
　ディスク・アクセス，3-3
　特徴，3-5
　ハードウェア・アーキテクチャ，3-2，3-6
素数，A-7
増分的な成長，17-12

た

帯域幅，1-11，2-2，2-12，3-3，3-4，3-5，3-6
待機時間，1-4，1-9
対称マルチプロセッサ，2-5，3-2，3-4
　構成，1-14
　疎結合システム，3-5
　パラレル処理，1-5
高い可用性
　パラレル・データベースの利点，1-13
多重 REDO ログ・ファイル，6-3
　ファイルの合計数，14-3
　例，6-4
　ログ履歴，21-7
単一インスタンス・データベース，1-15
単一共有モード，4-2，10-5
大規模パラレル処理システム，1-5，11-15
　アプリケーション・プロファイル，3-2
　ディスク・アクセス，3-3
　ハードウェア・アーキテクチャ，3-7

ち

チェックポイント
　強制，21-9
遅延ロールバック・セグメント，A-19
チャネル
　バックアップおよび回復のための割当て，21-8
調整，19-2

て

ディクショナリ・キャッシュ，4-14，5-5，6-6

ロック，10-5
ディクショナリ・キャッシュ・ロック，10-5
ディスク
I/O 統計，19-2，19-6
アクセス，3-2，3-3
競合，6-2，6-9
親和性，4-9，18-21
ブロックの書込み，4-6，9-8
ブロックの読込み，4-6
メディア障害，22-1
ロールバック・セグメント，6-9
データ依存ルーティング，3-8，12-7，19-6
データ・ウェアハウジング，2-7
データ・ディクショナリ，5-5
オブジェクト，6-6
行キャッシュ，6-6
順序キャッシュ，6-7
ビュー，14-5
ビューの問合せ，20-10
データの分割，11-12
PCM ロック，11-16，15-5，17-10
空きリスト，11-2，11-16，17-10，18-15
索引データ，15-5
データ・ファイル，6-2，17-12
表データ，11-11，11-16，15-5，17-10
ロールバック・セグメント，6-8，6-9
データ・ファイル
1 つの表につき複数のファイル，11-16，17-10，17-12
PCM ロック用に指定されていない，9-6
アクセス可能性，5-8
インスタンスの回復，6-2，22-4
エクステントの割当て，17-11
オンラインまたはオフライン，A-18
回復，22-7，A-22
改名，A-18
共有，5-4，6-2
最大数，B-3
削除，A-18
妥当性，15-12
追加，15-6，15-9，15-12，A-18
ディスクの競合，6-2
バックアップ，21-1
パラレル回復，22-7
表領域，A-18
表領域名，15-3
ファイル ID，15-3
ブロックの数，15-3
ブロックへのロックのマッピング，9-6
データ・ブロック，5-5
データベース
NOMOUNT の起動，22-11
アーカイブ・モード，14-2，14-9
アーカイブ・ログ・ファイル数，6-5，21-6
インスタンスの数，11-15，14-3
エクスポートとインポート，B-1
拡張性，2-5
クローズ，18-24
作成，14-2
スタンバイ，18-30，22-15
設計，1-14，13-1
ディスマウント，18-24
バックアップ，4-7，21-1
複数インスタンスへの移行，23-3
複数インスタンスへの変換，23-1，23-4
マウントされているがオープンされていない，14-8
ロールバック・セグメント，14-4
データベース・オブジェクトの削除
表領域，15-6，A-18
データベース管理者（DBA）
分散データベース，1-19
デッドロックの検出，8-9
デフォルト・インスタンス，18-16，18-18

と

統計
調整，19-2
統合運用，1-5
統合分散ロック・マネージャ
LCKn プロセス，9-8
インスタンス・アーキテクチャ，5-2
回復フェーズ，22-5
機能，8-8
キュー，8-2
グループ・ベースのロック，8-10
構成，16-3
最小限の使用，1-16
使用を最小限に抑える，4-13
定義，1-10
内部化，A-7
非 PCM ロックの容量，7-6
フェイルオーバー要件，16-2
フォールト・トレラント，8-8

分散アーキテクチャ，8-8
リソースの共有，9-8
ロックおよびリソースのための容量，16-2
ロックのマスター化，8-9
ロック要求の管理，8-3
登録
インスタンス，18-26
特殊サーバー，1-5
トランザクション
空きリスト，11-4
異常終了，6-8
インスタンス障害，22-3
オフライン表領域，6-9，A-19
回復，A-12
回復を待つ，22-3
切離し，4-16
行ロック，4-7，4-16
更新，4-7，11-2
コミットされたデータ，4-7，21-9
順序番号，6-6
挿入，4-8，11-2
同時，4-7，4-13，5-4
ロールバック，6-8，22-3
ロールバック・セグメント，6-9，A-19
ロック，4-14，7-2，7-3，7-5，7-6，10-3
トランザクション処理モニター，12-7
トレース・ファイル
複数インスタンスへの変換，23-4
同期
オーバーヘッド，1-9
コスト，1-10，1-17，2-9，2-11，2-12
非 PCM，4-17
同時実行性
インスタンスの最大数，11-15，14-3
共有モード，4-2
順序，6-7
挿入および更新，11-14，17-6
トランザクション，5-4，11-2
動的パフォーマンス・ビュー
作成，20-10

に

二重ポート・コントローラ，2-3
認証
パスワード・ファイル，18-23

ね

ネットワークの使用率，19-2

の

ノード
キャッシュの一貫性，4-13
障害，1-13，1-19，22-2
親和性の認識，21-13
単一ノードからクラスタへ，23-2
追加，17-12，23-2
定義，1-2
パラレル停止，18-24
パラレル・バックアップ，21-11
リモート，18-16，18-18
ローカル，18-6，18-8
ノード間通信，1-9

は

ハードウェア
アーキテクチャ，3-1
拡張性，2-2
要件，3-3
廃止されたパラメータ，A-20，A-21
排他的 PCM ロック，4-16
排他モード，8-7，18-11
MAXLOGHISTORY，21-7
アーカイブ・ログ・モードの切替え，14-8
空きリスト，17-6，17-11
インスタンス番号の指定，17-12
起動，17-12，18-11
互換性，B-2
スレッド番号の指定，18-14
データベース・アクセス，1-15，4-2
表領域のオフライン化，A-18
ファイル操作に必要，A-17，B-2
メディア回復，14-4
ハッシュ PCM ロック，9-4，9-16，9-20，9-21
解放可能，9-4，15-7，15-8
概要，7-5
作成，9-4
指定，15-8
ハッシュ・クラスタ，17-7
空きリスト，11-11，A-20
ハッシング

静的、ロック・マスター・スキーム，8-9
半共有排他モード，8-7
半共有モード，8-7
バージョン、Oracle
アップグレード，A-1
互換性，17-11，A-17
バージョン間の互換性
同じクラスタ，18-20
バックアップ，21-1
アーカイブ・ログ，21-7
エクスポート，B-1
オフライン，4-7，21-11
オンライン，4-7，21-11，21-15
回復に使用するファイル，22-7
パラレル，21-11
複数インスタンスへの変換，23-3
バックグラウンド・プロセス，5-5
ARCH，21-4
LGWR，21-6
SMON，18-24，22-2
インスタンス，5-4
インスタンス・ロックの保持，7-4，7-6
オプション，5-5
パラレル・キャッシュ管理，4-14
バッチ・アプリケーション，1-12
バッファ、REDO ログ，5-5
バッファ・キャッシュ，5-5
I/O 操作の最少化，4-6，4-13
一貫性，4-13
インスタンスの回復，22-3
ディスクへの書込み，4-6
分散ロック，2-7
バッファ状態，9-10
パラレル・バックアップ，21-11
番号ジェネレータ，6-6
パフォーマンス
Oracle8 機能，4-6
アプリケーション，11-12
順序のキャッシュ，6-7
順序番号，6-7
挿入および更新，11-12
パラレル・データベースの利点，1-13
ファイン・グレイン・ロック，9-19
リソース共有システム，1-14
ロールバック・セグメント，6-8，6-9
ロックのマスター化，8-9
パブリック・ロールバック・セグメント
オンライン化，14-6
共通パラメータ・ファイル，18-5
作成，14-5
指定，14-5
所有者，14-5
デフォルトによる使用，14-5
パラメータ
PCM ロックの制御，9-12
記憶領域，6-8，17-6，17-8，17-11
データベース作成，11-15，14-3，14-4
廃止，A-21
パラメータ・ファイル，18-3
PFILE，18-6，18-8，23-5
共通ファイル，18-5，18-6，23-7
重複値，18-7
同一パラメータ，18-7
バックアップ，21-1
複数インスタンスへの変換，23-6
不適切な NFS アクセス，23-9
リモート・インスタンス，18-6，18-8，18-18
パラレル回復，22-7，22-14，22-15
パラレル・キャッシュ管理，4-13
パラレル・キャッシュ管理ロック
NULL，4-16
インスタンス LCK プロセスが所有する，9-5
回数の最小化，12-1
解放，4-16
解放可能なハッシュ，9-4，15-7，15-8
機能，9-3
周期性，9-8
取得，4-16
実現，7-5
順序，4-8
相対数，7-6
定義，7-5，9-2
排他的，4-16
複数のインスタンスが所有するロック，9-6
変換，9-5
ユーザー制御，7-6
読込み，4-16
パラレル・サーバー，A-4
Oradebug，A-4
インストール，A-4
起動と停止，18-13，18-24
ジョブとインスタンスの親和性，A-4
定義，1-6
データベース構成，A-4

リスナーのロード・バランシング，A-3
パラレル処理
MPP，1-5
Oracle 構成，1-14
SMP，1-5
誤解，2-12
作業負荷のタイプ，1-12
実現，3-2
統合運用，1-5
特徴，1-4
ハードウェア・アーキテクチャ，3-2
メッセージ機能，1-11
有効でない状況，2-9
有効な状況，2-7
要素，1-6
利点，1-11
パラレル実行
Oracle Parallel Server，1-21
インスタンスの制限，18-25
オーバーヘッドの計算，16-4
拡張性，12-3
実行処理，1-2，1-21，2-5
スピードアップおよびスケールアップ，1-12
プロセッサのアフィニティ（親和性），A-15
ロード・バランシング，18-30
パラレル・データベース
可用性，1-13
パラレル実行，1-21
利点，1-13
パラレル・データベースの柔軟性，1-13
パラレル・トランザクション回復
8.1 での変更，A-3
パラレル・プロセッサの親和性，A-15
パラレル・モード
回復制限，A-22
起動，18-8
順序についての制限，6-7，B-3
停止，18-24
ファイル操作の制限，A-17，A-19，B-2

ひ

非 PCM リソース，16-4
非 PCM ロック
DML ロック，10-3
IDLM 容量，7-6
エンキュー，7-3
概要，10-1
システム変更番号，10-4
相対数，7-6
タイプ，7-5
ディクショナリ・キャッシュ・ロック，10-5
トランザクション・ロック，10-3
表ロック，10-3
マウント・ロック，10-5
ユーザー制御，7-6
ライブラリ・キャッシュ・ロック，10-4
非共有システム
拡張性，2-3
概要，3-6
共有ディスク・システムと，3-9
短所，3-8
大規模パラレル処理システム，3-7
ディスク・アクセス，3-2
利点，3-8
非固定モード、ロック要素，9-19
非対称マルチプロセッシング，2-5
非同期トラップ，8-2，8-3，8-4
表
PCM ロック，11-16，17-10
アクセス・パターン，12-2
エクステントの割当て，11-11，17-13
競合，6-8，17-12
クラスタ，17-7
初期記憶領域，11-17，17-12
使用不能な空き領域，17-11
表領域，6-8
複数ファイル，11-11，17-12
読込み専用，12-2
ロック，7-3，7-6
表領域
SYSTEM，14-4
アクティブ・ロールバック・セグメント，6-8
オフライン，6-8
オフライン化，6-8，A-18，A-19
オンライン・ロールバック・セグメント，14-5，14-7
回復，22-7，A-22
索引データ，15-5
削除，15-6，A-18
作成，15-6，A-18
データ・ファイル，A-18
バックアップ，4-7，21-1
パラレル回復，22-7

パラレル・バックアップ，21-11
表，6-8
読込み専用，15-11
ロールバック・セグメント，6-8，14-4，14-5，14-7
表領域の作成，A-18
表ロック，10-3
最少化，16-6
使用禁止にする，16-7
ビュー
グローバル，18-22，20-10
ロールバック・セグメント，14-5

ふ

ファースト・スタート・パラレル・ロールバック
パラレル・ロールバック、ファースト・スタート，22-15
ファースト・スタート・ロールバック，A-3
ファイル
ALERT，6-2，22-4
PFILE，18-6，18-8
REDO ログ，4-7，6-3，21-3，21-5，21-6，A-18
REDO ログのアーカイブ，4-7，21-3，21-4，21-5
エクステントの割当て，17-11
エクスポート，B-1
回復での使用，22-7
改名，14-9，21-10，A-18
共通パラメータ・ファイル，18-6
限定的運用，21-10，A-17
サイズ，9-6
最大数，B-3
削除，14-8，14-9，21-10，A-18
制御ファイル，6-1，6-5，21-6
多重，21-7
追加，A-18
データ・ファイル，6-1
番号、絶対，6-3
番号、相対，6-3
パラメータ，18-3，18-7
ファイルの改名
REDO ログ・ファイル，A-18
RENAME FILE オプション，A-18
ログ・スイッチ，21-10
ファイルの追加，14-9，15-9，A-18
ファイン・グレイン・ロック，2-8，9-4，9-6，9-15，9-17，9-18，9-19，9-20，A-9
1 ロック要素あたり 1 ブロック，9-16
DBA ロック，9-16
概要，7-5
グループ係数，15-8
作成，9-3
指定，15-8
フェイルオーバー，4-12，16-2，A-11
接続時，18-29
フォアグラウンド・プロセス
インスタンス停止，18-24
フォールト・トレラント，8-8
複数インスタンス・データベース
アプリケーションの変換，23-3
定義，1-15
変換しない理由，23-2
変換する理由，23-2
要件，23-3
複数共有モード，4-2，10-5
複数バージョン読込みの一貫性，4-7
部門別処理アプリケーション，2-8
ブロッキング AST，8-3
ブロック
空き領域，11-2
インスタンスとの関連付け，11-11
インスタンスに関連付けられた，22-3
インスタンス・ロック，4-14
競合，6-9，11-16，15-5，17-10，17-12
クリーンアウト，A-14
セグメント・ヘッダー，11-14
遅延ロールバック，6-9，A-19
ディスクに書き込まれるとき，9-8
ディスクに書き込むタイミング，4-6
動的割当て，17-14
複数コピー，4-6，4-13，5-5
分散ロック，5-5
ブロック・サーバー・プロセス
定義，1-16，5-6
ブロックの動的割当て，11-17
分割
アプリケーション，12-6
ガイドライン，2-10
垂直方向，2-10
水平方向，2-12
データ，12-7
要素，2-10
分割データ
アプリケーション，2-7
データ・ファイル，6-2

を処理するアプリケーション，1-17
分散ロック
行キャッシュ，6-6
合計数，6-6
順序，6-6
メモリー領域，5-5
ロールバック・セグメント，6-9
分散ロック・マネージャ，A-7
プライベート・スレッド，14-8
プライベート・ロールバック・セグメント
個別パラメータ・ファイル，18-5
作成，14-5
指定，6-9
取得，6-8
プロセス空きリスト
セグメント・ヘッダーの ping，11-6
定義，11-5
プロセッサの親和性
パラレル実行，A-15
プロテクト書込みモード，8-7

へ

ヘッダー
セグメント，11-14，14-7
ロールバック・セグメント，14-7
変換
アプリケーション，23-3
影響，23-3
データベース，23-4
データベース、複数インスタンス，23-1
変換キュー，8-3
変更データ
インスタンスの回復，22-3
更新，5-5

ほ

ホスト、IDLM，9-8

ま

マウント・ロック，7-2，10-5
マスター空きリスト，11-6
マスター化、ロック，8-9，19-8
マスター・ノード，8-9
待ち時間，1-11，1-13，2-2，2-12，3-3
待ち、セッション，15-17
マルチスレッド・サーバー，8-10
接続ロード・バランシング，18-29

み

未使用領域の割当て解除，17-15

め

メッセージ
ALERT ファイル，6-2，22-4
インスタンス停止，18-24
パラメータ値，6-1
ファイルへのアクセス，6-2，22-4
分散ロック・マネージャ，9-8
メッセージ機能
パラレル処理，1-11
メディア回復，22-7
O/S ユーティリティ，22-8
不完全，22-8
ログ履歴，14-4，21-7，22-9
メディア障害，22-1，22-7，A-22
回復，A-22
自動回復，21-7
ファイルへのアクセス，6-2
メディア・マネージャ
バックアップ作成の要件，21-8
メモリー
IDLM 要件，8-10
SGA，6-7
キャッシュ，4-6
キャッシュされたデータ，4-6
分散ロック，5-5
メンバー
MAXLOGMEMBERS，14-3

も

モード
PCM ロック，4-16
アーカイブ，4-7，14-2，14-8，21-3
データベース・アクセス，4-2，18-11，18-14
非互換，8-2
ロックの互換性，9-11
ロック要素，9-19
モジュロ，11-13，11-15，17-10，17-11

ゆ

有効ビット、ロック要素，9-19
ユーザー
　PUBLIC，14-5，14-6
　SYS，14-6
　名前とパスワード，18-18
　パラレル・データベースの利点，1-13
　複数，5-4
ユーザー・プロセス
　空きリスト，11-2，11-15，17-8
　インスタンス停止エラー，18-24
　手動アーカイブ，21-4
ユーザー・モード IPC
　キャッシュ・フュージョン，20-6
ユーザー・レベルの IDLM，8-10
ユーティリティ、Oracle
　Export、Import，B-1

よ

容量計画，19-3
読込み一貫性
　ロールバック情報，6-8
読込み専用アクセス，4-7，4-16
　アプリケーション，2-7
　索引データ，15-5
　読込み PCM ロック，4-16
読込み専用表，12-2
読込みの一貫性
　複数バージョン，4-7
読込みロック・モード，4-16

ら

ライブラリ・キャッシュ・ロック，10-4
ラッチ，7-3，10-5
ランダム・アクセス，2-8

り

リスナー
　インスタンス登録，18-28
　ロード・バランシング，A-3
リソース
　オペレーティング・システム，18-8
　解放，22-3
　持続，8-10
　データベース，4-13
リソース共有システム，17-12
リモート I/O，2-3
リモート・インスタンス，18-6，18-8，18-18
リモート・データベース，1-18
領域
　SGA，18-19
　空きブロック，11-2，11-17
　空きブロックのソース，11-6
　空きリスト，11-2
　インスタンスに割り当てられていない，11-6，17-11
　エクステントの割当て，17-12
　排他モードで使用不能，17-11
　未使用領域の判別，17-15
　未使用領域の割当て解除，17-15
履歴、アーカイブ，21-6，22-9
リンク，5-8

る

ルーティング、データ依存，12-7，19-6

れ

レプリケート・システム，23-1

ろ

ローカル I/O，2-3
ローカル・インスタンス
　ノード，18-18
ローカル・ロック，7-2
ロー・デバイス，23-4
ロード・バランシング，12-5，A-13
　パラレル実行，18-30
　複数ノード間での接続，18-29
　複数のリスナー間でのクライアント接続，18-29
ロールバック
　インスタンスの回復，22-3
　行ロック，4-16
　ロールバック・セグメント，6-8
ロールバック・セグメント，14-4
　ID 番号，14-5，14-6
　SYSTEM，6-8
　オンライン，6-8，14-6
　オンライン化，A-13

競合，6-8，6-9，14-4
グローバル定数パラメータ，6-9
指定，14-5
説明，6-8
遅延，6-9，A-19
名前，14-5，14-6
パブリック，14-6
パブリックとプライベート，6-9，14-6，18-5
表領域，6-8，14-4，14-5，14-7，23-9
複数，6-8，14-4，18-14
分散ロック，6-9
プライベート，23-9
ロールバック・セグメントの作成，14-4，14-5
ロールバック・セグメントの取得，14-5
初期化パラメータ，6-9
ログ順序番号，21-5，21-6
ログ・スイッチ，6-5
強制，21-10，21-15，A-19
クローズ状態のスレッド，21-10
グローバル，21-15
ファイルの追加または削除，14-9
ログ履歴，21-6
ログ・ファイル
REDO ログ・ファイル，21-1
アクセス可能性，5-8
ログ履歴，14-4，21-6，22-9
ロック
DML，7-3，10-3
IDLM による所有権，8-10
OPS 共有モード，7-2
OPS 排他モード，7-2
PCM ロック，9-3，11-16，17-10
インスタンス，7-2，7-4
インプリメント方式、PCM，7-5
エンキュー，7-3
境界，11-18
行，4-16，10-3
行キャッシュ，6-6
グループ・ベース，8-9，8-10
グローバル，18-8
コスト，7-7
識別子，7-8
システム変更番号，10-4
タイプ，7-2
ディクショナリ・キャッシュ，10-5
トランザクション，4-14，7-2，10-3
名前の形式，7-8
ハッシュ，7-5
非 PCM，7-3，7-5
表，7-3，7-5，7-6，10-3
ファイン・グレイン，7-5
プロセス所有，8-10
変換，8-4
マウント・ロック，7-2，10-5
マスター化，8-9，19-8
モードおよびバッファ状態，9-10
モードの互換性，9-11
要求、IDLM による管理，8-3
ライブラリ・キャッシュ，10-4
ラッチ，7-3
ローカル，7-2
ロールバック・セグメント，6-9
ロック・メカニズムの概要，7-1
ロック・オペレーション
キャッシュ・フュージョン，20-6
ロック値ブロック，10-4
ロックの競合
位置の特定，19-4
検出，19-3
ロック変換
過剰な率，19-7
検出，19-3
ロック要素，7-9
LRU リスト，9-19
空き，9-19
作成，9-19
名前，9-15
番号，9-22
非固定モード，9-19
モード，9-19
有効ビット，9-19
ロックへの対応，9-14
論理データベース，1-18

わ

割当て
PCM ロック，9-6，9-24，11-16，17-10
空き領域，11-6，11-11
エクステント，17-13，18-15
自動，11-17，17-12，17-13
順序番号，6-6
チャネル，21-8
ロールバック・セグメント，14-5