Oracle8*i* Parallel Server

管理、配置およびパフォーマンス

リリース 8.1

2000 年 2 月

部品番号 : J00957-01

ORACLE®

Oracle8*i* Parallel Server 管理、配置およびパフォーマンス , リリース 8.1

部品番号 : J00957-01

原本名：Oracle8*i* Parallel Server Administration, Deployment, and Performance, Release2 (8.1.6)

原本部品番号：A76970-01

原本著者：Mark Bauer

原本協力者：Cathy Baird、David Austin、Wilson Chan、Michael Zoll、Christina Anonuevo、Lance Ashdown、Bill Bridge、Sandra Cheever、Annie Chen、Carol Colrain、Mark Coyle、Sohan Demel、Connie Dialeris、Karl Dias、Anurag Gupta、Deepak Gupta、Mike Hartstein、Andrew Holdsworth、Merrill Holt、Ken Jacobs、Ashok Joshi、Jonathan Klein、Jan Klokkers、Boris Klots、Anjo Kolk、Tirthankar Lahiri、Bill Lee、Lefty Leverenz、Juan Loaiza、Sajjad Masud、Neil Macnaughton、Ravi Mirchandaney、Rita Moran、Kotaro Ono、Kant Patel、Erik Peterson、Mark Porter、Darryl Presley、Brian Quigley、Ann Rhee、Pat Ritto、Roger Sanders、Hari Sankar、Ekrem Soylemez、Vinay Srihari、Bob Thome、Alex Tsukerman、Tak Wang、Graham Wood、Betty Wu

目次

4 インスタンスの管理

第 III 部　Oracle Parallel Server の設計および配置

5　アプリケーションの分析およびパーティション化

6　データベースの設計方法

7 PCM および非 PCM インスタンス・ロックの使用計画

8 空きリスト・グループによるデータのパーティション化

9 インスタンス・ロックの設定

10 ロックおよびリソースに対する DLM 容量の確保

第 IV 部　Oracle Parallel Server のパフォーマンスの監視およびチューニング

11　一般的なチューニングの推奨事項

12 Oracle Parallel Server およびインスタンス間パフォーマンスのチューニング

第 V 部　Oracle Parallel Server のメンテナンス

13　データベースのバックアップ

14 データベースのリカバリ

第 VI 部　Oracle Parallel Server の参照情報

A　Parallel Server 用のデータベース設計の事例

索引

はじめに

このマニュアルでは、Oracle Parallel Server の管理および配置について説明します。『Oracle8*i* Parallel Server セットアップおよび構成ガイド』の手順が完了してから、このマニュアルを読んでください。また、『Oracle8*i* Parallel Server 概要』も読んでおいてください。

このマニュアルでは、Oracle Parallel Server のインストール後の管理手順に続いて、パラレル処理の実装について説明します。また、アプリケーション開発と配置方法およびチューニングについても説明します。このマニュアルの情報は、すべてのオペレーティング・システム上で動作する Oracle Parallel Server に適用されます。このマニュアルでは、必要に応じてプラットフォーム固有のドキュメントを参照します。

Oracle8*i* での新規事項

このマニュアルは、Oracle8*i* 用に改訂されています。Oracle8*i* では、キャッシュ・フュージョンが導入されています。この機能によって、インスタンス間の競合によって発生する読込み / 書込みの競合を解消する際のオーバーヘッドが削減されます。その結果、Oracle Parallel Server のスケーラビリティのみでなく、パフォーマンスも大幅に向上します。

> **参照：** Oracle Parallel Server のリリース間の機能変更については、『Oracle8*i* Parallel Server 概要』を参照してください。

リリース 8.1.5

リリース 8.1.5 では、キャッシュ・フュージョンの第 1 フェーズが導入されています。

リリース 8.1.6

リリース 8.1.6 では、プライマリ / セカンダリ・インスタンス機能と同様、キャッシュ・フュージョンへのさらなる拡張が導入されています。また、新しいパフォーマンス統計情報も追加されています。

対象読者

このマニュアルは、Oracle Parallel Server を操作する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としています。

このマニュアルの構成

このマニュアルでは、Oracle Parallel Server の管理、配置およびパフォーマンスについて 5 部構成で説明しています。まず、Oracle Parallel Server の基本管理について説明します。次に、Parallel Server のアプリケーション、データベース設計および配置について説明します。最後に、バックアップやリカバリなどのメンテナンス項目の他、パフォーマンスのチューニングについても説明します。

構成内容

このマニュアルは、次の 5 部で構成されています。

第 I 部「Oracle Parallel Server パラメータ・ファイルの管理」

第 1 章「パラメータ・ファイルおよび Oracle Parallel Server 固有のパラメータ」	この章では、パラメータ・ファイルおよび Oracle Parallel Server 固有のパラメータについて説明します。

第 II 部「Oracle Parallel Server の管理」

第 2 章「Oracle Parallel Server 環境のパラレル実行」	この章では、Oracle Parallel Server 環境でのパラレル実行について説明します。
第 3 章「Oracle Parallel Server データベース作成 に関する問題」	この章では、Oracle Parallel Server のデータベース作成について説明します。この章に記載する情報は、『Oracle8*i* Parallel Server セットアップおよび構成ガイド』の補足情報です。
第 4 章「インスタンスの管理」	この章では、インスタンスの基本的な管理方法について説明します。

第 III 部「Oracle Parallel Server の設計および配置」

第 5 章「アプリケーションの分析およびパーティション化」	この章では、Oracle Parallel Server 環境での分析およびパーティション化について説明します。
第 6 章「データベースの設計方法」	この章では、ブロックおよびエクステント操作、競合の削減、ロック方法などの、Oracle Parallel Server のデータベース設計について説明します。
第 7 章「PCM および非 PCM インスタンス・ロックの使用計画」	この章では、PCM および非 PCM ロック・リソースの両方の使用および管理方法について説明します。
第 8 章「空きリスト・グループによるデータの パーティション化」	この章では、パーティション・データに空きリスト・グループを使用して、パフォーマンスを向上する方法について説明します。
第 9 章「インスタンス・ロックの設定」	この章では、Oracle Parallel Server 環境でのインスタンス・ロックの設定方法について説明します。
第 10 章「ロックおよびリソースに対する DLM 容量の確保」	この章では、パフォーマンスを最適化するための、ロックおよびリソースの使用計画および管理方法について説明します。

第 IV 部「Oracle Parallel Server のパフォーマンスの監視およびチューニング」

第 11 章「一般的なチューニングの推奨事項」	この章では、一般的なチューニングに関する推奨事項を提示します。
第 12 章「Oracle Parallel Server およびインスタンス間パフォーマンスのチューニング」	この章では、インスタンス間のパフォーマンスの監視およびチューニング方法について説明します。

第 V 部「Oracle Parallel Server のメンテナンス」

第 13 章「データベースのバックアップ」	この章では、Oracle Parallel Server データベースのバックアップ手順について説明します。
第 14 章「データベースのリカバリ」	この章では、Oracle Parallel Server データベースのリカバリ手順について説明します。

関連マニュアル

このマニュアルを読む前に、『Oracle8*i* Parallel Server 概要』および『Oracle8*i* Parallel Server セットアップおよび構成ガイド』を読んでください。

詳細は、次のマニュアルを参照してください。

インストレーション・ガイド

- 『Oracle8*i* for Sun SPARC Solaris インストレーション・ガイド』
- 『Oracle8*i* for HP 9000 Servers and Workstations インストレーション・ガイド』
- 『Oracle8*i* for IBM AIX インストレーション・ガイド』
- 『Oracle8*i* for Windows NT インストレーション・ガイド』
- 『Oracle Diagnostics Pack インストレーション・ガイド』

オペレーティング・システム固有の管理ガイド

- 『Oracle8*i* for Sun SPARC Solaris 管理者リファレンス』
- 『Oracle8*i* for HP 9000 Servers and Workstations 管理者リファレンス』
- 『Oracle8*i* for IBM AIX 管理者リファレンス』
- 『Oracle Parallel Server for Windows NT 管理者ガイド』
- 『Oracle8*i* for Windows NT 管理者ガイド』

Oracle Parallel Server Management

- 『Oracle Enterprise Manager 管理者ガイド』
- 『Oracle Diagnostics Pack スタート・ガイド』

Oracle Server マニュアル

- 『Oracle8*i* 概要』
- 『Oracle8*i* 管理者ガイド』
- 『Oracle8*i* リファレンス・マニュアル』
- 『Oracle8*i* Net8 管理者ガイド』

表記規則

この項では、このマニュアルで使用する次のものに関する表記規則について説明します。

- 本文
- コード例

本文

この項では、本文中で使用される表記規則について説明します。

大文字

大文字のテキストは、コマンド、キーワード、オブジェクト名、パラメータ、ファイル名などに対して注意を促すために使用されます。

たとえば、「プライベート・ロールバック・セグメントを作成する場合、その名前はパラメータ・ファイルの ROLLBACK_SEGMENTS に含まれる必要があります。」のように使用されます。

コード例

SQL および SQL*Plus のコマンドおよび文は、本文と区別して、固定幅フォントで示されます。たとえば次のとおりです。

```
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る場合があります。例文中のすべての句読点は必要です。例文は、すべてセミコロン（;）で終了します。アプリケーションによって、文を終了するためのセミコロンまたはその他の終了記号が、必要な場合と必要でない場合があります。

例文中の大文字語は、Oracle SQL 内のキーワードを示します。ただし、文を発行する場合、キーワードの大 / 小文字は区別されません。

例文中の小文字語は、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、小文字語は、表、列またはファイルの名前を示します。

第I部

Oracle Parallel Server パラメータ・ファイルの管理

第 I 部では、初期化パラメータ・ファイルの管理に関する問題について説明します。また、Oracle Parallel Server 環境の構成後に考慮する必要がある、Oracle Parallel Server 固有のパラメータに関する問題についても説明します。第 I 部に含まれる章は、次のとおりです。

- 第 1 章「パラメータ・ファイルおよび Oracle Parallel Server 固有のパラメータ」

1

パラメータ・ファイルおよび Oracle Parallel Server 固有のパラメータ

この章では、初期化パラメータ・ファイルおよび Oracle Parallel Server 固有のパラメータについて説明します。内容は次のとおりです。

- Oracle Parallel Server のパラメータ・ファイルの管理
- 複数インスタンスに対する初期化パラメータの設定

Oracle Parallel Server 環境でのパラレル実行用の追加パラメータについては、第 2 章で説明します。

Oracle Parallel Server のパラメータ・ファイルの管理

単一インスタンス環境で使用されるパラメータの他に、いくつかの Oracle Parallel Server 固有のパラメータがあります。これらのパラメータには、すべてのインスタンスで同一の値を設定する必要があるものがあります。

これらのパラメータは、1 つ以上のパラメータ・ファイルを使用して設定できます。パラメータ・ファイルは、テキスト・エディタで編集します。Oracle は、これらの設定をパラメータ・ファイルから読み込み、値を制御ファイルに書き込みます。

Oracle Parallel Server にあるパラメータ・ファイルは、次のものを使用したいくつかの方法で実装できます。

- 1 つの共通パラメータ・ファイル
- インスタンス固有のパラメータ・ファイル
- 特定のセッションに対するデフォルト以外のパラメータ・ファイル

パラメータ・ファイルの命名規則

パラメータ・ファイルには、次の命名規則を使用することをお薦めします。

- 共通パラメータ・ファイルには、`init_`*`dbname`*`.ora` という名前を付けます。`DBNAME` は、Oracle Parallel Server データベースの名前です。
- インスタンス固有のすべてのパラメータ・ファイルには、`init_`*`sid`*`.ora` という名前を付けます。`SID` は、インスタンス名または番号です。
- デフォルト以外のすべてのパラメータ・ファイルに名前を付けます。

これらの命名規則を使用すると、Oracle Parallel Server の管理が簡単になります。

1 つの共通パラメータ・ファイル

すべてのデフォルト・パラメータ設定を Oracle Parallel Server に使用する場合、1 つの共通パラメータ・ファイルを共有ディスクに配置します。これによって、1 つのファイルでパラメータ設定をグローバルに管理できるため、管理が簡単になります。クラスタ・システムがファイルを共有しない場合は、共通ファイルを各ノードにコピーします。

> **参照：** `init_`*`dbname`*`.ora` 初期化パラメータ・ファイル・エントリの詳細は、『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。

インスタンス固有のパラメータ・ファイル

構成によっては、パフォーマンスを向上するためにインスタンス固有のパラメータ設定を使用する必要がある場合があります。たとえば、各インスタンスで、異なるサイズのシステム・グローバル・エリア（SGA）を作成できます。これを共有ファイル・システムで行う場合も、すべてのインスタンスに同一の設定を持つ必要があるパラメータに関しては、共通パラメータ・ファイルを使用します。IFILE（インクルード・ファイル）パラメータを設定することによって、インスタンス固有のパラメータ・ファイルの中から共通ファイルを識別することをお薦めします。

注意： Database Configuration Assistant は、この設定をデフォルトで行います。

インスタンス固有のファイルを使用する必要がある条件

次のインスタンスを作成する場合は、インスタンス固有のパラメータ・ファイルを使用する必要があります。

- INSTANCE_NUMBER を指定するインスタンス
- THREAD を指定するインスタンス
- プライベート・ロールバック・セグメントを使用するインスタンス

パブリック・ロールバック・セグメントのみを使用するインスタンスは、共通パラメータ・ファイルを共有できます。

参照： 1-9 ページの「共通パラメータ・ファイルのパラメータ」を参照してください。

インスタンス固有のファイル内にある IFILE パラメータの配置および使用

パラメータ・ファイル内でパラメータ・エントリが重複している場合、そのパラメータに対してファイル内で最後に指定された値が、それより前の値をオーバーライドします。Oracle に正しい共有パラメータ値を確実に使用させるには、IFILE パラメータを、すべてのインスタンス固有のパラメータ・ファイルの最後に配置します。逆に、インスタンス固有のパラメータ設定の前に IFILE パラメータを配置することによって、共通パラメータ値をオーバーライドできます。

注意： Database Configuration Assistant は、IFILE パラメータをパラメータ・ファイルの先頭に配置します。

複数の IFILE の使用

IFILE を 2 回以上パラメータ・ファイルに指定して、複数の共通パラメータ・ファイルを含めることができます。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、インスタンス固有のパラメータ・ファイルは、init_*dbname*.ora ファイル、および LOG_* パラメータと GC_* パラメータに対する異なるパラメータ・ファイルを含む場合があります。たとえば次のとおりです。

```
IFILE=INIT_OPS.ORA
IFILE=INIT_LOG.ORA
IFILE=INIT_GC.ORA
LOG_ARCHIVE_START=FALSE
THREAD=3
ROLLBACK_SEGMENTS=(RB_C1,RB_C2,RB_C3)
```

この例では、LOG_ARCHIVE_START の値は、このパラメータに対して init_*log*.ora に指定されたすべての値をオーバーライドします。これは、IFILE パラメータが LOG_ARCHIVE_START パラメータの前にあるためです。

参照 :

- 1-8 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」を参照してください。
- 4-5 ページの「インスタンスの停止」を参照してください。

特定のセッションに対するデフォルト以外のパラメータ・ファイル

特定のセッションに対しては、STARTUP コマンドの PFILE オプションを使用してデフォルト以外のパラメータ・ファイルを指定します。これは、たとえば、ピーク時以外のバッチ操作のパラレル実行を最適化するために、特定のパラメータ設定を使用する場合に行います。

リモート・ノード上にあるインスタンスに対してパラメータを指定する場合も、PFILE によって指定したパラメータ・ファイルは、ローカル・ノードにアクセス可能なディスク上にある必要があります。デフォルト以外のパラメータ・ファイルを複数指定し、オンデマンドで使用できます。

初期化ファイルの位置

インスタンスを起動したデータベースは、初期化パラメータ・ファイルにアクセスできる必要があります。Oracle Parallel Server は、次の場所にあるデータベース初期化ファイルを使用します。

- Windows NT の場合、*ORACLE_HOME*¥admin¥*db_name*¥pfile
- UNIX の場合、$*ORACLE_HOME*/admin/*db_name*/pfile

Parallel Server 環境での起動プロセスおよびパラメータ

前述したように、Oracle は、ご使用の環境で最初のインスタンスが起動したときに、すべてのパラメータ・ファイルからのパラメータ値を制御ファイルに書き込みます。最初のインスタンスのアラート・ログによって、インスタンスが最初に起動し、データベースをマウントしたことを確認できます。起動プロセスの詳細は、第 4 章を参照してください。

後続のインスタンスに対するパラメータ・ファイルに、すべてのインスタンスに対して同じである必要があるパラメータがあり、このパラメータの値が、制御ファイル内のそのパラメータにすでに設定されている値と一致しない場合、インスタンスはデータベースをマウントできません。

注意： ご使用のアラート・ログの位置を確認するには、検索文字列 `alert*.log` を使用してください。

リモート・ノード上での 2 つのインスタンスの起動

1 つのノード上の SQL*Plus セッションから、複数のノードを起動できます。たとえば、ローカル・ノード上で SQL*Plus セッションを使用して、`init_ops1.ora` および `init_ops2.ora` という個別パラメータ・ファイルを使用する 2 つのインスタンスをリモート・ノード上で起動できます。

データベースに接続する前に、SQL*Plus で、次のように入力して最初のインスタンスにコマンドを発行します。

```
SET INSTANCE OPS1;
```

次のように入力して、最初のインスタンスに接続し、そのインスタンスを起動した後、切断します。

```
CONNECT INTERNAL;
STARTUP PFILE=INIT_OPS1.ORA;
DISCONNECT;
```

次のように入力して、2 つ目のインスタンスにコマンドを再発行します。

```
SET INSTANCE OPS2;
```

次のように入力して、2 つ目のインスタンスに接続し、起動します。

```
CONNECT INTERNAL;
STARTUP PFILE=INIT_OPS2.ORA;
```

ここでは、OPS1 および OPS2 は、2 つのインスタンスの Net8 ネット・サービス名です。これらのネット・サービス名は、`TNSNAMES.ORA` で定義します。

2 つの個別パラメータ・ファイルには、IFILE パラメータを使用して、init_*dbname*.ora ファイルからのパラメータ値を含めることができます。

インスタンス番号および起動順序

Oracle Parallel Server を使用可能または使用禁止にしてインスタンスを起動するときに、初期化パラメータ INSTANCE_NUMBER を使用することによって、インスタンス番号を明示的に指定できます。INSTANCE_NUMBER は、THREAD_ID の値と同じ値になるように設定してください。インスタンス番号を指定しない場合、インスタンスは、使用可能な最も小さい番号を自動的に取得します。

挿入および更新のために、インスタンスへエクステントを一貫して割り当てる必要がある場合は、必ず INSTANCE_NUMBER パラメータを使用してください。これによって、インスタンス間のデータのパーティション化を保持できます。INSTANCE_NUMBER パラメータを使用する場合は、各インスタンスに一意のインスタンス番号を指定する必要があります。

インスタンスは、起動時にインスタンス番号を取得します。この番号によって、インスタンスは、FREELIST GROUPS 記憶域オプションによって作成された各表に対する空きリスト・グループの 1 つにマップされます。

デフォルトでの起動順序によるインスタンス番号の決定

INSTANCE_NUMBER に値を指定しないと、起動順序によってインスタンス番号が決定されます。インスタンスがパラレルで起動する場合、デフォルトの起動番号の制御は困難です。さらに、インスタンスを停止および再起動すると、インスタンス番号が変わる場合があります。次の SQL*Plus コマンドによって、各インスタンスの現在の番号を表示できます。

```
SHOW PARAMETER INSTANCE_NUMBER
```

Oracle が、起動順序に基づいてインスタンス番号を割り当てている場合、このコマンドは NULL を表示します。

インスタンスを停止した後、そのインスタンスのインスタンス番号は再使用可能です。1 つ目のインスタンスが再起動される前に 2 つ目のインスタンスが起動されると、2 つ目のインスタンスは、1 つ目のインスタンス番号が使用していたインスタンス番号を取得できます。

起動順序に基づくインスタンス番号は、INSTANCE_NUMBER パラメータによって指定されるインスタンス番号に依存しません。INSTANCE_NUMBER を使用するかしないかにかかわらず、インスタンスがいずれかの方法でインスタンス番号を取得した後、他のインスタンスが他の方法で同じ番号を取得することはできません。すべてのインスタンス番号は、取得された方法にかかわらず、一意である必要があります。

Oracle Parallel Server を使用禁止にして起動したインスタンスは、INSTANCE_NUMBER パラメータを使用してインスタンス番号を指定できます。これは、インスタンスが挿入および更新を実行した場合、およびデータベースにある表が FREELIST GROUPS 記憶域オプションを使用して空き領域をインスタンスに割り当てる場合にのみ必要です。

管理操作の実行のみのために、Oracle Parallel Server を使用禁止にしてインスタンスを起動する場合、パラメータ・ファイルの INSTANCE_NUMBER パラメータを省略できます。

Oracle Parallel Server を使用禁止にして起動したインスタンスは、1 以外のスレッドを指定して、そのスレッドに対応付けられたオンライン REDO ログ・ファイルを使用することもできます。

参照：

- 3-4 ページの「追加のロールバック・セグメントの作成」を参照してください。
- 挿入および更新用の空き領域の割当ての詳細は、第 8 章を参照してください。
- Oracle データベースの起動の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

複数インスタンスに対する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対する Oracle Parallel Server 初期化パラメータについて説明します。内容は次のとおりです。

- 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
- すべてのインスタンスで一意にする必要があるパラメータ
- 共通パラメータ・ファイルのパラメータ
- 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初期化パラメータに関する注意

参照： その他の Oracle 初期化パラメータについては、『Oracle8*i* リファレンス・マニュアル』を参照してください。

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

データベース作成に重要な初期化パラメータ、または特定のデータベース操作に影響する特定の初期化パラメータは、Oracle Parallel Server の各インスタンスに対して同じ値にする必要があります。これらのパラメータ値は、共通パラメータ・ファイル、または各インスタンスの init_*dbname*.ora 内に指定する必要があります。表 1-1 に、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を示します。

表 1-1 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

CONTROL_FILES	LM_LOCKS および LM_RESS（Oracle によって自動的に計算されますが、同じ値を設定することをお薦めします。）
DB_BLOCK_SIZE	LOG_ARCHIVE_DEST（オプション）
DB_FILES	MAX_COMMIT_PROPAGATION_DELAY
DB_NAME	SERVICE_NAMES
DB_DOMAIN	ACTIVE_INSTANCE_COUNT
DML_LOCKS	ROW_LOCKING
GC_FILES_TO_LOCKS	GC_ROLLBACK_LOCKS
PARALLEL_SERVER_INSTANCES	DML_LOCKS（0（ゼロ）に設定されている場合のみ）

すべてのインスタンスで一意にする必要があるパラメータ

パラメータ INSTANCE_NUMBER、THREAD または ROLLBACK_SEGMENTS を使用する場合、インスタンス固有のパラメータ・ファイルを使用して、これらのパラメータに一意の値を設定することをお薦めします。

- Oracle は、INSTANCE_NUMBER パラメータを使用して、起動時にインスタンスを識別します。また、INSTANCE_NUMBER を使用して、ALTER TABLE または ALTER CLUSTER 文にある ALLOCATE EXTENT 句の INSTANCE オプションによってインスタンスに空き領域を割り当てます。あるインスタンスに INSTANCE_NUMBER の値を割り当てた場合、すべてのインスタンスに値の割当てを行い、各インスタンスに一意の値を使用する必要があります。
- 起動時および停止時に異なるスレッド番号を取得することによって発生するオーバーヘッドをインスタンスが回避できるように、THREAD パラメータを指定します。Oracle は、この THREAD 番号を使用して、REDO ログ・ファイル・グループを特定のインスタンスに割り当てます。管理を簡単にし、混乱を避けるために、INSTANCE_NUMBER パラメータと同じ番号を THREAD パラメータに使用します。
- プライベート・ロールバック・セグメントは、パブリック・ロールバック・セグメントより書込み競合が少ないため、いくつかのシステムのパフォーマンスを向上できます。Oracle は、インスタンスの起動時に、ROLLBACK_SEGMENTS 初期化パラメータを使用して識別したロールバック・セグメントを取得します。このパラメータによって、イ

ンスタンスに対してロールバック・セグメント・ファイル名を宣言しない場合、Oracle はこのインスタンスに対してパブリック・ロールバック・セグメントを取得します。

共通パラメータ・ファイルのパラメータ

この項では、共通パラメータ・ファイルについて説明します。内容は次のとおりです。

DB_NAME パラメータ

DB_NAME パラメータは、必ず共通パラメータ・ファイルに含めてください。共通ファイルに DB_NAME の値を設定しない場合、このパラメータの値をインスタンスまたはデフォルト以外のパラメータ・ファイルに設定する必要があります。このパラメータに設定する値は、すべてのインスタンスで同じである必要があります。

IFILE によって参照される共通パラメータ・ファイルに同じ値を持つパラメータを指定する場合、デフォルト値を使用しているパラメータは省略可能です。

GC_* グローバル・キャッシュ・パラメータ

GC（グローバル・キャッシュ）という接頭辞が付く初期化パラメータは、Oracle Parallel Server でのみ使用できます。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のデータベース・バッファを保護するグローバル・ロックのコレクション・サイズが決定されます。設定した値は、特定のオペレーティング・システム・リソースの使用に影響します。これらのパラメータは、`init_dbname.ora` ファイルに設定します。

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・パラメータ値を決定します。最初のインスタンスの起動時に、制御ファイルが、GC_* パラメータの値を記録します。

他のインスタンスが共有モードで起動しようとすると、Oracle は、パラメータ・ファイル内のグローバル・キャッシュ・パラメータの値と、すでに使用されている値を比較し、一致しない値があればメッセージを発行します。インスタンスは、そのインスタンスのグローバル・キャッシュ・パラメータの値が正しくない場合は、データベースをマウントできません。

Oracle Parallel Server のグローバル・キャッシュ・パラメータを次に示します。

パラメータ	説明
GC_FILES_TO_LOCKS	データ・ブロックに対するデータ・ブロック・ロックの割合を制御します（すべてのインスタンスで同じにする必要があります）。
GC_ROLLBACK_LOCKS	UNDO ブロック・ロックの数を制御します（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	解放可能ロックの数を制御します。
GC_DEFER_TIME	他のインスタンスからのホット・ブロックの強制書込み要求に応答する前に、Oracle が待機する時間を 1000 分の 1 秒単位で指定します。

参照： グローバル・キャッシュ・パラメータの設定の詳細は、第 III 部「Oracle Parallel Server の設計および配置」を参照してください。

複数インスタンスの初期化パラメータに関する注意

表 1-2 に、複数インスタンスの初期化パラメータに関する注意を示します。

表 1-2 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	説明およびコメント
DML_LOCKS	0（ゼロ）に設定されている場合のみ、すべてのインスタンスで同じ値である必要があります。この値は、すべてのユーザーが現在参照している表にあるロックの合計と同じである必要があります。たとえば、3 人のユーザーが 1 つの表のデータを変更している場合、3 つのエントリが必要です。3 人のユーザーが 2 つの表のデータを変更している場合は、6 つのエントリが必要です。 デフォルト値では、1 つのトランザクションで参照される表の平均を 4 と仮定しています。システムによっては、この値は十分ではない場合があります。DML_LOCKS の値を 0（ゼロ）に設定すると、エンキューが使用不可になり、パフォーマンスがわずかに向上します。ただし、DROP TABLE、CREATE INDEX または明示的なロック文（LOCK TABLE IN EXCLUSIVE MODE など）は使用できません。Oracle は、パラレル DML 中は、シリアル実行中より多くのロックを保持します。したがって、データベースが多くのパラレル DML をサポートする場合、このパラメータの値を増やす必要があることがあります。
INSTANCE_NUMBER	このパラメータの値を指定する場合は、すべてのインスタンスに対して一意である必要があります。Oracle Parallel Server 環境では、複数インスタンスを単一データベース・サービスに対応付けることができます。クライアントは、データベースへの接続に特定のインスタンスを指定することによって、接続時ロード・バランス化をオーバーライドできます。INSTANCE_NAME は、このインスタンスの一意の名前を指定します。単一インスタンス・データベース・システムでは、通常、インスタンス名はデータベース名と同じです。

表 1-2 複数インスタンスの初期化パラメータに関する注意（続き）

パラメータ	説明およびコメント
LOG_ARCHIVE_FORMAT	ARCHIVELOG モードで REDO ログを使用している場合にのみ適用できます。REDO ログ・ファイルをアーカイブするときは、テキスト文字列および変数を使用してデフォルト・ファイル名フォーマットを指定します。このフォーマットから生成された文字列は、LOG_ARCHIVE_DEST パラメータに指定されている文字列に追加されます。スレッド番号を含める必要があります。 次の変数をフォーマットに使用できます。 ■ %s: ログ順序番号 ■ %S: 0（ゼロ）が埋められたログ順序番号 ■ %t: スレッド番号 ■ %T: 0（ゼロ）が埋められたスレッド番号 変数（たとえば、%S）に大文字を使用すると、値が固定長になり、左詰に 0（ゼロ）が埋められます。アーカイブ REDO ログ・ファイル名フォーマットを指定する例を次に示します。 ■ LOG_ARCHIVE_FORMAT = "LOG%s_%t.ARC"
MAX_COMMIT_ PROPAGATION_ DELAY	Oracle Parallel Server 固有のパラメータです。ただし、Oracle Parallel Server 固有の、ある限られた状況下以外では、このパラメータを変更しないでください。 このパラメータは、インスタンスの SGA に保持されるシステム変更番号（SCN）が、ログ・ライター・プロセス（LGWR）によってリフレッシュされるまでの時間の最大値を指定します。また、問合せに対するスナップショット SCN の取得時に、ロック値からローカル SCN をリフレッシュするかどうかを決定します。単位は 1000 分の 1 秒です。異なるインスタンスからの、同一データの高速更新や問合せなどの異常な状況では、SCN は、すぐにはリフレッシュされない場合があります。パラメータを 0（ゼロ）に設定すると、SCN はコミットの後すぐにリフレッシュされます。デフォルト値（700,000 分の 1 秒、または 7 秒）は、推奨する既存の高パフォーマンス・メカニズムを維持できる上限値です。 リモート・インスタンスがすぐにコミットを参照する必要がある場合は、このパラメータの値を変更する必要があることがあります。
NLS_* パラメータ	『Oracle8*i* リファレンス・マニュアル』に記載するとおり、NLS パラメータには数種類あります。異なる値を異なるインスタンスに設定できます。
PARALLEL_SERVER	データベースを Oracle Parallel Server モードで起動するには、インスタンス初期化ファイル（`init_sid.ora`）でこのパラメータを TRUE に設定します。

表 1-2 複数インスタンスの初期化パラメータに関する注意（続き）

パラメータ	説明およびコメント
PARALLEL_SERVER_INSTANCES	Oracle Parallel Server 環境のインスタンスの数と同じ値を設定します。Oracle は、このパラメータの値を使用して、パフォーマンスを最適化するためのメモリー構造のサイズを決めます。PARALLEL_SERVER_INSTANCES は、Oracle Parallel Server パラメータであり、現在構成されているインスタンスの数を指定します。このパラメータは、各インスタンスに設定する必要があります。 通常、このパラメータには、Oracle Parallel Server 環境のインスタンスの数を設定する必要があります。Oracle は、このパラメータの値を使用して、PARALLEL_AUTOMATIC_TUNING パラメータが TRUE に設定されたときに LARGE_POOL_SIZE パラメータのデフォルト値を計算します。このパラメータを適切に設定すると、メモリー使用が改善されます。
PROCESSES	このパラメータには、すべてのバックグラウンド・プロセスおよびユーザー・プロセスを処理するために十分な値を設定する必要があります。オペレーティング・システムによっては、追加の DBWR プロセスがある場合があります。SESSIONS 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。PROCESSES は、Oracle に同時に接続できるオペレーティング・システム・ユーザー・プロセスの最大数を指定します。 この値には、ロック、ジョブ・キュー・プロセス、パラレル実行プロセスなどのすべてのバックグラウンド・プロセスを考慮する必要があります。SESSIONS および TRANSACTIONS パラメータのデフォルト値は、このパラメータから導出されます。したがって、PROCESSES の値を変更する場合、これらの導出パラメータの値を調整するかどうかを評価する必要があります。デフォルトを使用しない場合、追加の LCKn プロセスおよび他のオプションのバックグラウンド・プロセスを考慮して、前述のいくつかのパラメータの値を増やす必要があります。
この表に示した 8 つのパラメータにデフォルトを使用しない場合、いくつかのパラメータの値を増やす必要がある場合があります。これによって、Oracle は、追加の LCKn プロセスおよび他のバックグラウンド・プロセスを作成してパフォーマンスを向上します。	
RECOVERY_PARALLELISM	ロール・フォワードまたはキャッシュ・リカバリ・フェーズの処理速度を上げるために、このパラメータを設定して、インスタンスまたはクラッシュ・リカバリに使用するプロセスの数を指定できます。0（ゼロ）または 1 に設定すると、リカバリが 1 つのプロセスによってシリアルに実行されます。

表 1-2 複数インスタンスの初期化パラメータに関する注意（続き）

パラメータ	説明およびコメント
ROLLBACK_SEGMENTS	このパラメータを使用すると、1 つ以上のロールバック・セグメントの名前を 1 つのインスタンスに割り当てることによって、各インスタンスに対するプライベート・ロールバック・セグメントを指定できます。このパラメータを設定すると、ロールバック・セグメントの数がインスタンスに必要な最小数を超える場合でも、インスタンスは、このパラメータに指定したすべてのロールバック・セグメントを取得します。この最小値は、次の割合から計算します。 TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT このパラメータの値を動的に変更することはできません。ただし、値を変更してからインスタンスを再起動することはできます。このパラメータは、通常、プライベート・ロールバック・セグメントを指定しますが、パブリック・ロールバック・セグメントがまだ使用されていない場合は、このセグメントも指定できます。データベース内の各ロールバック・セグメントの名前、セグメント ID 番号および状態を検索するには、データ・ディクショナリ・ビュー DBA_ROLLBACK_SEG を問い合せます。
THREAD	指定する場合、このパラメータの値はすべてのインスタンスに対して一意である必要があります。THREAD は、Oracle Parallel Server パラメータであり、このインスタンスに使用される REDO スレッド番号を指定します。データベースの作成時に、Oracle は、スレッド 1 をパブリック・スレッド（どのインスタンスも使用できるスレッド）として作成および使用可能にします。ALTER DATABASE 文の ADD LOGFILE THREAD 句および ENABLE THREAD 句を使用して、後続のスレッドを作成および使用可能にする必要があります。 作成するスレッドの数は、CREATE DATABASE 文に指定する MAXINSTANCES パラメータによって制限されます。排他モードでは、スレッド 1 はデフォルト・スレッドです。ただし、スレッド 1 以外のスレッドで REDO ログ・ファイルを使用する必要がある場合は、排他モードで実行するインスタンスに THREAD を指定できます。パラレル・モードでは、スレッド番号が使用可能で、他のインスタンスが使用していない限り、使用可能なすべての REDO スレッド番号を指定できます。 0（ゼロ）に設定すると、このインスタンスは使用可能なすべてのパブリック・スレッドを使用できます。

MTS_DISPATCHER パラメータおよび Oracle Parallel Server

マルチスレッド・サーバー構成を使用可能にするには、共通ファイルに MTS_DISPATCHERS パラメータを設定します。MTS_DISPATCHERS パラメータは、多くの属性を含むことができます。

少なくとも、PROTOCOL 属性および LISTENER 属性を構成することをお薦めします。PROTOCOL は、ディスパッチャがリスニングのエンドポイントを生成するネットワーク・プロトコルを指定します。LISTENER は、PMON プロセスがディスパッチャ情報を登録するリスナーの別名を指定します。別名は、`tnsnames.ora` ファイルなどのネーミング・メソッドを使用して変換される名前に設定します。

参照： MTS_DISPATCHER パラメータおよびその属性の構成、およびマルチスレッド・サーバーの構成の詳細は、『Oracle8*i* Parallel Server セットアップおよび構成ガイド』および『Oracle8*i* Net8 管理者ガイド』を参照してください。

LM_* 初期化パラメータ

分散ロック・マネージャの容量は、Oracle が LM_RESS パラメータおよび LM_LOCKS パラメータに設定する値によって決定されます。表 1-3 に、これらのパラメータを示します。分散ロック・マネージャは、LM_RESS および LM_LOCKS に対する値を自動的に計算します。

共有プールの領域が不足した場合、または V$RESOURCE_LIMIT ビューに表示される共有プールの最大使用率が、これらのパラメータに Oracle が設定した値を超える場合は、第 10 章を参照して、LM_RESS および LM_LOCKS を調整してください。これ以外の場合は、これらのパラメータを設定する必要はありません。設定を調整する場合、管理を簡単にするために、これらのパラメータの値をすべてのインスタンスに対して同じに設定することをお薦めします。

表 1-3 LM_* 初期化パラメータ

パラメータ	説明
LM_RESS	このパラメータは、分散ロック・マネージャ（DLM）がロックできるリソースの数を制御します。このパラメータには、DML、DDL、データ・ディクショナリ・キャッシュ・ロックやファイル管理ロックとログ管理ロックに割り当てられるロック・リソースの数などの非 PCM リソースが含まれます。
LM_LOCKS	このパラメータは、ロックの数を制御します。N は、ノードの合計数です。 LM_LOCKS = LM_RESS + (LM_RESS * (N - 1))/N

パラレル DML または DML をパーティション・オブジェクト上で実行する場合は、LM_RESS および LM_LOCKS の値を増やしてください。

第II部

Oracle Parallel Server の管理

第 II 部では、Oracle Parallel Server 環境について考慮する必要がある、一般的な管理作業について説明します。第 II 部に含まれる章は、次のとおりです。

2

Oracle Parallel Server 環境のパラレル実行

この章では、Oracle Parallel Server 環境のパラレル実行およびその使用方法について説明します。内容は次のとおりです。

- Oracle Parallel Server のパラレル実行
- Oracle Parallel Server のパラレル実行パラメータ
- パラレル実行のその他のリソース管理機能
- 動的パフォーマンス・ビュー
- ディスク親和性およびパラレル実行

注意： パラレル実行パラメータは、インスタンスを起動する前に設定してください。

Oracle Parallel Server のパラレル実行

パラレル実行に Oracle Parallel Server は必要ありませんが、この章で説明するパラレル実行の一部には、Oracle Parallel Server 環境にのみ適用されるものがあります。

参照：

- Oracle Parallel Server のパラレル実行の詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。
- パラレル実行の使用方法の詳細は、『Oracle8*i* データ・ウェアハウス』を参照してください。

並列度の設定

並列度を設定する場合、クラスタ内の使用可能なすべての CPU を考慮してください。パラレル実行自動チューニング機能を使用する場合、Oracle は他のパラレル実行関連パラメータを設定します。

参照： パラレル実行自動チューニング機能、および並列度の設定の詳細は、『Oracle8*i* データ・ウェアハウス』を参照してください。

Oracle Parallel Server のパラレル実行パラメータ

Oracle Parallel Server 環境では、次の 2 つのパラメータがパラレル実行に影響します。

- INSTANCE_GROUPS
- PARALLEL_INSTANCE_GROUP

Oracle Parallel Server 環境でパラレル実行がリソースを使用する方法を制御するには、これらのパラメータを同時に使用します。

インスタンス・グループへのリソースの割当て

インスタンス・グループを使用すると、管理を簡単にし、パラレル実行に使用するインスタンスをより効率的に制御できます。インスタンス・グループは、パラレル実行、OLTP、データ・ウェアハウス操作などの特定の目的に使用したインスタンス集合です。

たとえば、ユーザーが午前 9 時～午後 5 時にはインスタンス・グループ B にアクセスし、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、通常の OLTP 挿入および更新の場合は、ユーザーをグループ C にアクセスさせ、大きいパラレル作業の場合はユーザーをグループ D にアクセスさせることによって、OLTP のパフォーマンスに影響しないようにすることができます。

すべてのインスタンス・グループは、データベースを起動する前に定義する必要があります。インスタンスを動的にグループに追加したり、グループから削除することはできません。インスタンス・グループは、著しいオーバーヘッドを起こさないため、グループの数は無制限に定義できます。定義したインスタンス・グループは、必ず使用する必要があるわけではありません。1 つのインスタンスは複数のグループに所属でき、グループは互いにオーバーラップできます。

パラレル実行の場合、インスタンス・グループを使用しない場合は、Oracle がパラレル実行に使用するインスタンスを判断します。Oracle は、これをディスク親和性および実行中のインスタンスの数に基づいて判断します。パラレル化 SQL 文を開始するインスタンスが、その文を処理するインスタンス・グループのメンバーである必要はありません。ただし、パラレル・コーディネータは、SQL 文を発行したインスタンス上で実行されます。

インスタンス・グループの指定

インスタンスが所属するインスタンス・グループを指定するには、INSTANCE_GROUPS 初期化パラメータを設定します。パラメータの後にインスタンス・グループの名前を指定し、これらのエントリを、グループに対応付ける各インスタンスのパラメータ・ファイルに含めます。これによって、INSTANCE_GROUPS パラメータは、グループの定義および現行インスタンスとの対応付けを同時に行います。

たとえば、インスタンス 1 がグループ A およびグループ B のメンバーであることを定義するには、インスタンス 1 のデータベース初期化ファイルに次のエントリを含めます。

```
INSTANCE_GROUPS = GROUPA, GROUPB
```

インスタンス 2 がグループ A およびグループ C のメンバーであることを定義するには、インスタンス 2 のデータベース初期化ファイルに次のエントリを含めます。

```
INSTANCE_GROUPS = GROUPA, GROUPC
```

この結果、インスタンス 1 およびインスタンス 2 の両方がインスタンス・グループ A に所属し、かつ、それぞれが他のグループにも所属することになります。

注意： INSTANCE_GROUPS は静的なパラメータであり、セッションの途中で変更することはできません。

パラレル・インスタンス・グループの定義

パラレル操作に使用するインスタンス・グループを定義するには、PARALLEL_INSTANCE_GROUP パラメータを使用します。INSTANCE_GROUPS パラメータを使用した場合、PARALLEL_INSTANCE_GROUP のデフォルトは、すべての実行中のインスタンスから導出されたグループです。

あるパラレル操作に特定のインスタンス・グループを使用するには、共通パラメータ・ファイルに次のように指定します。

```
PARALLEL_INSTANCE_GROUP = GROUPNAME
```

GROUPNAME は、パラレル操作に指定するインスタンス・グループ名です。

パラメータ・ファイルにこのエントリを含むインスタンスから開始されたすべてのパラレル操作は、GROUPNAME で定義されたグループ内でのみプロセスを起動します。INSTANCE_GROUPS の設定とは異なり、PARALLEL_INSTANCE_GROUP の値は、ALTER SESSION 文または ALTER SYSTEM 文を使用して変更できます。ただし、1 つのインスタンス・グループを参照するには、PARALLEL_INSTANCE_GROUP のみを使用します。ユーザーが実行しているインスタンスは、特定の操作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。

インスタンス・グループ例

この例では、インスタンス 1 のパラメータ・ファイルの設定は次のとおりです。

```
INSTANCE_GROUPS = GROUPA, GROUPB
PARALLEL_INSTANCE_GROUP  GROUPB
```

インスタンス 2 のパラメータ・ファイルの設定は次のとおりです。

```
INSTANCE_GROUPS = GROUPB, GROUPC
PARALLEL_INSTANCE_GROUP = GROUPC
```

インスタンス 1 で次の文を入力した場合、Oracle はパラレル操作にグループ GROUPB のインスタンスを使用します。インスタンス 1 およびインスタンス 2 は、ともにグループ GROUPB のメンバーであるため、Oracle はインスタンス 1 で 2 つのサーバー・プロセスを起動し、インスタンス 2 でも 2 つのサーバー・プロセスを起動します。

```
ALTER TABLE TABLE PARALLEL (DEGREE 2);
SELECT COUNT(*) FROM TABLE;
```

ただし、インスタンス 1 で次の文を入力した場合、Oracle は 2 つのサーバー・プロセスをインスタンス 2 のみで起動することによって、グループ GROUPC のみをパラレル実行に使用します。

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = 'GROUPC';
SELECT COUNT (*) FROM TABLE;
```

これは、インスタンス 1 がパラレル・インスタンス・グループ GROUPC のメンバーではないためです。

すべてのインスタンスをパラレル操作に使用するには、パラレル・インスタンス・グループを宣言するときに、単一引用符で囲んだ空白を使用します。たとえば、インスタンス 1 で次の文を入力した場合、Oracle はデフォルトのインスタンス・グループまたは現在実行中のすべてのインスタンスをパラレル処理に使用します。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;
```

インスタンス・グループ・メンバーのリスト

インスタンス・グループのメンバーを参照するには、GV$PARAMETER ビューを問い合せ、INSTANCE_GROUPS パラメータのエントリを調べます。

> **参照：** 初期化パラメータおよびビューの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

パラレル実行のその他のリソース管理機能

リソースの使用を最適化するパラレル実行のその他の機能には、次のものがあります。

- パラレル実行ロード・バランス
- パラレル実行マルチユーザー問合せ調整

パラレル実行ロード・バランス

パラレル実行ロード・バランスは、サーバー・プロセスをインスタンス間に分散して、ロードを均衡化します。これによって、複数インスタンス上でのパラレル実行およびパラレル DML 操作のロード・バランスが改善されます。

この自動並列度をチューニングすることはできませんが、自動パラレル実行のロード・バランス・アルゴリズムに影響するように、データベース・スケジューラ値を調整することはできます。

類似ノードのロードは、非類似ノードより約 10 ～ 15% 高くなります。ロード・バランス機能は、ベンダー固有の Cluster Manager ソフトウェアを使用してインスタンス間の通信を行います。超並列システムでは、Oracle は非類似ノードを移入する前に、類似ノードを移入します。

パラレル実行マルチユーザー問合せ調整

Oracle Parallel Server システムの作業負荷が変更されると、マルチユーザー問合せ調整機能は、クラスタ全体で検出した作業負荷を基に、入力される SQL 文の並列度を変更します。Oracle は、インスタンスの数および各インスタンスの現在の作業負荷に基づいて、並列度を調整します。この機能を使用可能にするには、PARALLEL_ADAPTIVE_MULTIUSER パラメータを TRUE に設定します。

入力される SQL 文の並列度が小さく、かつ、他のインスタンスがビジーで SQL 文の一部を処理できない場合、Oracle は作業負荷をいくつかのインスタンスに分散するのではなく、1 つのインスタンスに置きます。これは、複数のインスタンスが SQL 文の処理を共同で行う場合、パラレル実行に使用するインスタンスは、リソースを使用して互いに通信する必要があるため、ノード内通信コストが非常に高くなるためです。

ユーザーが同時パラレル実行操作を処理する場合は、パラレル・マルチユーザー問合せ調整機能を使用することをお薦めします。パラレル実行自動チューニング機能を使用可能にすると、Oracle は、自動的に PARALLEL_ADAPTIVE_MULTI_USER を TRUE に設定します。

パラレル・プロセスにおけるディスク競合の回避

パラレル表スキャン中のディスク競合を回避するには、インスタンス間で表をストライプ化します。そのためには、ディスク上でオペレーティング・システム・ストライプ化を実行するか、または複数ノード上のファイルを使用する表領域を作成します。

> **参照：** パラレル実行ロード・バランスおよびマルチユーザー問合せ調整機能の詳細は、『Oracle8*i* データ・ウェアハウス』を参照してください。

動的パフォーマンス・ビュー

Oracle Parallel Server 環境内のパラレル実行アクティビティを調べるには、次のパフォーマンス・ビューを使用します。

- GV$PX_SESSION
- GV$PX_SESSSTAT
- GV$PX_PROCESS
- GV$PX_PROCESS_SYSSTAT

参照： これらのビューを使用してパラレル実行を評価する場合の詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

ディスク親和性およびパラレル実行

ディスクに対してシェアード・ナッシング・アプローチを使用し、オペレーティング・システム固有のソフトウェア・レイヤーを介してそのようなディスクをグローバルに参照できるようなシステムに限り、ディスク親和性が使用可能です。ディスク親和性は、パラレル化 DML または問合せ操作を実行するインスタンスを判断します。親和性は、Oracle Parallel Server 構成のパラレル DML に特に重要です。複数の文を通じた一貫した親和性情報によって、バッファ・キャッシュ・ヒット率が改善され、強制読込み / 書込みが削減されます。

ほとんどのパラレル DML 操作のパラレル化の細分性は、パーティション単位です。ただし、パラレル実行の場合、細分性は ROWID 単位です。パラレル DML 操作には、親和性を実現するためにパーティションからインスタンスへのマッピングが必要です。パーティションのセグメント・ヘッダーは、超並列プロセス・システムのパーティションの親和性を判断するために使用されます。ノードをローカル・デバイスにアクセスさせることによって、パフォーマンスを向上できます。これによって、各ノードのバッファ・キャッシュ・ヒット率が向上します。

その他の Oracle Parallel Server 構成の場合、パーティションからインスタンスへの決定的なマッピングが使用されます。パーティションおよびインスタンス間の親和性情報は、すべての Oracle Parallel Server/MPP 構成のプロセス割当ておよび作業割当てを判断するために使用します。

参照：

- パラレル・データ操作言語（PDML）および並列度の詳細は、『Oracle8*i* 概要』を参照してください。
- PDML チューニングの説明およびオプティマイザのヒントは、『Oracle8*i* パフォーマンスのための設計およびチューニング』を参照してください。
- ディスク親和性に関するオペレーティング・システム固有の情報については、各インストレーションおよび構成ガイドも参照してください。

3

Oracle Parallel Server データベース作成に関する問題

この章では、Oracle Parallel Server データベースの作成に関する問題について説明します。この章の情報は、『Oracle8*i* Parallel Server セットアップおよび構成ガイド』に記載されている Database Configuration Assistant の使用方法についての情報を補足します。内容は次のとおりです。

- 複数インスタンス環境に対するデータベース作成
- データベース作成に対する初期化パラメータの設定
- CREATE DATABASE オプションの設定
- 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
- CREATE DATABASE オプションの値の変更

注意： Database Configuration Assistant は、Oracle Parallel Server データベースに必要なオブジェクトを作成します。Database Configuration Assistant が作成するデータベースが要件に一致しなかった場合は、手動でデータベースを作成するのではなく、Database Configuration Assistant を使用してデータベースを作成した後、それを変更することをお薦めします。

参照： データベースの作成手順に関する情報は、『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。

複数インスタンス環境に対するデータベース作成

この項では、Oracle Parallel Server 固有のデータベース作成の次の設定について説明します。

- データベース作成に対する初期化パラメータの設定
- CREATE DATABASE オプションの設定

データベース作成に対する初期化パラメータの設定

第 1 章で説明するとおり、データベース作成に重要な、または特定のデータベース操作に影響する特定の初期化パラメータは、すべてのインスタンスに対して同じ値を持つ必要があります。Oracle Parallel Server データベースを作成する前に、これらのパラメータの設定がすべてのインスタンスで同じであることを確認してください。

ARCHIVELOG モードの使用

データベースの作成中、アーカイブ・プロセス（ARCH）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し、オーバーヘッドを削減することもできます。これはデフォルトです。その後、ARCHIVELOG モードに変更します。

STARTUP コマンドを使用してデータベース・アーカイブ・モードを変更することはできません。かわりに、データベースを作成した後、次のコマンドを使用してアーカイブ・モードを変更し、Parallel Server を使用可能にしてデータベースを再度オープンします。

```
ALTER DATABASE CLOSE;
ALTER DATABASE ARCHIVELOG;
SHUTDOWN;
STARTUP;
```

CREATE DATABASE オプションの設定

この項では、Oracle Parallel Server 固有の CREATE DATABASE オプションの設定について説明します。

- MAXINSTANCES の設定
- MAXLOGFILES および MAXLOGMEMBERS の設定

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可能なインスタンスの数を制限します。MAXINSTANCES のデフォルトは、ご使用のオペレーティング・システム固有の最大値です。

Oracle Parallel Server では、MAXINSTANCES を、同時に実行すると予想されるインスタンスの最大数より大きい値に設定します。これによって、インスタンス A に障害が起きてインスタンス B によってリカバリされている場合でも、インスタンス A が完全にリカバリ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 オプションは、データベースに作成可能な REDO ログ・グループの最大数を指定します。MAXLOGMEMBERS オプションは、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。Parallel Server では、MAXLOGFILES を、予想スレッド数の最大値に、1 スレッドあたりのグループの予想最大数を掛けた値に設定します。

MAXLOGHISTORY の設定

CREATE DATABASE の MAXLOGHISTORY オプションは、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、Oracle Parallel Server の自動メディア・リカバリに使用します。

Oracle Parallel Server の場合、MAXLOGHISTORY を 1000 などの大きい値に設定する必要があります。これによって、制御ファイルはこの数の REDO ログ・ファイルの情報のみを格納します。ログ履歴がこの制限を超えた場合、Oracle は一番古いエントリを上書きします。MAXLOGHISTORY のデフォルトは 0（ゼロ）で、これはログ履歴を使用禁止にします。

MAXDATAFILES の設定

MAXDATAFILES オプションは汎用ですが、Oracle Parallel Server は、標準システムより多くのデータ・ファイルおよびログ・ファイルを持つ傾向があります。ご使用のプラットフォームでは、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。

参照：

- SQL 文 CREATE DATABASE および ALTER DATABASE の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。
- REDO ログ・グループおよびメンバーの詳細は、13-7 ページの「制御ファイルの REDO ログ履歴」を参照してください。

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

Oracle Parallel Server に新しいデータベースを作成するには、次のように、追加のデータベース・オブジェクトを作成および構成します。

- 追加のロールバック・セグメントの作成
- Oracle Parallel Server のオンライン REDO ログの構成
- 追加データ・ファイルへのロックの提供

追加のロールバック・セグメントの作成

Parallel Server の各インスタンスに、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。競合を回避するには、これらのロールバック・セグメントを個別の表領域に作成します。これらのロールバック・セグメントは、SYSTEM 表領域に格納しないでください。

別の表領域にロールバック・セグメントを作成する前に、1 つの追加ロールバック・セグメントを SYSTEM 表領域に作成し、オンライン化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は、追加のロールバック・セグメントおよび新しい表領域を作成できますが、追加のロールバック・セグメントをオンライン化するまで、SYSTEM 表領域以外にデータベース・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 1 つのインスタンスに割り当てるには、次の手順に従います。

1. SQL 文 CREATE ROLLBACK SEGMENT を使用して（キーワード PUBLIC は省略）ロールバック・セグメントを作成します。ロールバック・セグメントを作成する前に、オプションで、それに対する表領域を作成しておくこともできます。
2. ロールバック・セグメントをパラメータの値として指定することによって、インスタンスのパラメータ・ファイルにロールバック・セグメントを指定します。これによって、そのインスタンスにロールバック・セグメントが予約されます。
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。また、インスタンスを再起動して、予約したロールバック・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 つのインスタンスのみに対応付けられるように、インスタンス初期化パラメータでのみ指定されている必要があります。インスタンスが、別のインスタンスがすでに取得したプライベート・ロールバック・セグメントを取得しようとすると、エラー・メッセージが戻され、インスタンスは起動できません。

パブリック・ロールバック・セグメントの使用

どのインスタンスでも、すべてのインスタンスが使用可能なパブリック・ロールバック・セグメントを作成できます。ロールバック・セグメントが 1 つのインスタンスによって使用されると、インスタンスが停止するまで、そのインスタンスのみで使用されます。ロールバック・セグメントを使用していたインスタンスが停止すると、ロールバック・セグメントは解放され、他のインスタンスによって使用されます。

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 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、データ・ディクショナリ・ビュー DBA_ROLLBACK_SEGS の中では PUBLIC が所有者となります。インスタンスに ROLLBACK_SEGMENTS パラメータの値を設定しない場合、インスタンスはパブリック・ロールバック・セグメントを使用します。ロールバック・セグメントを作成および管理する手順は、Parallel Server が使用可能でも使用禁止でも同じです。

パブリック・ロールバック・セグメントは、それらを必要とするすべてのインスタンスに使用可能であると想定されるため、通常、特定のインスタンスのパラメータ・ファイルはパブリック・ロールバック・セグメントを指定しません。ただし、別のインスタンスがパブリック・ロールバック・セグメントをまだ使用していなければ、ROLLBACK_SEGMENTS パラメータの値にパブリック・ロールバック・セグメントを指定できます。

パブリック・ロールバック・セグメントは、インスタンスが起動時に取得したときにオンライン化されますが、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特定のパブリック・ロールバック・セグメントを使用することは保証できません。ただし、使用可能なすべてのパブリック・ロールバック・セグメントをインスタンスが取得する場合を除きます。

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、オフラインのままです。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のため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してそれを取得するまで、オフラインのままです。

パブリック・ロールバック・セグメントをオフラインにしておく必要があり、削除および再作成しない場合は、パブリック・ロールバック・セグメントを要求する他のインスタンスが起動され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

ロールバック・セグメントを監視するには、動的パフォーマンス・ビュー V$ROLLNAME および V$ROLLSTAT を問い合せ、現行インスタンスのロールバック・セグメントの情報を入手します。また、データ・ディクショナリ・ビュー DBA_ROLLBACK_SEGS および

DBA_SEGMENTS、またはグローバル動的ビュー GV$ROLLNAME および GV$ROLLSTAT を問い合せて、ロールバック・セグメントの情報を入手することもできます。

別のインスタンス上のロールバック・セグメントを監視するには、MONITOR コマンドを使用したり V$ ビューを問い合せる前に、CONNECT@*instance-path* コマンドを使用して現行インスタンスを変更します。

インスタンスが現在使用しているロールバック・セグメントを表示するには、次の構文で DBA_ROLLBACK_SEGS を問い合せます。

```
SELECT segment_name, segment_id, owner, status
   FROM dba_rollback_segs
```

この問合せは、ロールバック・セグメント名、ID 番号、所有者、および使用中かオンラインかを表示します。次に出力例を示します。

```
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 が所有するロールバック・セグメントは、プライベート・ロールバック・セグメントです。ユーザー PUBLIC が所有するロールバック・セグメントは、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、セグメント・ヘッダーを含むデータ・ファイル、およびエクステント・サイズについての情報（この例では示しません）が含まれます。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ます。

参照：

- ロールバック・セグメントの詳細およびデータベースとの接続方法は、『Oracle8*i* 管理者ガイド』を参照してください。
- DBA_ROLLBACK_SEGS および DBA_SEGMENTS の説明、および他の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。

Oracle Parallel Server 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、独自のオンライン REDO ログ・グループからなる、オンライン REDO の独自スレッドがあります。Oracle Parallel Server の実行時は、複数のインスタンスが同時に単一のデータベースにアクセスし、各インスタンスは独自のスレッドを持つ必要があります。この項では、Oracle Parallel Server を使用した複数インスタンスのオンライン REDO スレッドの設定方法を説明します。

各スレッドには、少なくとも 2 つの REDO ログ・ファイルが必要です（多重化）。また、インスタンスがスレッドを使用する前に、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、それを自動的に使用可能にします。ALTER DATABASE 文を使用して、後続のスレッドを作成および使用可能にします。

スレッドの作成

これらのスレッドは、パブリックでもプライベートでもかまいません。初期化パラメータ THREAD は、インスタンスに一意のスレッド番号を割り当てます。THREAD をデフォルトの 0（ゼロ）に設定すると、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。

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で作成される必要があります。インスタンスがスレッドを使用する前に、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、それを自動的に使用可能にします。後続のスレッドは、ALTER DATABASE 文を使用して作成および使用可能にできます。たとえば、次の文は、それぞれ 3 つのメンバーを持つ 2 つのグループのあるスレッド 2 を作成します。

```
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 を省略すると、デフォルトでは取得されないプライベート・スレッドになります。ALTER DATABASE ADD LOGFILE 文では、1 つのスレッド番号のみを指定できます。デフォルトで現行インスタンスのスレッド番号が選択されている場合は、THREAD 句を指定する必要があります。

> **参照：** REDO スレッドの詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

スレッドの使用禁止

パブリックまたはプライベート・スレッドを使用禁止にするには、ALTER DATABASE DISABLE THREAD 文を使用します。スレッドを使用するインスタンスがデータベースをマウントしている場合、その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変更するか、またはその逆の場合、スレッドを使用禁止にしてから、再度使用可能にする必要があります。インスタンスは、インスタンス自体のスレッドを使用禁止にできません。スレッドを使用可能または使用禁止にする場合、データベースはオープンされている必要があります。

スレッドを使用禁止にする場合、Oracle は、現行の REDO ログ・ファイルをアーカイブが必要であるとマークします。そのファイルを削除するには、まず、手動でそのファイルをアーカイブする必要があります。

スレッドを使用可能にする際に、エラーまたは障害が発生すると、現行ログ・ファイルの集合を持っていても、使用可能ではないスレッドになる可能性があります。これらのログ・ファイルを削除またはアーカイブすることはできません。この場合、すでに使用禁止の状態でもスレッドを使用禁止にし、再度使用可能にします。

ログのモードの設定

REDO ログを使用する ARCHIVELOG モードまたは NOARCHIVELOG モードは、データベース作成の際に設定します。アーカイブ・モードは、SQL 文 ALTER DATABASE でも変更できますが、これが必要となることはほとんどありません。アーカイブが使用可能の場合、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、Oracle Parallel Server を使用禁止にしてデータベースをマウントする必要があります。データベースはオープンできません。

REDO ログ・モードは、個々のインスタンスではなく、データベースに対応付けられます。ARCHIVELOG モードで REDO ログが使用されている場合、すべてのインスタンスは、ほとんどの場合、同じアーカイブ・メソッド（自動または手動のいずれか）を使用する必要があります。

REDO ログの変更

Oracle Parallel Server を使用可能または使用禁止にしてデータベースがマウントされているとき、ログ・ファイルまたはログ・ファイル・メンバーの追加、削除、改名などを行い、REDO ログの構成を変更できます。ただし、いずれかのスレッドで現在使用中のログ・ファイルまたはログ・ファイル・メンバーを削除または改名することはできません。さらに、スレッドのログ・グループ数が 1 つになってしまう場合は、ログ・ファイルを削除できません。

すべてのインスタンスは、他のどのインスタンスのどのグループの REDO ログ・ファイルまたはメンバーでも追加または改名できます。インスタンスに 2 つ以上のグループがあれば、他のどのインスタンスでも REDO ログ・グループをインスタンスから削除できます。REDO ログ・ファイルおよびログ・メンバーへの変更は、次のログ・スイッチで有効になります。

参照： 13-2 ページの「REDO ログ・ファイルのアーカイブ」を参照してください。

追加データ・ファイルへのロックの提供

Oracle Parallel Server が実行中にファイルを追加した場合、新しいファイルを処理するために使用可能なロックが十分かどうかを評価します。このように追加されたデータ・ファイルは、未割当てのロックを使用しますが、それらのロックは、Oracle Parallel Server が最初に GC_FILES_TO_LOCKS の値に適応するようにロックを作成したときに作成され、まだ割り当てられていないロックです。

残りのロック数では、新しいファイルの保護および競合回避に不十分な場合、GC_FILES_TO_LOCKS パラメータの値を増やして、より多くのロックを提供します。これらの調整を行わないと、パフォーマンスが低下する場合があります。この問題は、特に、データベースに対して何度も挿入が実行された場合に発生します。ただし、読込み専用データベースにおいては、多くの新しいデータ・ファイルを追加した場合でも付加的なロックは必要ありません。

ロックの数を増やす必要がある場合は、次の手順に従ってください。

1. データベースを停止します。
2. GC_FILES_TO_LOCKS 初期化パラメータを変更して、追加のデータ・ファイルに十分なロックを提供します。
3. システムを再起動します。

参照： 9-6 ページの「GC_FILES_TO_LOCKS の設定に対するヒント」を参照してください。

CREATE DATABASE オプションの値の変更

CREATE CONTROLFILE 文を使用して、データベースの次のデータベース・パラメータを変更できます。

- MAXINSTANCES
- MAXLOGFILES
- MAXLOGMEMBERS
- MAXLOGHISTORY
- MAXDATAFILES

参照： CREATE CONTROLFILE 文および ALTER DATABASE BACKUP CONTROLFILE TO TRACE 文の説明は、『Oracle8*i* SQL リファレンス』を参照してください。

4

インスタンスの管理

この章では、インスタンスの起動および停止について説明します。内容は次のとおりです。

- インスタンスの起動
- インスタンスの停止
- インスタンスへの SQL*Plus および SQL の適用方法

インスタンスの起動および停止

この項では、次のトピックについて説明します。

インスタンスの起動

この項では、次の項目について説明し、Oracle Parallel Server を使用可能または使用禁止にしてインスタンスを起動する手順を示します。

Oracle Parallel Server の使用可能化およびインスタンスの起動

インスタンスを起動するには、『Oracle8*i* Parallel Server セットアップおよび構成ガイド』で説明するとおり、SQL*Plus または Oracle Enterprise Manager のいずれかを使用します。

注意： Oracle8*i* では、SHARED、EXCLUSIVE および PARALLEL の各キーワードは、STARTUP 文および ALTER DATABASE MOUNT 文で使用不可になりました。

SQL*Plus を使用したインスタンスの起動

Oracle Parallel Server インスタンスの起動方法は、単一インスタンスを起動する場合と同じです。ただし、次の処理も必要です。

1. インスタンス初期化ファイルで PARALLEL_SERVER パラメータが TRUE に設定されていることを確認します。
2. Cluster Manager ソフトウェアが実行されていることを確認します。Cluster Manager ソフトウェア管理の詳細は、オペレーティング・システム固有のドキュメントを参照してください。Cluster Manager が使用できない場合、または Oracle がこのコンポーネントと通信できない場合は、エラー・メッセージの「ORA-29701: Cluster Manager に接続できません。」が表示されます。

   参照： Windows NT での Cluster Manager については、ベンダーのドキュメントを参照してください。

3. オペレーティング・システム固有の必要なプロセスを実行します。これらのプロセスの詳細は、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。

データベースの作成に Oracle Database Configuration Assistant を使用する場合は、初期化ファイルの PARALLEL_SERVER を TRUE に設定して、Oracle Parallel Server データベースを作成するように要求してください。Oracle Parallel Server を使用禁止にしてデータベースを起動する場合は、インスタンスの PARALLEL_SERVER パラメータにデフォルト値の FALSE を使用します。

> **参照：** Cluster Manager の詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

RETRY を使用した共有モードでのデータベースのマウント 他のインスタンスが同じデータベースをリカバリ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、リカバリが完了するまでは、新しいインスタンスはデータベースをマウントできません。

インスタンスの起動を繰り返し実行するかわりに、STARTUP RETRY 文を使用します。これによって、再試行が成功するか、再試行制限に達するまで、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。次の構文を使用します。

```
STARTUP OPEN database_name RETRY
```

インスタンスがデータベースをマウントしようとする最大回数を設定するには、SQL*Plus の SET コマンドに RETRY オプション指定して使用します。整数（たとえば 10）またはキーワード INFINITE を指定できます。

他のインスタンスによるリカバリでしかデータベースがオープンできない場合、RETRY は接続の試行を繰り返しません。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は、他のインスタンスが共有モードで STARTUP RETRY コマンドを実行しても機能しません。

インスタンスの設定および接続

インスタンスを設定してそれらに接続する前に、Oracle Parallel Server ノードおよびそれらのノードにアクセスするすべてのクライアントに、Net8 をインストールおよび構成する必要があります。これによって、クライアントはノードにリモート接続を確立できます。

> **参照：**
>
> - 『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。
> - 『Oracle8*i* Net8 管理者ガイド』を参照してください。

SQL*Plus コマンドは、次の項に示すいくつかの場合を除いて、現行インスタンスで処理されます。現行インスタンスは、SQL*Plus セッションを開始するローカルのデフォルト・インスタンス、またはリモート・インスタンスです。SQL*Plus プロンプトでは、どのインスタンスが現行インスタンスであるかは示されないため、適切なインスタンスにコマンドを発行する必要があります。

SQL*Plus セッションを開始して、インスタンスを指定せずにデータベースに接続すると、すべての SQL*Plus コマンドは、ローカル・インスタンスで処理されます。この場合も、デフォルトのインスタンスが現行インスタンスです。

現行インスタンスをローカル・インスタンスからリモート・インスタンスに切り替えるには、次のいずれかを実行します。

- リモート・インスタンス・ネット・サービス名を指定して、次の CONNECT コマンドを再実行します。

```
CONNECT SYSTEM/MANAGER@net_service_name
```

- データベースからの接続を切断し、次の SET INSTANCE コマンドを実行します。

```
SET INSTANCE net_service_name
```

その後、ユーザー ID およびパスワードのみを含む CONNECT コマンドを実行します。あるインスタンスを経由してデータベースに接続しながら、CONNECT コマンドでリモート・インスタンスを指定すると、切断せずに 1 つのインスタンスから他のインスタンスに切り替えることができます。

参照 :

- ネット・サービス名の構成方法の詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。
- SET INSTANCE コマンドおよび CONNECT コマンドで使用する接続文字列の完全な書式については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

SET INSTANCE コマンドおよび SHOW INSTANCE コマンド

SET INSTANCE を使用して、STARTUP コマンドで指定するリモート・ノード上のインスタンスを指定する場合、リモート・インスタンスのパラメータ・ファイルは、ローカル・ノードからアクセス可能である必要があります。

SHOW INSTANCE コマンドは、現行インスタンスのネット・サービス名を表示します。SQL*Plus セッション中に SET INSTANCE を使用しなかった場合、SHOW INSTANCE は、LOCAL 値を戻します。

デフォルト・インスタンスにリセットするには、ネット・サービス名を指定せずに SET INSTANCE を使用するか、または LOCAL を指定します。SET INSTANCE の後に DEFAULT という単語は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の接続文字列を示します。

CONNECT コマンド

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 SQL*Plus セッションが、同時に同じインスタンスに接続できます。他のインスタンスに接続すると、SQL*Plus によって最初のインスタンスとの接続が自動的に切断されます。

参照 :

- *net_service_name* の適切な仕様については、『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。
- SYSDBA または SYSOPER の権限でデータベースに接続する方法については、『Oracle8*i* 管理者ガイド』を参照してください。

インスタンスの停止

Oracle Parallel Server インスタンスの停止方法は、単一インスタンスを停止する場合と同じです。ただし、次の例外があります。

- Parallel Server では、1 つのインスタンスを停止しても他の実行中のインスタンスの操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には、Parallel Server クラスタ内のすべてのインスタンスを停止します。
- インスタンスが異常終了した場合、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、データベースにアクセスしている場合、Oracle は、そのアクセスを終了させ、「ORA-01092: 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表示します。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していない場合、Oracle は、次のコール時または Oracle への要求実行時に、「ORA-01012: ログオンされていません。」というメッセージを表示します。
- NORMAL または IMMEDIATE での停止後は、インスタンス・リカバリは不要です。ただし、SHUTDOWN ABORT コマンドを実行した後、またはインスタンスが異常終了した後は、リカバリが必要です。まだ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・リカバリを実行します。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、リカバリが必要なすべてのインスタンスにインスタンス・リカバリを実行します。
- 複数の SQL*Plus セッションが同じインスタンスに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 つを除くすべてのセッションを切断しておく必要

があります。複数の SQL*Plus セッション（または他のセッション）がインスタンスに接続されている場合は、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。

> **参照：** Oracle データベースを停止する方法の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

インスタンスへの SQL*Plus および SQL の適用方法

この項の内容は次のとおりです。

インスタンスへの SQL*Plus コマンドの適用方法

表 4-1 に、共通 SQL*Plus コマンドのインスタンスへの適用方法を示します。

表 4-1 インスタンスへの SQL*Plus コマンドの適用方法

SQL*Plus コマンド	SQL*Plus コマンドが適用されるインスタンス
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	現行インスタンスおよびデフォルト・インスタンスの位置に関係なく、SQL*Plus セッションを実行中のノードに適用されます。
RECOVER	特定のインスタンスではなく、データベースに適用されます。
SHOW INSTANCE	現行インスタンスについての情報を表示します。リモート・インスタンスにコマンドをリダイレクトしている場合、現行インスタンスはデフォルトのローカル・インスタンスではない場合があります。
SHOW PARAMETER および SHOWN SGA	現行インスタンスに関するパラメータおよび SGA 情報を表示します。
STARTUP および 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 SQL*Plus コマンドです。

注意： 権限付きの SQL*Plus コマンドの実行時に Oracle が使用するセキュリティ・メカニズムは、ご使用のオペレーティング・システムによって異なります。ほとんどのオペレーティング・システムは、オペレーティング・システムへのログイン時用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、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定されます。詳細は、ご使用のオペレーティング・システム固有のドキュメントを参照してください。

インスタンスへの SQL 文の適用方法

ほとんどの SQL 文は現行インスタンスに適応されます。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のみ適用されます。

ALTER SYSTEM CHECKPOINT LOCAL も、現行インスタンスに適用されます。それとは対称的に、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のみ適用されます。グローバル・ログ・スイッチを強制するには、ALTER SYSTEM ARCHIVE LOG CURRENT 文を使用します。ALTER SYSTEM ARCHIVE LOG の THREAD オプションを使用すると、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。

第III部

Oracle Parallel Server の設計および配置

第 III 部では、Oracle Parallel Server アプリケーションの配置に関する設計上の問題について説明します。第 III 部に含まれる章は、次のとおりです。

- 第 5 章「アプリケーションの分析およびパーティション化」
- 第 6 章「データベースの設計方法」
- 第 7 章「PCM および非 PCM インスタンス・ロックの使用計画」
- 第 8 章「空きリスト・グループによるデータの パーティション化」
- 第 9 章「インスタンス・ロックの設定」
- 第 10 章「ロックおよびリソースに対する DLM 容量の確保」

5

アプリケーションの分析およびパーティション化

この章では、Oracle Parallel Server のアプリケーション設計を最適化する方法について説明します。内容は次のとおりです。

注意： アプリケーションをパーティション化できないと判断した場合、第 8 章を参照してください。

開発方法の概要

Oracle Parallel Server にアプリケーションを配置する場合は、特別な項目を考慮する必要があります。また、単一インスタンス環境でアプリケーションを開発するために使用する方法以外に、特別な方法を使用する必要があります。この章では、これらの問題について説明し、Parallel Server ベースのアプリケーションを配置して、Oracle Parallel Server の機能を最適化するための高レベルな開発方法を示します。

単一インスタンスに対するパフォーマンスの最適化が不十分な、不完全なアプリケーションのパフォーマンスは、Oracle Parallel Server 環境では、さらに低下する可能性があります。単一インスタンスから Oracle Parallel Server へアプリケーションを移動する前に、できるだけチューニングを行います。適切にチューニングされたアプリケーションでも、Parallel Server 上ではパフォーマンスが低下する場合があることに注意してください。これは、通常、過剰な ping によるものです。

注意： この章で説明する、Oracle Parallel Server アプリケーションの配置方法の情報を参照してください。また、Oracle8*i* 開発者用マニュアル・セットで説明されている、単一インスタンスのアプリケーションの配置の情報も参照してください。

開発前に行うアプリケーションの適性判断

Oracle Parallel Server アプリケーションを開発する前に、システムが複数インスタンス環境に適しているかどうかを判断します。一般に、データを簡単にパーティション化してインスタンス間の競合を回避できる場合、Oracle Parallel Server はご使用のシステムに適したソリューションになると判断できます。

Oracle Parallel Server 上に配置するのに最適なアプリケーションは、意思決定支援システム（DSS）などの読込み専用表のみを使用するタイプです。これらのアプリケーションは、完全にパーティション化できるため、インスタンス間の競合を最小限に抑えられます。

ただし、ほとんどの OLTP アプリケーションは、様々な程度にパーティション化できます。キャッシュ・フュージョンの導入によって、厳密なデータおよびインスタンス間の親和性を作成するためのデータのパーティション化ができない場合でも、パフォーマンスが大幅に向上することがあります。ただし、パーティション・アプリケーションは、非パーティション・アプリケーションよりスケーラブルです。

注意： 単純で更新可能なスター・スキーマおよびパーティション化できないアプリケーションでは、Oracle Parallel Server に配置すると、パフォーマンスが低下する場合があります。

分析の詳細さの判断

Oracle Parallel Server を使用して全体のデータベース・スループットを改善するには、データベースの設計およびアプリケーションの作業負荷に対する詳細な分析を行います。これによって、追加ノードによってアプリケーション処理に対して提供される、追加の処理能力を完全に利用できることが保証されます。高可用性を得るためにのみ Oracle Parallel Server を使用する場合でも、慎重に分析することによって、システム・リソースの要件がより適切に予測できるようになります。

高パフォーマンスの Oracle Parallel Server システムの主な特性は、パラレル・キャッシュ管理に使用される計算リソースを最小化することです。これは、インスタンス・ロック・オペレーションの数を最小化することを意味します。さらに、マシン間の高速インターコネクトの通信量が、クラスタの設計制限の範囲内、もしくはかなり低い量にとどまります。

アプリケーション・トランザクションおよび表アクセス・パターン

分析を始める前に、次のようなトランザクション・タイプに基づいた表内で、Oracle Parallel Server が様々なトランザクションをどのように処理するかを理解する必要があります。

- 読込み専用表
- ランダム SELECT および UPDATE 表
- INSERT、UPDATE または DELETE 表

読込み専用表

主に読込み専用表内では、すべての Oracle Parallel Server ノードは、PCM ロックを迅速に共有モードに初期化するため、ロック・アクティビティがほとんど発生しません。理想的には、各読込み専用表およびそれに対応付けられた索引構造に必要な PCM ロックは 1 つのみです。このため、Oracle Prallel Server に読込み専用表を使用すると、優れたパフォーマンスおよびスケーラビリティが得られます。

また、SQL 文 ALTER TABLESPACE READ ONLY を使用して、読込み専用表を読込み専用表領域に入れることもできます。これには、次のような効果があります。

- リカバリが高速化されます。
- PCM ロックが必要ありません。
- 読込み専用にした後は、表領域のバックアップが 1 回しか必要ありません。

Oracle Parallel Server でのパラレル実行のスケーラビリティは、ノード間のインターコネクトの処理速度に左右されます。また、プロセッサをビジー状態に保つためのみに、高い並列度を使用することが必要な場合もあります。ノードまたはプロセッサ数の 3 倍の並列度を実行することも珍しくありません。

参照： 並列度の設定の詳細は、『Oracle8*i* データ・ウェアハウス』を参照してください。

ランダム SELECT および UPDATE 表

ランダム SELECT および UPDATE 表には、表の任意の行を読み込み、その後更新するトランザクションがあります。このタイプのアクセスには、複数のロック変換が必要です。まず、このトランザクションを実行するインスタンスは、1 つ以上のデータ・ブロック上に対する共有 PCM ロックを取得する必要があります。このロック要求によって、別のノード上でロック・ダウングレード操作が発生する場合があります。トランザクションを実行するインスタンスは、UPDATE が実際に実行されたときに、最終的に排他モード PCM ロックを取得する必要があります。

別々のノード上のユーザー・トランザクションが、同時および頻繁に同じ範囲のデータ・ブロックを変更する場合、応答時間のパフォーマンスが大幅に低下する場合があります。ロックの粒度またはアクセスの頻度を制御して、競合を削減できる場合もあります。

ただし、大きい表では、ハードウェア上の制限および実用上の制限から、効率的に使用できる固定 PCM ロックの数が制限される場合があります。このような場合は、解放可能ロックを選択すると効率的な場合があります。

参照： A-17 ページの事例にある「高粒度ロックまたは低粒度ロックの実装」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 表上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、読み込まれたデータ・ブロックの一部またはすべてを変更する必要があります。この処理を指定された各データ・ブロックに行う場合、再び PCM ロックを共有モードに変換して、ブロック変更時にそれを排他モードに変換する必要があります。この処理には、前の項で説明したランダム SELECT および UPDATE 表と同じパフォーマンス上の問題があります。

ランダムに変更される表に対するパフォーマンス上の問題は、索引を更新またはメンテナンスする必要がある場合に発生することがあります。この問題は、特に、挿入が実行されたとき、または Oracle が空き領域を検索し、それを表または索引に割り当てるときに発生します。

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。この処理には、ルート・ブロック、ブランチ・ブロック、リーフ・ブロックなどの複数の索引ブロックへのアクセスが必要です。したがって、潜在的なロック変換の数が増加します。索引のブランチ・ブロックまたはリーフ・ブロックが分割されるため、複数のロックを同時に行う必要があります。これによって、インスタンス間の競合の可能性が高くなります。索引内にキー値を分散することは、非常に重

要です。単調に増加する索引キーでは、索引キーの右端にホット・スポットが作成される場合があります。

INSERT 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、トランザクションの完了のために読取り一貫性情報が別のインスタンスで必要になる可能性が高くなります。このタイプの同時実行は、読込みに対するキャッシュ・フュージョンによって効率的に処理されます。

シーケンス・ジェネレータを使用して、複数の Oracle Parallel Server ノードから 1 つの表に対して一意キーを生成する場合、索引ブロックの競合が問題になる場合があります。一意キーを生成する場合、主キーの一部にインスタンス番号を使用して、各インスタンスが索引の別の部分に INSERT を実行するようにします。INSERT のロードを索引の全域に分散させると、単一および複数インスタンスのパフォーマンスが向上します。これは、逆キー索引を使用して行います。

逆キー索引の作成

逆キー索引を作成すると、索引への変更がリーフ・ブロックの小さい集合に集中する、Oracle Parallel Server 環境でのパフォーマンスが向上します。逆キー索引は、列順序を保持しながら、各索引列（ROWID 以外）のバイトを逆にします。索引のキーを逆にすることによって、索引内のすべてのリーフ・キーに挿入が分散されます。

たとえば、次の構文を使用して逆キー索引を作成します。

```
CREATE INDEX i ON t (a,b) REVERSE;
```

ここで、a はインスタンス番号を、b は生成された一意キーを示します。

INSERT 操作では、エクステント内での空き領域の割当てが、高いロック変換率の原因になる場合もあります。これは、複数インスタンスが、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ためです。これらのデータ・ブロックが、同じ PCM ロックによって管理されている場合に競合が発生します。これを回避するには、表および索引をパーティション化して複数のインスタンスがそれらを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複数の空きリスト・グループが使用できるようにします。

参照：

- 第 8 章を参照してください。
- 『Oracle8*i* 概要』も参照してください。

パーティション化方法の選択

Oracle Prallel Server を最適化するアプリケーションを実装するには、次のいずれかのパーティション化方法を使用します。

- アプリケーションのパーティション化の方法
- 部門パーティション化およびユーザー・パーティション化
- 物理表のパーティション化
- トランザクションのパーティション化

パーティション化は、グローバル・データ・ブロックの競合または ping を回避する最適な方法であるため、Oracle Parallel Server の配置における重要な部分です。キャッシュ・フュージョンでは、ディスクへの強制書込みの数が大幅に減少するため、パーティション化はあまり重要ではありません。

Oracle Parallel Server のパフォーマンスを向上させるための表のパーティション化には、開発および管理との様々な関係があります。開発の観点からすると、表をパーティション化すると、その表に必要なアプリケーション・コードの量および複雑さが増加します。さらに、表をパーティション化することによって、バッチやデータ・ウェアハウス問合せなどの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場合があります。

システムのパフォーマンスへの影響を理解し、パーティション化による設計上のトレードオフにも注意する必要があります。目標は、同期化の必要性を最小化することです。ロック変換を最小化することで分散ロック・マネージャ・アクティビティが減少すると、Oracle Parallel Server のパフォーマンスが予測可能およびスケーラブルになります。

アプリケーションおよびデータをパーティション化することによって、データおよびノード間の親和性をメンテナンスできます。システムに分散ロック・マネージャの過剰なロック・アクティビティが発生した場合、パーティション化の方法が適切でないか、またはデータベースの作成およびチューニング・プロセスが効果的でない可能性があります。

次の項で説明するように、使用する方法にかかわらず、その方法は、最適な結果を得るためにデータ中心である必要があります。

機能ではなくデータに基づくパーティション化

パーティション化の方法は、アプリケーションの機能ではなく、データおよびそのデータの使用方法に注目して決定します。パーティション化を行う場合、ロード・バランスが主な目標であるとは限りません。

パーティション化方法を決定するには、たとえば、場所、タイプ、頻度などのビジネス機能のデータ・アクセスのプロパティを調べます。それらを少数の主要なカテゴリにグループ化し、ロックの方法および構成を決定します。

この方法でパーティション化を設定すると、すべてのインスタンスにデータ・ブロックおよびキャッシュ間の親和性が作成されます。また、分散ロック・マネージャのアクティビティがより予測可能になるため、次のことがより簡単になります。

- 最小のロック割当てでの高可用性
- 最小のロック変換でのさらなるスピードアップおよびスケールアップ

この章で説明する方法で十分なパフォーマンスを得られない場合、次のいずれかを行うことを考慮してください。

- ロック・グループの作成
- 解放可能ロックの使用
- 空きリストの使用によるデータのパーティション化（第 8 章を参照）

1 回のロックで処理するブロックの数が多くなるほど、アプリケーションで過剰な false ping が発生する可能性が高くなることに注意してください。

アプリケーションのパーティション化の方法

データベースのロードをパーティション化する最も簡単な方法の 1 つは、同じデータベースにアクセスするアプリケーションのサブコンポーネントを、クラスタの別々のノードで実行することです。たとえば、あるサブコンポーネントは、データ・ファイルの 1 つの集合に常駐する同じ表の集合のみを参照し、別のアプリケーションは、データ・ファイルの別の集合に常駐する別の表を参照するとします。

この場合、これら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。さらに次の効果があります。

- 同じデータベース・オブジェクトに対する競合がほとんどありません。
- データ・ブロックおよびインスタンス間の親和性が高くなります。
- 各サブコンポーネントが使用する分割データの集合によって、グローバル競合が減少します。

この方法は、特に、日中に多くのユーザーおよび作業負荷の高い OLTP をサポートし、夜間には作業負荷の高いバッチおよび意思決定支援を実行するアプリケーションに対して適用できます。この場合、クラスタのノード間でアプリケーションをパーティション化し、日中に効率のよい OLTP パフォーマンスを確保できます。

この例は、同時にアクセスされる表をまとめて格納する分散データベース・モデルに似ています。夜間に、OLTP の目的でパーティション化される表にアクセスする必要がある場合にも、単一データベースのメリットを利用できます。すべてのデータは、単一データベース内に効率的に格納されます。これによって、バッチおよび意思決定支援のパフォーマンスが向上し、SQL 全体のパフォーマンスが向上します。また、ネットワーク通信量が削減し、データ・レプリケーションの問題が減少します。

この方法では、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してください。そうでない場合、パーティション化のメリットが失われます。これを行うには、各アプリケーションの表および索引データを別々のデータ・ファイル内に格納します。

SQL 文を共有しているアプリケーションは、同じインスタンス上で実行する場合にパフォーマンスが最高になります。共有 SQL 領域はインスタンス間で共有されないため、類似した一連の SQL 文は、メモリー使用量を改善し、解析を削減するために、1 つのインスタンス上で実行する必要があります。

アプリケーションのパーティション化の方法

この項では、アプリケーションをパーティション化するための、次の 5 つのステップについて説明します。

- ステップ 1: システムの主な機能領域を定義する
- ステップ 2: 表アクセスの要件を識別し、オーバーラップを定義する
- ステップ 3: 各オーバーラップに対するアクセス・タイプを定義する
- ステップ 4: トランザクション・ボリュームを識別する
- ステップ 5: オーバーラップを分類する

ステップ 1: システムの主な機能領域を定義する

アプリケーションの主な機能を識別します。たとえば、ある大手のホテル・チェーンで、次の高レベルの機能を自動化するシステムを開発している場合があります。

- 予約
- プロパティの管理およびメンテナンス
- セールスおよびマーケティング
- フロント・デスク、コンセルジュおよびダイニング施設の管理

また、どのユーザーが各機能領域からデータにアクセスするかを判断します。

ステップ 2: 表アクセスの要件を識別し、オーバーラップを定義する

各機能領域がどの表にアクセスするかを判断し、オーバーラップを識別します。オーバーラップとは、単に、複数の機能領域からユーザーがアクセスする表です。表 5-1 に、この例でオーバーラップしている表を太字で示します。残りの表は、列のヘッダーに示されている機能によって、排他的にアクセスされます。

表 5-1 オーバーラップしている表の例

ホテルの予約操作	フロント・デスクの操作
表 1	表 12
表 7	表 14
表 15	**表 15**
表 11	表 16
表 19	**表 19**
表 20	**表 20**

目標は、グローバル競合の原因になり、アプリケーションのパフォーマンスに悪影響を及ぼすオーバーラップを識別することです。この例では、両方の機能が同時に 3 つの表にアクセスします。残りの表は排他的アクセスされているため、必要なロックはさらに少なくなります。

ステップ 3: 各オーバーラップに対するアクセス・タイプを定義する

各オーバーラップに対するアクセス・タイプを判断します。

表 5-2 表アクセス・タイプの例

ホテルの予約の操作	予約によるオーバーラップ・アクセス・タイプ	オーバーラップ	フロント・デスクによるオーバーラップ・アクセス・タイプ	フロント・デスクの操作
表 1	S（選択）	表 15	S	表 12
表 7	I（挿入）	表 19	I	表 14
表 11	U（更新）	表 20	U	表 16

この例では、両方の機能が表に対して次のようにアクセスします。

- 選択の場合、表 15
- 挿入の場合、表 19
- 更新の場合、表 20

ステップ 4: トランザクション・ボリュームを識別する

オーバーラップによって生成されると予測されるトランザクションの数を見積ります。

表 5-3 表トランザクション・ボリュームの例

ホテルの予約の操作	予約によるトランザクション・オーバーラップ	オーバーラップ	フロント・デスクによるトランザクション・オーバーラップ	フロント・デスクの操作
表 1	S（毎秒 10）	表 15	S（毎秒 50）	表 12
表 7	I（毎秒 100）	表 19	I（毎秒 10）	表 14
表 11	U（毎秒 50）	表 20	U（毎秒 90）	表 16

これらのトランザクション・ボリュームによって、オーバーラップ表にパフォーマンスの問題が発生する場合があります。ただし、アプリケーションが表にほとんどアクセスしない場合、表 5-3 に示すボリュームは問題にはならない場合があります。

ステップ 5: オーバーラップを分類する

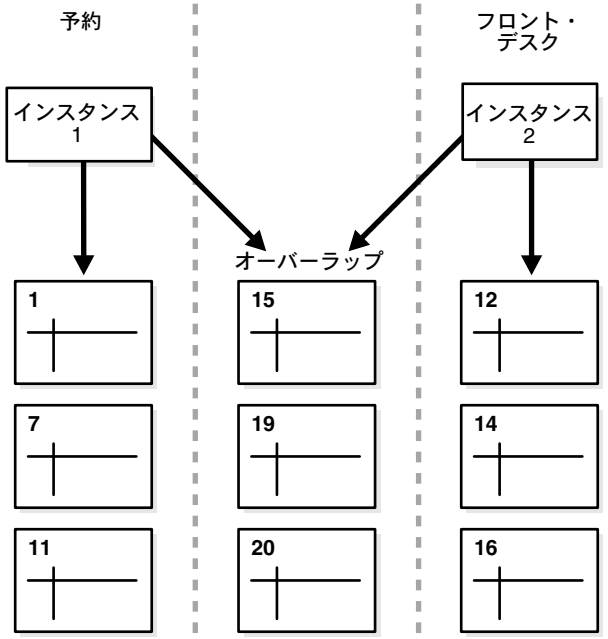
次の基準を使用して、表の処理方法を判断します。

- オーバーラップしていない表、選択専用のオーバーラップおよび低頻度のオーバーラップは無視します。
- 索引構成表およびそれらの索引を分類します。
- 複合アクセス表およびそれらの索引を分類します。

索引構成表およびそれらの索引 これらの表には、インスタンスごとにエクステントを割り当て、排他ロックを取得することができます。これらの表およびその索引に対する競合は、Oracle Parallel Server では比較的簡単に解消できます。

得られた構成例 図 5-1 に、表の構成を示します。

図 5-1 アプリケーション・パーティション化のオーバーラップの分類



アプリケーション・パーティション化を使用してこれらの表内の競合を解消できない場合は、次の項で説明する部門パーティション化を検討します。

部門パーティション化およびユーザー・パーティション化

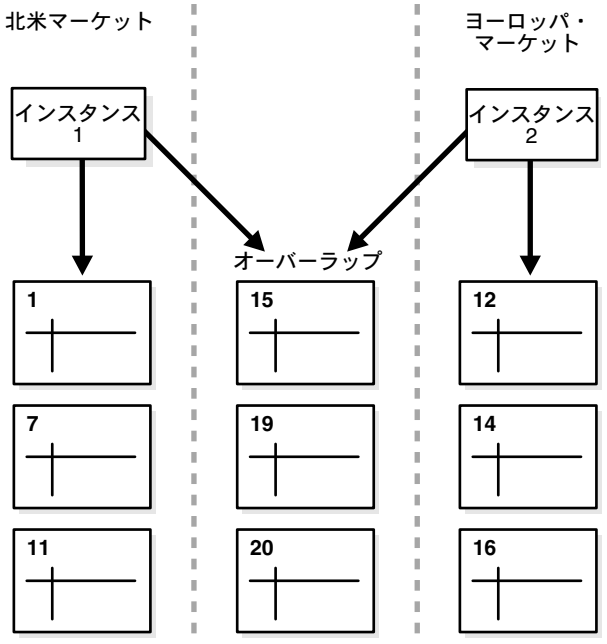
競合の最小化に有効なもう１つのパーティション化の方法は、部門パーティション化またはユーザー・パーティション化です。部門パーティション化およびユーザー・パーティション化の実装には、いくつかの方法があります。

部門パーティション化の１つに、地理的な場所に基づくアクセス・グループごとに表を分割する方法があります。たとえば、ホテルの予約システムで、世界中のホテルに対する客室予約を処理すると想定します。この場合、次のような地理的なマーケットごとにアプリケーションをパーティション化する場合があります。

- ヨーロッパ・マーケット
- 北米マーケット
- 中南米マーケット
- アジア太平洋マーケット

この構成は、図 5-2 に示すパーティション化に似ています。

図 5-2 地理的パーティション化のオーバーラップ



地理的パーティション化に加えて、Oracle8*i* Enterprise Edition の Oracle Partitioning Option を使用することもできます。このオプションには、次の 3 つの表パーティション化メソッドが含まれます。

- レンジ
- ハッシュ
- コンポジット

各メソッドには、それぞれのメリットおよびデメリットがあります。したがって、各メソッドは、特定の状況には適していても、その他の状況には適さない場合があります。

レンジ・パーティション化 レンジ・パーティション化は、各パーティションに設定した列値の範囲によって規定された境界に基づいて、データをパーティションにマップします。この機能は、一般に、意思決定支援システム・アプリケーションにのみ有効です。

ハッシュ・パーティション化 ハッシュ・パーティション化は、Oracle がパーティション化キーに適用するハッシング・アルゴリズムに基づいて、データをパーティションにマップします。この機能は、主に DSS アプリケーションに有効です。

コンポジット・パーティション化 コンポジット・パーティション化は、レンジ・パーティション化およびハッシュ・パーティション化の機能を組み合せます。コンポジット・パーティション化では、Oracle は、まず、パーティション・レンジの始点および終点ごとに設定された境界に基づいて、データをパーティションに分散させます。その後、Oracle は、さらにデータを分割し、ハッシング・アルゴリズムを使用して各パーティション・レンジ内のサブパーティションに分散させます。

> **参照：** パーティション化の詳細は、『Oracle8*i* データ・ウェアハウス』を参照してください。

次に説明する 2 つのパーティション化の方法は、プログラミングに時間がかかるため、前述の方法ではパフォーマンスが最適化されない場合にのみ使用してください。

物理表のパーティション化

物理表のパーティション化では、1 つの表を 2 つ以上の小さい表に分割します。これには、小さい表の新しい名前を使用するようにアプリケーションを変更する必要があります。レポート・プログラムも、必要に応じて小さい表を結合できるように変更する必要があります。

ただし、Oracle はパーティションの独立性を自動的に管理できます。つまり、アプリケーション内のすべての小さい表に同じ名前を使用した場合、データは自動的に分離されます。

リソースが十分な場合は、トランザクションのパーティション化を使用できます。ただし、この方法は非常に複雑で、前述したどの方法より実装に時間がかかります。

トランザクションのパーティション化

トランザクションのパーティション化は、最も低レベルのパーティション化方法です。この方法には、3 層アーキテクチャが必要です。ここでは、クライアントおよびサーバーが、トランザクション・モニター処理レイヤーによって分割されます。トランザクションの内容に基づいて、トランザクション・モニターは、特定の表で動作するトランザクションを特定のノードに送ります。この方法では、トランザクション・モニターが、どのノードが特定のトランザクションを処理するかを決定するため、どの方法を使用しても表を作成およびロードできます。

この方法では、ファイングレイン・トランザクション・コントロールも実現します。これによって、トランザクション・プロセス・モニターが非常にスケーラブルになります。ただし、この方法を配置するには、非常に多くの開発工数が必要です。

トランザクションを実行する正しいノードは、そのトランザクションで使用されている実際の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いわれます。

データ依存ルーティングは、次の 2 つのいずれかの方法で実現します。表のパーティション化が、実際のパーティション使用のパターンによく適している（表を状況またはコール・センターごとにパーティション化し、ユーザーを同じようにパーティション化できる）場合、関連するアプリケーションを実行しているインスタンスにユーザーを接続させることによって、手動ルーティングができます。そうでない場合、データ依存ルーティングの管理が複雑になり、追加のアプリケーション・コードが必要になります。

このプロセスを単純にするには、アプリケーションがトランザクション・プロセス・モニター（TPM）またはリモート・プロシージャ・コール（RPC）・メカニズムを使用します。入力 RPC 引数に基づいて、TPM の構成にデータ依存ルーティング方法をコード化できます。同様に、どのインスタンスがトランザクションを実行するかを決定する CASE 文を使用して、このプロセスをプロシージャ・コードにコード化できます。

スケールアップおよびパーティション化

Oracle Parallel Server のアプリケーションを適切にパーティション化している場合、データベースのサイズが増加しても、同じパーティション化方法を維持すると同時に、最適なパフォーマンスを得ることができます。

新しい機能を追加するときに使用する方法は、新機能がアクセスするデータの型によって異なります。機能が分割データにアクセスする場合、既存のパーティション化方法が適切です。新機能が既存の機能と同じデータにアクセスする場合、パーティション化方法の変更が必要な場合があります。

アプリケーションを予測以上のユーザーが使用する場合、インスタンスを追加する必要がある場合があります。新しいインスタンスを追加する場合も、アプリケーションの再パーティション化が必要な場合があります。また、アプリケーションのトランザクション率の増加に対して新しいインスタンスを追加する場合も、再パーティション化が必要な場合があります。

インスタンスの追加

Oracle Parallel Server 環境にインスタンスを追加する前に、新しいインスタンスのデータ・アクセス要件を分析します。新しいインスタンスが、インスタンス独自のデータのサブセット、または既存のインスタンスにアクセスされていないデータにアクセスする場合、現行のパーティション化方法によって、データ競合を適切に回避する必要があります。ただし、新しいインスタンスが既存のデータにアクセスする場合、次の問題を考慮してください。

新しいインスタンスに新しい機能性を追加し、新しい機能性が既存の表にアクセスする必要がある場合、パーティション化方法の変更を検討してください。また、既存のアプリケーションのユーザーの一部を追加のインスタンスに再度割り当てる場合も、パーティション化方法の変更が必要な場合があります。

予測以上の拡張に対応するためのインスタンスの追加によって、複雑で非効率的な再パーティション化作業が必要になる場合があることも考慮する必要があります。このため、計画および設計プロセスの初期段階で、長期間における拡張を考慮する必要があります。

設計に関連するバッチ処理の問題

バッチ操作をスケジューリングする場合、バッチ・ジョブをオンライン処理から分離して、個別のインスタンス上で実行できると考えないでください。バッチ・ジョブは、通常の日常作業の一部と考えてください。

パーティション化方法を開発する場合、アプリケーションのバッチ処理の必要性を考慮します。バッチ・ジョブは、夜間、オフピーク時など、対話ユーザーが少ないときに実行するようにしてください。

注意： 共有表上でフル・スキャンを実行するバッチ・ジョブには注意してください。

DBMS_JOB パッケージの使用によるバッチ・ジョブおよびインスタンス親和性の管理

このパッケージを使用して、どのインスタンスがどのジョブを処理するかを制御します。このパッケージによって、各ジョブの機能を最適化するように、ジョブ処理をクラスタに分散できます。これによって、選択されたインスタンスの SMP プロセスのみでジョブが実行されるため、ロード・バランスが改善され、ブロック競合が制限されます。

たとえば、クラスタ環境内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する場合、Oracle Parallel Server および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 が発生することがあります。表に対するアクティビティを 1 つのインスタンスのみに制限するには、DBMS_JOB を使用して、キュー内にある処理中のジョブの作業を、特定の Oracle Parallel Server インスタンスに割り当てます。

レプリケーションを使用してジョブの親和性について説明している例もありますが、この機能は、他の場合にも使用できます。

参照 :

- DBMS_JOB の詳細は、『Oracle8*i* PL/SQL 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。
- 『Oracle8*i* 管理者ガイド』を参照してください。

6

データベースの設計方法

この章では、Oracle Parallel Server 環境に対するデータベース設計のテクニックについて説明します。内容は次のとおりです。

- Oracle Parallel Server に対するデータベース設計の原理
- データベース処理、ブロック・タイプおよびアクセス制御
- グローバル・キャッシュ一貫作業およびブロック・クラス
- データベース・オブジェクト・パラメータに対する一般的な 推奨事項
- 索引の問題
- 順序番号の使用
- 論理および物理データベース・レイアウト
- グローバル・キャッシュ・ロック割当て
- 結論およびガイドライン

Oracle Parallel Server に対するデータベース設計の原理

共有 Oracle Parallel Server データベースに対してデータベース・レイアウトを設計する場合、グローバルに共有されているデータに複数ノードからアクセスすると、トランザクション・プロセスのコストが増加することに注意してください。つまり、複数ノードのトランザクションでは、単一ノード・システムで処理されるトランザクションより、待機時間が長くなり、CPU 消費も多くなります。このため、スケーラブルなデータベース設計を作成できるように、アプリケーションのデータベース・アクセス特性を慎重に考慮してください。一般に、次の処理を行うことによってスケーラブルなシステムが実現します。

- 類似したデータ・アクセス特性を持つトランザクションを、特定のノードに割り当てます。
- グローバルに共有された場合に、より効率的なアクセスを可能にするパラメータを使用して、データ・オブジェクトを作成します。

これらの原理については、第 5 章で説明します。クラスタ化システムに対して、最もスケーラブルで効率的なアプリケーション設計を行うと、トランザクションがノード上でアクセスするデータへのトランザクションの親和性が高くなります。アプリケーションのデータ・アクセスがローカルで、インスタンス間の同期化の必要がなければ、アプリケーションがより効率的になります。

すべてのシステムには、ノードの親和性が低い、一定の割合のデータがあります。このデータはクラスタ間で共有されるため、同期化が必要です。キャッシュ・フュージョンによって、複数ノード間でのデータの同期化をより効率的に維持できるため、グローバル共有データベースのパーティションに対応付けられたコストが削減されます。これによって、クラスタ内のすべてのノードに対してデータの可用性が向上します。グローバル共有データへの同時アクセスを最適化するための機能がいくつかあります。

あるトランザクションが Oracle Parallel Server 環境で実行する場合、いくつかのデータベース・リソースが重要になります。同じ表内のブロックに対して、頻繁にインスタンス間のアクセスを行うと、I/O、メッセージ表示、コンテキストの切替えおよび一般処理のオーバーヘッドが増加する原因になります。表にメンテナンスする索引が 1 つ以上ある場合、索引アクセスの相対的な複雑さによって、コストがさらに増加する場合があります。空き領域を検索して、新しいデータの挿入時にそれに割り当てるには、セグメント空きリストなどの領域管理構造へのアクセスが必要です。また、Oracle 順序番号を使用して、一意の順序番号を生成すると、クラスタ内のすべてのノードがそれを使用する場合に、重大なボトルネックになる可能性があります。

すべてのリソースは、ディスク上にローカルにキャッシュされるか、または頻繁に再生成される必要があります。INSERT、UPDATE、DELETE、SELECT などのデータベース処理には、アプリケーションが共有モードまたは排他モードのどちらで実行するかによって、異なるタイプのリソースが必要です。

データベース処理、ブロック・タイプおよびアクセス制御

ほとんどのビジネス・トランザクションでは、INSERT、UPDATE、DELETE および SELECT を組み合せて使用します。これらの各トランザクションでこれらの操作を行う正確な割合は、ビジネス・トランザクションのタイプによって異なります。

同様に、これらの各操作は、特定のタイプのデータ・ブロックにアクセスします。これらのブロック・タイプは次のように分類できます。

- データ・ブロック
- 索引ブロック（ルート、ブランチ、リーフ）
- セグメント・ヘッダー・ブロック
- ロールバック・セグメント・ヘッダー・ブロック
- ロールバック・セグメント・ブロック

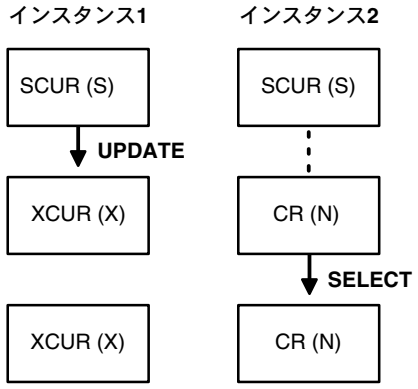
キャッシュ内のデータ・ブロックへの同時アクセスは、アクセス・モードおよびロック・モードによって制御されます。バッファ・キャッシュでは、ブロックは、排他現行（XCUR）、共有現行（SCUR）または一貫読込み（CR）モードでアクセスされます。これらのアクセス・モードは、表 6-1 に示すように、グローバル・キャッシュの一貫性を保証するために、グローバル・ロック・モードにマップします。

表 6-1 パラレル・キャッシュ管理ロック・モードおよびバッファの状態

パラレル・キャッシュ管理ロック・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 EXCLUSIVE 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 SHARED 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

図 6-1 に示すように、各操作は、あるタイプのブロックに特定のモードでアクセスします。

図 6-1 ブロック・アクセスのモード



INSERT 中のブロック・アクセス

Oracle は、INSERT を処理するときに、データベース・オブジェクトのセグメント・ヘッダーを読み込みます。これは、INSERT が新しい行を fit させるために十分な領域があるブロックをセグメントの空きリストから見つけるために、表または索引のセグメント・ヘッダーを読み込む必要があることを表します。したがって、Oracle は INSERT を処理するために、CURRENT またはヘッダー・ブロックの最新バージョンを読み込みます。INSERT の完了後にブロック内に十分な空き領域がある場合、ブロックは空きリスト内に残るので、トランザクションは対応するデータ・ブロックを読み込み、そのブロックに書き込みます。この一連のイベントでは、次のことが行われます。

- セグメント・ヘッダーは SCUR モードで取得されます。これは、インスタンスはグローバル S ロックを要求する必要があることを表します。
- データ・ブロックは XCUR モード、またはグローバルに X モードで取得されます。

挿入後のブロック内の空き領域が不十分な場合、Oracle は空きリストからそのブロックのリンクを解除します。これは、Oracle が、空きリストを含むセグメント・ヘッダー・ブロックを更新することを表します。この一連のイベントでは、次のことが行われます。

1. セグメント・ヘッダー・ブロックが、まず SCUR モード（グローバル S ロック）で取得されます。
2. ブロックのチェック後、バッファ・アクセス・モードは XCUR（グローバル X）に拡張されます。

3. ブロックが空きリストから削除されます。
4. 現行の最高水位標を超えて新しいブロックが使用される場合、最高水位標が上げられます。
5. データ・ブロックは、XCUR モードで読み込まれ、ディスクに書き込まれます。

この場合、セグメントの最高水位標、またはデータを含むセグメント内の最後または最高のブロックが、割り当てられたエクステント内に常駐すると想定しています。空きリスト・グループが定義されていない場合、最高水位標および割り当てられたエクステントのマップは、セグメント・ヘッダー内に格納されます。オブジェクトを挿入するために追加のエクステントを割り当てる必要がある場合、Oracle は、最高水位標を上げ、エクステント・マップを更新します。つまり、Oracle は、セグメント・ヘッダー・ブロックを一貫して変更します。この場合、Oracle は、ヘッダー・ブロックを排他モードでロックする必要もあります。

注意： 前述および後述の説明では、操作に必要なすべてのブロックがメモリー内にキャッシュされると想定しています。

索引がある表に挿入を行う場合、より多くのデータ・ブロック・アクセスが必要です。まず、Oracle は索引セグメントのヘッダー・ブロックを SCUR モードで読み込み、その後ルートおよびブランチ・ブロックを SCUR モードで読み込みます。最後に、Oracle はリーフ・ブロックを XCUR モードで読み込みます。索引ツリーの高さによって、Oracle はより多くのブランチ・ブロックを読み込む必要があります。空きリストの変更が必要な場合、Oracle は、索引セグメント・ヘッダー・ロック・モードを XCUR モードに拡張する必要があります。空きリストの変更によって、セグメント・ヘッダーに同時実行性がある場合、ヘッダー・ブロックは複数インスタンス間を何度も ping することがあります。

表の作成時に FREELIST GROUPS を使用すると、空きリストを効率的にパーティション化できます。オブジェクトに定義する FREELIST GROUPS の数は、INSERT に関わるクラスタ内のノードの数以上にする必要があります。空きリスト・グループでは、空きリストは、ヘッダー・ブロックの後にある個別のブロック内にあります。異なるノードから同じオブジェクトに行を挿入するプロセスは、別の空きリスト・グループ・ブロックにハッシュする必要があります。そのため、空きリスト・グループ・ブロックは、特定のインスタンスに対してローカルのままになります。これは、Oracle がグローバルに競合しているロックを取得する必要がないことを表します。

静的および動的エクステントの割当て

Oracle がエクステントを割り当てて、新しく挿入された行に十分な空き領域を確保するには、2 つの方法があります。1 つの方法では、手動での介入が必要です。もう 1 つの方法では、特定の設定から自動的に導出されます。たとえば、静的割当てでは、次のような文を発行する必要があります。

```
CREATE TABLE
STORAGE (FREELIST GROUPS 2)
ALTER TABLE
ALLOCATE EXTENT
FILE SIZE INSTANCE;
```

この文は、表または索引を作成または変更するときに発行します。このような文によって、特定の物理ファイルから、データ・ブロックのエクステントを空きリスト・グループに割り当て、それによりインスタンスに割り当てることができます。パラメータ INSTANCE_NUMBER を設定して、インスタンスが、空きリスト・グループの特定の割当てを一貫して使用するようにする必要があります。INSTANCE に値を設定しない場合、Oracle は領域をオブジェクトに割り当てます。Oracle は、特定の空きリスト・グループからこの領域を割り当てるのではなく、一般のセグメント・ヘッダー内のマスター空きリストを使用します。

Oracle がこの方法で事前にエクステントを割り当てる場合、Oracle は、連続してブロックを特定の空きリスト・グループに割り当てます。これらのオブジェクトを含むファイルは、ブロッキング要素を持つ 1:N ロックに適しています。つまり、GC_FILES_TO_LOCKS パラメータを設定するとき、およびブロッキング要素を割り当てられたエクステント・サイズに設定するときに、! 構文を使用します。たとえば、1000 ブロックからなるファイル番号 4 のエクステント・サイズが 10 ブロックの場合、次の構文を使用します。

```
GC_FILES_TO_LOCKS= "4=1000!10"
```

これによって、エクステント内のブロックが 1 つのロックによって処理されることが保証されます。

動的割当てでは、前述の構文で示されているように、単純に GC_FILES_TO_LOCKS にブロッキング要素を設定します。領域管理レイヤーでは、*!n* の値に基づいて新しいエクステント・サイズが決定されます。

いくつかの方法を使用してこれを定義できます。たとえば次のとおりです。

```
GC_FILES_TO_LOCKS= "4=1000!10"
```

または

```
GC_FILES_TO_LOCKS= "4=1000!10R"
```

または

```
GC_FILES_TO_LOCKS= "4=0!10"
```

この場合、最初の例では、10 の連続するブロックが、このファイルにプールされた 1000 ロックのうちの 1 つの固定ロックに割り当てられます。2 番目の例では、1 つの解放可能ロックが同じ方法で割り当てられます。また、3 番目の例では、無制限のプール内から複数の解放可能ロックが使用されます。

使用する方法によって、一定数の連続するブロックによって空き領域のエクステントが導出され、これらのブロックは同じロックによって処理されます。使用する方法は、アプリケーションの要件によって異なります。使用しやすさの優勢順位が高く、データ・ファイルが拡張可能である場合、前述の例にある 3 番目の例のように、動的割当てを使用します。ただし、静的割当てには、ランタイム領域管理、必要なデータ・ディクショナリ表および行キャッシュ更新が削減されるというメリットがあります。これは、エクステントがすでに割り当てられているためです。

つまり、Oracle Parallel Server 内で INSERT 集中型トランザクション用に設計する際、パーティション化方法の決定時に、ある表を挿入専用とし場合は、次の処理を行います。

1. 1 つのノードからトランザクションの挿入を実行します。
2. これらの表およびこれらに対応付けられたすべての索引上で、空きリスト・グループを使用します。
3. 領域を動的に割り当てる場合は、GC_FILES_TO_LOCKS パラメータに ! 構文を使用します。
4. CREATE TABLE... ALLOCATE EXTENTS または ALTER TABLE... ALLOCATE EXTENTS 文を使用してエクステントを表または索引に事前に割り当て、エクステントのサイズに GC_FILES_TO_LOCKS パラメータのブロッキング要素 ! を設定します。

UPDATE 中のブロック・アクセス

UPDATE 文では、常に現行バージョンのデータベース・ブロックが読み込まれ、バッファが XCUR モードに設定されます。これは、ブロック上の X ロックに対する要求にグローバルにマップします。すべてのブロックがキャッシュされていると想定すると、トランザクションは次のことを行います。

1. バッファを XCUR モードで読み込み、グローバル X ロックを取得します。
2. バッファに書き込み、行を変更します。
3. 更新された行が同じブロックに入れられた場合、インスタンスは、空きリストから新しいブロックを取得する必要がなく、変更は完了します。セグメント・ヘッダー・アクセスは必要ありません。
4. インスタンスは、別のインスタンスが競合モードでブロック上にロックを要求しない限り、グローバル X ロックを保持します。したがって、Oracle は、他のインスタンスが最新の変更を参照できるように、使用済バッファをディスクに書き込みます。

5. Oracle は、ロックをクローズするか、NULL モードで保持します。また、ローカル・インスタンスが後続の更新に対してブロックを要求すると、Oracle はディスクからバッファを読み込みます。これは、「強制読込み」と呼ばれます。

表にすでに索引が作成されている場合、UPDATE によって次のことが行われます。

1. SCUR モードで、索引のルート・ブロックが読み込まれます。
2. SCUR モードで、1 つ以上のブランチ・ブロックが読み込まれます。
3. SCUR モードで、リーフ・ブロックが読み込まれ、キャッシュに確保されます。
4. XCUR モードで、データ・ブロックが読み込まれます。
5. データ・ブロックが変更されます。

索引キー値が変更された場合、Oracle によって次のことが行われます。

1. SCUR モードで、ルートおよびブランチ・ブロックが再読込みされます。
2. XCUR モードで、リーフ・ブロックが読み込まれます。
3. 更新された行に対する索引キー値が変更されます。

索引を使用した更新操作中、ブロックは、更新中のインスタンスのキャッシュからいつでも ping でき、再取得される必要があります。ルートおよびブランチ・ブロック上の共有グローバル・ロックは、別のインスタンスがこれらのブロックのみを読み込んでいる間、問題ではありません。リーフ・ブロックが分割されたためにブランチ・ブロックを変更する必要がある場合、Oracle は S ロックを X ロックに拡張します。そのため、競合の可能性が高くなります。

そのため、パーティション化方法を設定するときに、頻繁に更新される表を明確に区別することが重要です。更新の頻度、アクセスのランダムさおよびデータの参照パターンによって、使用するレイアウトおよびロック方法が結果的に決定されます。ランダム・アクセスでは、1:1 解放可能ロックを使用するロック方法が適切です。あるトランザクションがブロックの同じ範囲に繰り返しアクセスする場合、表パーティション化およびトランザクション・ルーティング方法を使用できます。表内のデータ・パーティションを一度設定すると、固定ロックはほとんど必要ありません。

ランダム・アクセスが頻繁に発生する場合、表の作成またはロード時に PCTFREE を高い値に設定し、ブロックに少数の行が移入されるようにすることによって、同じデータまたは索引ブロックに対する競合の可能性を減少させることができます。ただし、トレードオフとして、ブロックの読込み、および同じ数の行にアクセスするための I/O が増加します。

> **参照：** 1:1 ロックと 1:N ロック、および解放可能ロックと固定ロック期間の詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

DELETE 中のブロック・アクセス

Oracle は、DELETE の場合も、UPDATE の場合と同じ方法でキャッシュ内のブロックにアクセスします。Oracle は、削除する行を含むブロックに対して表をスキャンします。そのため、表に索引がない場合、トランザクションはセグメント・ヘッダーを読み込み、ブロックを読み込んだ後、そのブロックを変更します。トランザクションはブロック内に空き領域を作成し、ブロック内のデータが PCTUSED 未満になったとき、ブロックが空きリストにリンクされるようにします。

したがって、トランザクションは、セグメント・ヘッダーまたは空きリスト・グループ・ブロックを排他モードで取得します。問題のあるブロックは、インスタンスの空きリスト・グループ（ある場合）に戻されます。一般に、大量の削除は、オフピーク時に 1 つのインスタンスから行うようにスケジューリングする必要があります。

SELECT 中のブロック・アクセス

SELECT では、バッファが SCUR または CR モードで読み込まれます。セグメント・ヘッダーに対する読込み、またはシステム内の唯一の読込み側が特定のブロックを読み込む場合などの SCR の場合は、グローバル S ロックが取得されます。ブロックが変更されるか、またはブロックにコミットされていない変更が含まれる場合に、一貫読込みバージョンが要求されると、そのバージョンを含むバッファは CR モードになります。別のノードで最新の変更が行われた場合、変更中のノードからの CR コピーが要求される必要があります。一度要求が完了すると、一貫読込みバージョンのブロックを受信したインスタンス上にはロックが保持されません。

フル・テーブル・スキャンの場合、SELECT では表に割り当てられたエクステントのエクステント境界を決定するために、セグメント・ヘッダーが読み込まれる必要があります。これには、ヘッダー・ブロックを含むバッファへの共有現行アクセスが必要なため、Parallel Server では、グローバル S ロックが、ヘッダー・ブロックを変更する INSERT と競合する場合があります。

グローバル・キャッシュ一貫作業およびブロック・クラス

一般に、データ・ブロック、索引ブロック、ロールバック・セグメント・ヘッダー、空きリスト・グループ・ブロック、ロールバック・セグメント・ブロックおよびセグメント・ヘッダーは、異なるクラスのブロックとみなされます。これらすべてのクラスは、共有データベースの構造を表すため、ping の対象になります。

Oracle Parallel Server では、インスタンスがロールバック・セグメントをプライベートに所有する必要があるため、ロールバック・セグメント・ヘッダーおよびロールバック・セグメント・ブロックに対しては、グローバル・キャッシュ同期化による影響があまり重要ではありません。これによって、ローカル・インスタンスに対するロールバック・セグメントの書込みが制限されます。一貫読込みに対してキャッシュ・フュージョンを使用すると、ロールバック・セグメント・キャッシュ一貫操作の影響は制限されます。最も一貫性のある読込み情報は、データ・ブロックのバージョンを作成し、それを要求しているインスタンスに直接送信するローカル・インスタンスによって生成されます。

空きリスト・グループがなく、索引または表に対する物理記憶パラメータが適切に最適化されていない場合、セグメント・ヘッダー・ブロックは頻繁に ping されます。空きリスト・グループを構成すると、これを削減できます。一般に、セグメント・ヘッダーの ping は、ping の合計の 5% を超える必要があります。

V$CLASS_PING ビューには、ブロック・クラスが異なる特定のロック変換の数がリストされます。このビューを使用して、合計 ping に対する各ブロック・クラスの ping の割合を監視します。

データベース・オブジェクト・パラメータに対する一般的な推奨事項

次に、データベース・オブジェクトに対するいくつかの一般的な推奨事項を示します。

- 通常の OLTP 操作中に拡大または縮小が予測される表または索引に対して、あるいは複数ノードから INSERT または DELETE が頻繁に行われる可能性があるオブジェクトに対しては、FREELISTS および FREELIST GROUPS を使用して、オブジェクトを作成します。
- GC_FILES_TO_LOCKS を定義する場合、エクステントをオブジェクトの FREELIST GROUP に事前に割り当てるか、またはブロッキング要素を使用します（あるいはその両方を行います）。これによって、特定のエクステントに属するブロックが 1 つのインスタンスのみに使用され、ブロックが同じロックによって処理されることが保証されます。
- ランダム・ローカルおよびリモート・アクセスのあるオブジェクトおよびファイルの場合は、正しいデータベース・ブロック・サイズを選択するか、データをロードするときに PCTFREE をより高い値（デフォルト値は 10%）に設定して、より多くの空き領域を使用可能にし、1 つのブロックに含まれる行の数を制限します。ただし、この場合は領域要件が増加します。

索引の問題

ほとんどの高ボリュームの OLTP システムでは、索引ブロックのインスタンス間競合によって、Oracle Parallel Server 処理のコストが増加します。また、一般に使用されている B*tree 索引構造では、様々なレベルで ping の影響を受けやすくなります。右上がりツリーでは、1 つの特定のリーフ・ブロックに頻繁な ping が発生します。ツリー構造を横断中にブランチ・ブロックを強制読込みする必要がある場合があります。これは、ブランチ・ブロックが最後に別のインスタンスによって変更されたためです。3 つのブロックを 1 つのトランザクションで変更する必要があるため、リーフ・ブロックの分割は影響を受けやすくなります。一般に、回避する必要がある状況を次に示します。

- リーフおよびブランチ・ブロックの競合

- ルート・ブロックの競合
- 索引セグメント・ヘッダーの競合

次の項では、リーフおよびブランチ・ブロックの競合の回避方法を示します。

リーフ / ブランチ・ブロックの競合の最小化

様々な方法を使用して、索引の異なる部分へアクセスを隔離または分散させ、パフォーマンスを向上できます。

右上がり索引ツリーを回避するには、逆キー索引を使用します。キーを逆にすることによって、索引のリーフ・ブロック上に索引キーをより広範囲に分散できます。そのため、複数インスタンスが同じリーフおよびブランチ・ブロックにアクセスする可能性が低下します。ただし、逆キー索引では索引の範囲スキャンができないため、逆キー索引の使用には十分注意してください。

順序番号に基づく索引の場合は、各インスタンスに異なる値を割り当てます。これによって、ある特性に基づいてパーティション化できるデータベース・オブジェクトについて、アクセス・パターンが適切に分散される場合があります。

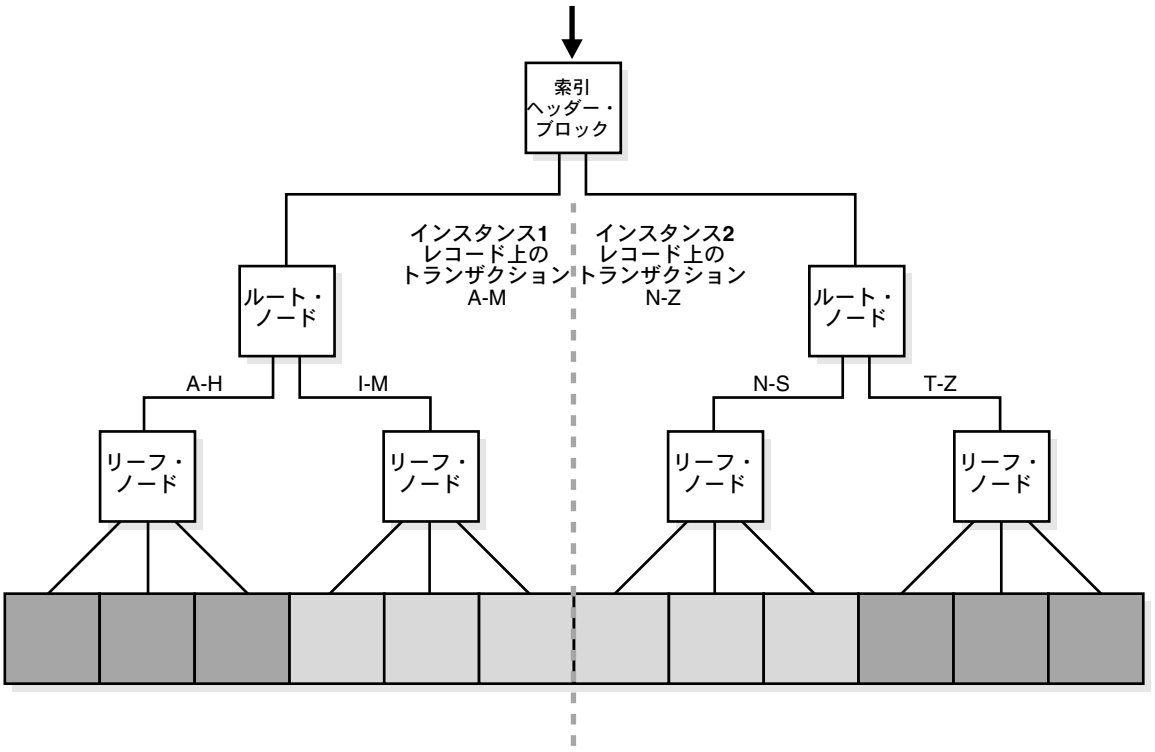
その後割り当てられたその他の値については、索引値を調整し、次のように INSTANCE_NUMBER を使用して、索引キーを生成します。

*index key = (instance_number -1) * 100000 + Sequence number*

注意 : ローカル・パーティション索引を使用するようにしてください。

図 6-2 に、レンジ・パーティション化された表に格納されたレコードに対するトランザクション操作によって、どのようにリーフおよびブランチ・ブロックの競合が最小化されるかを示します。

図 6-2 レンジ・パーティション化された表に対するトランザクションのノードの親和性



索引に対するロック方針

索引があるファイルに対する最適なロック方法を決定するのは困難な場合があります。通常、解放可能ロックを使用するのが最も実用的です。索引に固定ロックを使用すると、false ping の可能性が増加します。ロックの割当て方法を決定する場合、次のガイドラインが有効です。

- 逆キー索引を使用して（可能な場合）、右上がり索引ツリーを回避するか、または複数リーフ・ブロック上にアクセスを分散するテクニックを使用します。
- 論理パーティション化キーに基づく適切な索引キー値を使用するか、またはできるだけローカル・パーティション化索引を使用して、索引アクセスをパーティション化します。つまり、トランザクションをキー・データ値に基づいて専用ノードに送ります。
- まったくパーティション化できない場合は、解放可能ロックを使用します。
- ローカル索引パーティション化を使用できる場合は、パーティションを含むファイルに、より少ない 1:N 固定ロックを割り当てます。

> **参照：** 1:1 ロックと 1:N ロック、および解放可能ロックと固定ロック期間の詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

順序番号の使用

Oracle Parallel Server のアプリケーションを設計する場合は、できるだけ順序番号を使用します。順序の使用を最大化するには、各インスタンスのキャッシュに、順序を保持するための十分な大きさが必要です。デフォルトのキャッシュ・サイズでは、20 個の順序番号が保持されます。これを、200 個の順序番号を保持できるようにするには、次の構文を使用します。

```
ALTER SEQUENCE SEQUENCE_NAME CACHE 200;
```

順序番号キャッシュ・サイズの計算

次の 3 つの要素に基づいて、順序番号のキャッシュ・サイズを見積ります。

- トランザクション率
- クラスタ内のノード数
- クラスタの安定性

順序付けを使用すると、Oracle Parallel Server のキャッシュが抑制されます。通常、SHUTDOWN コマンドを実行すると、いくつかの番号が失われます。また、インスタンス障害の後に、順序番号が完全に失われることもあります。

外部シーケンス・ジェネレータ

次の場合に、外部シーケンス・ジェネレータを実装する必要があります。

- 順序付けが必須の場合
- すべての番号に対する責任が要求される場合
- トランザクション率が高い場合
- 多くのインスタンスがある場合
- 多くの順序番号がある場合

> **注意：** 順序番号を保持するためにデータベース表は使用しないしてください。

順序におけるグローバル競合の検出

順序のキャッシュが不十分か、またはまったくキャッシュされない場合、サービス回数の増加によって重大なパフォーマンスの問題が発生する場合があります。これによって、スケーラビリティが低下する場合もあります。

パフォーマンスの問題が発生した場合、後述のように V$SYSTEM_EVENT ビューの統計情報を調べて、問題が Oracle 順序の使用によるものかどうかを判断します。

- 順序による問題は、'row cache locks' に対する拡張平均待機時間として、数百 ms（ミリ秒）の範囲で、V$SYSTEM_EVENT に表示されます。非アイドル・イベントの合計待機時間に対する 'row cache locks' の待機時間の割合が、非常に高くなります。
- DC_SEQUENCES パラメータの場合、DLM_CONFLICTS 対 DLM_REQUESTS の割合は、非常に高くなります。この割合が 10 ～ 15% を超え、'row cache locks' 待機時間が合計待機時間の割合の大部分を占める場合、順序のキャッシュが不十分なために、サービス時間が低下することがあります。

論理および物理データベース・レイアウト

次の考慮点に基づいて、物理パーティション化方法を決定します。

- オブジェクトに対するトランザクション・プロファイル
  - アクセス頻度
  - アクセス方法
- オブジェクトに対するファンクション依存性（ある場合）
  - ビジネス・ファンクション
  - データ

これらのオブジェクトをファイルおよび表領域にグループ化し、一貫したロック方法を適用できるようにする必要があります。

物理レイアウトに対する一般的な提案

オブジェクトを表領域にグループ化する前に、後で表領域に割り当てることができるロー・ボリュームのプールを作成します。Oracle Parallel Server は、共有ロー・ボリュームまたは共有ロー・パーティションを使用して、物理的にデータを格納する必要があるため、使用可能なディスクを様々なサイズのボリュームにパーティション化する必要があります。まず、次の推奨事項を考慮してください。

1. 後で論理パーティションに割り当てる共有ボリュームのラージ・プールを作成します。
2. ロー・ボリュームの標準サイズを 10MB、100MB、500MB、1GB などに定義します。
3. 初期 I/O ボリュームの見積もりで指定されている数のディスクに、ボリュームをスライス化またはストライプ化します。グローバル・キャッシュ同期化 I/O の見積もりに、10 ～ 15% の読込みおよび書込み I/O を追加する必要があることを考慮します。
4. 計画を確認します。ping によって、いくつかのファイルの I/O ボリュームが他のファイルより高くなるため、レイアウトの再構築が必要な場合があります。

目標は、柔軟な物理レイアウトを作成することです。また、キャッシュ一貫性を保つために必要な I/O を考慮し、これによって、I/O 待ち時間が悪影響を受けないようにする必要があります。

表領域の設計

表領域の設定における目標は、そのデータ・アクセス分散によってデータベース・オブジェクトをグループ化することです。データベース・オブジェクトに対する依存性の分析およびトランザクション・プロファイルを考慮すると、表領域を次のオブジェクトに対するコンテナに分割できます。

- 特定のトランザクションまたはデータベースの一部分への親和性を持つ可能性が低い、頻繁かつランダムにアクセスされる表および索引
- 主に読込みまたは読込み専用で、ほとんど変更されない表および索引
- データの大部分を分割および自律型データ・セットに細分化して参照できる表および索引

データベース・オブジェクトを表領域に分割する場合、さらに次の基準を考慮します。

- 表は、索引から分離する
- 実際に読込み専用である表は、読込み専用表領域に割り当てる
- 各ノードには、一時表領域が必要
- 小さい参照表は、同じ表領域にグループ化する
- 挿入専用表は、他の表から分離する

次の章に示すように、データベース・オブジェクトを論理的にパーティションにグループ化し、それらを物理的に表領域に割り当てると、効率的で適応性のあるロック方法を開発できます。

グローバル・キャッシュ・ロック割当て

表領域を設計すると、それらに予測されるアクセス分散、アクセス頻度の見積りおよびトランザクションやノードへの親和性に基づいて、いくつかの個別のグループ化が必要になります。次のようなグループ化があります。

- トランザクションまたはノードへの親和性がほとんどまたはまったくなく、false ping の可能性が高い、頻繁かつランダムにアクセスされるファイル
- 読込み専用ファイルまたは読込みが主なファイル
- 複数ノードで共有されるが、データの大部分が自律型データ・セットに細分化されている、頻繁にアクセスされるファイル
- ロールバック・セグメントおよび一時セグメントを含むファイル

これらの細分化に基づいて、次のようにロックを割り当てます。

- 解放可能ロックは、複数ノードによって頻繁かつランダムに変更されるファイルに割り当てます。このロック方針は、多くの場合、索引ファイルに対して最も適しています。
- 読込み専用表領域には、ロックを割り当てないでください。
- 固定ロックはアクセス競合の可能性が低く、ほとんどの場合、分割された形でアクセスされるデータ部分を持つ、読込みが主なデータにのみ割り当てます。
- 一時表領域に属するロールバック・セグメントおよびファイルには、ロックを割り当てないでください。かわりに、GC_ROLLBACK_LOCKS を設定してロールバック・セグメント・ロックを割り当て、わずかな固定ロックのみが必要になるようにします。

ロックを割り当てるとき、GC_RELEASABLE_LOCKS によって十分なリソースが提供されている場合は、常に、ロックを解放可能にして割り当てます。これによって、ロックのオープンおよびクローズのコストが最小化されます。

結論およびガイドライン

最終的には、応答時間およびスループット要件によって、パーティション化方法をどれだけ強力で独立性があるものにする必要があるか、および最適な設計を実現するためにどれだけの工数が必要かが決定されます。キャッシュ・フュージョンによって、他のノードにデータを要求する一貫読込みトランザクションに対応付けられたオーバーヘッドの量が、大幅に削減されます。これによって、より単純なデータベース設計を実装しながら、最適なパフォーマンスを実現できます。

トランザクション・プロファイルおよびファンクション依存性の点から、データ・アクセス分散を慎重に分析することは、特定のインスタンスに対する割当て作業の基礎になります。さらに、これによって、システムがより強力でスケーラブルになります。

一般に、80% 以上のオーバーヘッドが、20% 以下の特定の作業負荷によって発生します。いくつかの簡単なガイドラインおよび注意に従ってこの 20% を処理すると、最小限の工数で効果を得ることができます。

- 索引ブロック競合は、必ず回避する必要があります。ただし、索引キー値が複数インスタンスによって変更されない場合、そのオーバーヘッドは受け入れられます。
- リモート・アクセスおよびその結果の false ping の可能性が低い場合、グローバル共有データ・アクセスは受け入れられます。
- グローバル共有データを変更する長時間実行トランザクションは、回避する必要があります。また、システムの使用がより少ない時間に、実行する必要があります。
- データが一定に保たれる場合は、読込み専用表領域を使用する必要があります。

- パーティション化データ、および頻繁に変更される非パーティション化データには、空きリスト・グループを定義する必要があります。

一般に、アプリケーション・パーティション化用データベースを設計し、データのストライプ化が可能な表領域を作成する必要があります。これによって、パラレル・データ・ロードおよびインスタンス間パラレル問合せの処理が簡単になります。

7

PCM および非 PCM インスタンス・ロックの使用計画

この章では、Oracle Parallel Server 環境のデータ・ファイルに対し、パラレル・キャッシュ管理（PCM）ロックおよび非 PCM ロックを割り当てるために設定する、初期化パラメータについて説明します。内容は次のとおりです。

- PCM ロックの使用およびメンテナンスの計画
- Oracle によるブロックに対するロックの割当て方法
- ブロックへの PCM ロックのマッピング例
- 非 PCM インスタンス・ロック

注意： PCM ロックをチューニングしても、最適なアプリケーション・スケーラビリティが得られない場合があります。詳細は、第 5 章および付録 A を参照してください。

参照：

- PCM ロックおよび GC_* パラメータの概要は、『Oracle8*i* Parallel Server 概要』を参照してください。
- Oracle Parallel Server にロックを割り当てるために使用する初期化パラメータについては、『Oracle8*i* リファレンス・マニュアル』を参照してください。

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。内容は次のとおりです。

- インスタンス・ロックの計画およびメンテナンス
- PCM ロック割当て
- データ・ファイルおよびデータ・ブロックの検査

インスタンス・ロックの計画およびメンテナンス

分散ロック・マネージャ（DLM）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。このため、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の量を調べる必要もあります。ロックの計画の際には、次の点を考慮してください。

- DLM 機能で構成された数より多いロックを使用しようとした場合、Oracle はエラー・メッセージを表示し、インスタンスを停止します。
- GC_* または LM_* 初期化パラメータを変更して多くのロックを指定すると、メモリー使用量およびシステム・グローバル領域で使用可能なメモリーの量に影響します。
- インスタンスの数は、システムに必要なメモリー量およびロックの数に影響します。

PCM ロック割当て

PCM ロックを割り当てるときに重要なことは、INSERT、UPDATE および DELETE 文を使用してデータを変更する頻度を分析することです。この分析を行い、読込み専用にするか読込み / 書込みにするかに従い、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を決定します。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ます。
- 読込み / 書込み集中ファイルには、多数のロックを割り当てます。
- 表領域全体が読込み専用の場合、単一ロックを割り当てることができます。表領域にロックを割り当てない場合、システムは予備ロックの使用を試みます。これは、予備ロックを取得するため、他の表領域との競合の原因になることがあります。その結果、不要な強制読込み / 書込みが行われることがあります。

オブジェクト（索引または表）のタイプ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決まります。

参照： 第 5 章および第 6 章を参照してください。

データ・ファイルおよびデータ・ブロックの検査

ロックは、次の値を指定することによって、様々なレベルで割り当てる必要があります。

- 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
- 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
- 任意のファイル内の特定のデータ・ブロックのクラスを処理する特定のロック

最初にデータ・ファイルおよびそれに含まれるブロックを調査します。

ファイル ID、表領域名およびブロックの数の判断

すべてのデータベースについてファイル ID、ファイル名、表領域名およびブロックの数を判断するには、次の文を使用します。

```
SELECT FILE_NAME, FILE_ID, TABLESPACE_NAME, BLOCKS
FROM DBA_DATA_FILES;
```

Oracle は、次の例のような結果を表示します。

```
FILE_NAME               FILE_ID     TABLESPACE_NAME     BLOCKS
-----------------------------------------------------------------
/v7/data/data01.dbf     1           SYSTEM               200
/v7/data/data02.dbf     2           ROLLBACK            1600
. . .
```

必要なロックの数の判断

次の方法で、特定の使用方法に必要なロックの数を見積もります。

- データおよびアプリケーションの性質を考慮します。

  使用頻度が高く、同時更新されるデータ・ファイルには、多くのロックが必要です。ただし、問合せ専用アプリケーションには、多くのロックは必要ありません。データ・ファイルに単一のロックで十分です。

- 多くのインスタンスが同時に変更を行うファイルには、多くのロックを割り当てます。

これによって、ロックの競合が減少し、I/O アクティビティが最小になり、ファイル内のデータのアクセス性が高くなります。

- 複数のインスタンスが同時にアクセスする必要がないファイルには、割り当てるロックの数を少なくします。

  これによって、ロック管理の不要なオーバーヘッドを回避できます。

- ファイル内のオブジェクトを調査し、そのオブジェクトに対して行われる操作を検討します。
- 読込み専用表領域内の読込み専用オブジェクトをグループ化します。

Oracle によるブロックに対するロックの割当て方法

この項では、ブロックへの固定ロックおよび解放可能ロックの割当て方法を説明します（1:1 ロックは、ブロックに 1 対 1 で対応します）。

ファイルへのロックのマッピング

ファイルへのロックのマッピングは、システム・グローバル領域における 2 つのデータ構造によって制御されます。1 番目の構造が、各ファイル（DB_FILES）を 2 番目の構造のバケット（索引）にマップします。この構造には、このバケットに割り当てられたロックの数、ベース・ロック番号およびグループ係数についての情報が含まれます。表領域に対するロックの数を調べるには、異なるファイルを保護する実際の固定ロックの数をカウントします。そのファイルがロックを共有する場合、共有ロックは 1 回のみカウントします。

1. 表領域に対するロックの数を調べるには、まず、FILE_LOCK データ・ディクショナリ表から選択を実行します。

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

たとえば、GC_FILES_TO_LOCKS="1=500:5=200" を設定した場合、Oracle は次のように応答します。

```
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. START_LCK 列の値が異なる行のロックの数（NLOCKS 列の値）のみを合計して、表領域のロックの数を計算します。

この例では、ファイル 1 およびファイル 5 は、START_LCK に異なる値を持ちます。したがって、NLOCKS の値を合計すると 700 ロックになります。

ただし、GC_FILES_TO_LOCKS="1-2=500:5=200" を設定した場合、結果は次のようになります。

```
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
```

ここでは、ファイル 1 およびファイル 2 は、それぞれの START_LCK の値が同じであるため、ロックを共有していることがわかります。ファイル 5 は、START_LCK に異なる値を持ちます。したがって、ファイル 1 と 2 で共有している 500 ロックを 1 回カウントし、それにファイル 5 の 200 ロックを加えて、合計は 700 になります。

ブロック・クラスごとのロックの数

データのブロックの数および UNDO ブロック・クラスのみを考慮する必要があります。データ・ブロック（クラス 1）には、索引または表のデータが含まれます。UNDO 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表とも呼ばれます。システム UNDO ブロック（クラス 11）は、ロールバック・セグメントの一部分であり、UNDO レコードに対して記憶域を提供します。

ユーザー UNDO セグメント *n* のヘッダー・ブロックは、クラス 10 + (*n* * 2) として識別されます。この場合、*n* はロールバック・セグメント番号を表します。*n* = 0 の場合は、システム・ロールバック・セグメントであり、*n* > 0 の場合は、非システム・ロールバック・セグメントです。同様に、ユーザー UNDO セグメント *n* のヘッダー・ブロックは、10 + ((n * 2) + 1) として識別されます。

次の問合せによって、クラスごとに割り当てられるロックの数が表示されます。

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

次の問合せによって、固定（リリース不可能）PCM ロックの数が表示されます。

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

次の問合せによって、解放可能 PCM ロックの数が表示されます。

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

ロック要素番号

データ・クラス・ブロックに対するファイル番号は、データ・ブロック・アドレス（DBA）から判断されます。バケットは、X$KCLFI 動的パフォーマンス表から検索されます。データ・クラス・ブロックは、次のようにロック要素番号に固定されます。

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

他のブロック・クラスは、次のようにロック要素番号に固定されます。

(*DBA*) modulo (クラスのロック数)

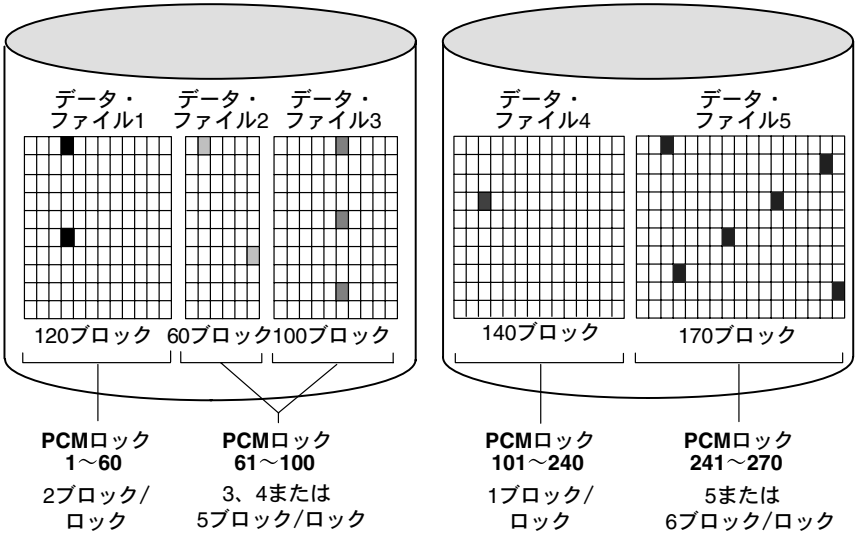
ブロックへの PCM ロックのマッピング例

GC_FILES_TO_LOCKS の設定

次の例では、異なる方法を使用したブロックへの PCM ロックのマッピング、および複数のデータ・ファイルでの同じロックの使用方法を示します。

注意： これらの例では、重要な概念を説明するために非常に小さなサンプル・ファイルを使用しています。実際には、もっと大きいファイルを管理します。

図 7-1 データ・ブロックへの PCM ロックのマッピング



例 1 図 7-1 に、パラメータ値 GC_FILES_TO_LOCKS = "1=60:2-3=40:4=140:5=30" に対する PCM ロックへのブロックのマッピング例を示します。

図 7-1 に示すデータ・ファイル 1 では、60 の PCM ロックが、60 の倍数である 120 ブロックにマップされます。

データ・ファイル 2 および 3 では、40 の PCM ロックが合計 160 ブロックにマップされます。1 つの PCM ロックは、データ・ファイル 2 では 1 つまたは 2 つのデータ・ブロックを、データ・ファイル 3 では 2 つまたは 3 つのデータ・ブロックを処理できます。したがって、1 つの PCM ロックが、この 2 つのデータ・ファイル全体の、3 つ、4 つまたは 5 つのデータ・ブロックを処理できます。

データ・ファイル 4 では、PCM ロックの数とデータ・ブロックの数が同じであるため、各 PCM ロックは単一のデータ・ブロックにマップされます。

データ・ファイル 5 では、30 の PCM ロックが、30 の倍数でない 170 ブロックにマップされます。したがって、各 PCM ロックは 5 つまたは 6 つのデータ・ブロックを処理します。

図 7-1 で示した PCM ロックはそれぞれ、読込みロック・モードまたは読込み排他モードのどちらかに保持されます。

例 2 次のパラメータ設定では、データ・ファイル 1 に 500 の PCM ロックを、データ・ファイル 2、3、4、10、11 および 12 にそれぞれ 400 の PCM ロックを、データ・ファイル 5 に 150 の PCM ロックを、データ・ファイル 6 に 250 の PCM ロックを、ファイル 7 ～ 9 にまとめて 300 の PCM ロックを割り当てます。

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

この例では、(500 + (6*400) + 150 + 250 + 300) = 3600 で合計 3600 の PCM ロックを割り当てます。さらに多くのデータ・ファイルを追加する場合は、これより多くの PCM ロックを指定できます。

例 3 例 2 では、句 "7-9=300" によって 300 の PCM ロックがまとめてデータ・ファイル 7、8 および 9 に割り当てられます。キーワード EACH は省略されています。これらのデータ・ファイルにそれぞれ 900 のデータ・ブロックがある場合、データ・ブロックの合計は 2700 となるため、各 PCM ロックは 9 つのデータ・ブロックを処理します。データ・ファイルが 300 の倍数であるため、PCM ロックで処理される 9 つのデータ・ブロックは、3 つのデータ・ファイルに分散します。つまり、1 つの PCM ロックは各データ・ファイルで 3 つのデータ・ブロックを処理します。

例 4 次のパラメータ値は、ファイル 1 ～ 3 にそれぞれ 200 の PCM ロックを、データ・ファイル 4 に 50 の PCM ロックを、データ・ファイル 5、6、7 および 9 にまとめて 100 の PCM ロックを、データ・ファイル 8 と 10 の組合せに、連続する 50 のブロック・グループの 20 の PCM ロックを割り当てます。

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

この例では、結合したデータ・ファイル 5、6、7 および 9 に割り当てられた PCM ロックは、各データ・ファイルにおいて 1 つ以上のデータ・ブロックを処理します。ただし、1 つのデータ・ファイルが 100 より少ないデータ・ブロックを含む場合を除きます。データ・ファイル 5 ～ 7 がそれぞれ 500 のデータ・ブロックを含み、データ・ファイル 9 が 100 のデータ・ブロックを含む場合、各 PCM ロックは 16 のデータ・ブロックを処理します。この場合、データ・ファイル 9 で 1 つ、その他のデータ・ファイルでそれぞれ 5 つのデータ・ブロックが処理されます。または、データ・ファイル 9 が 50 のデータ・ブロックを含む場合は、PCM ロックの半数が 16 のデータ・ブロック（そのうちの 1 つはデータ・ファイル 9）を処理し、残りの半分の PCM ロックは 15 のデータ・ブロックのみ（データ・ファイル 9 にはなし）を処理します。

データ・ファイル 8 および 10 にまとめて割り当てられた 20 の PCM ロックは、50 のデータ・ブロックの連続するグループを処理します。データ・ファイルが 50 の倍数のデータ・ブロックを含み、かつデータ・ブロックの合計数が 20 * 50（つまり 1000）を超えない場合、各 PCM ロックは、データ・ファイル 8 またはデータ・ファイル 10 のいずれかのデータ・ブロックを処理します。これは、PCM ロックがそれぞれ 50 の連続するデータ・ブロックを処理するためです。データ・ファイル 8 のサイズが、50 のデータ・ブロックの倍数でない場合、1 つの PCM ロックは、両方のファイルのデータ・ブロックを処理する必要があります。データ・ファイル 8 および 10 のサイズが、1000 のデータ・ブロックを超える場合、いくつ

かの PCM ロックは、50 のデータ・ブロックのグループを 2 つ以上処理する必要があります。このグループは、1 つのデータ・ファイルにあるとは限りません。

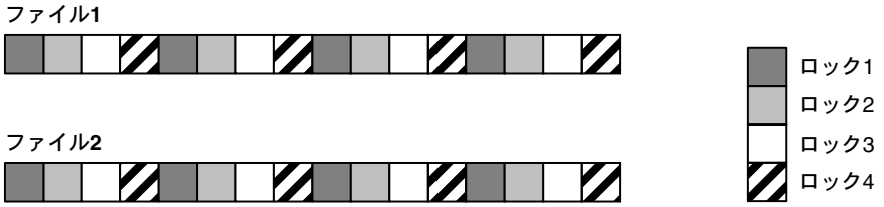
GC_FILES_TO_LOCKS の固定ロックの設定例

例 5、6 および 7 に、様々な値の GC_FILES_TO_LOCKS を指定した結果を示します。これらの例では、ファイル 1 および 2 にはそれぞれ、データが 16 ブロックあります。

例 5 GC_FILES_TO_LOCKS="1-2=4"

この例では、ファイル 1 および 2 に対して 4 つのロックが指定されています。したがって、各ロックが処理するブロックの数は 8（(16+16)/4）です。これらのブロックは連続していません。

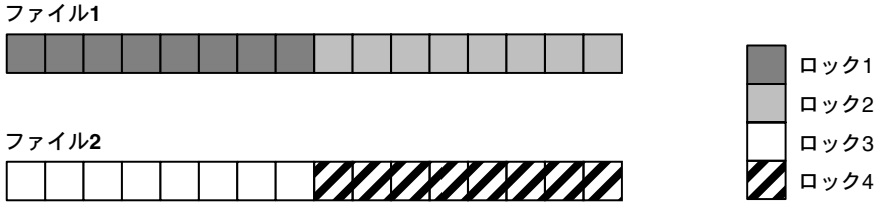
図 7-2 GC_FILES_TO_LOCKS の例 5



例 6 GC_FILES_TO_LOCKS="1-2=4!8"

この例では、ファイル 1 および 2 に対して 4 つのロックが指定されています。ロックは、8 つの連続するブロックを処理する必要があります。

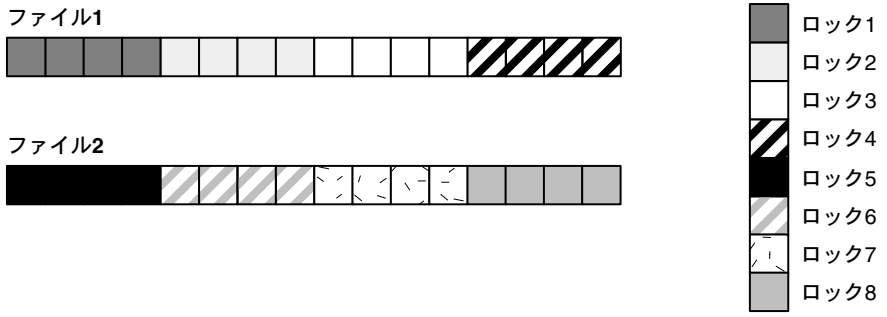
図 7-3 GC_FILES_TO_LOCKS の例 6



例 7 GC_FILES_TO_LOCKS="1-2=4!4EACH"

この例では、ファイル 1 に対して 4 つのロック、およびファイル 2 に対しても 4 つのロックが指定されています。ロックは、4 つの連続するブロックを処理する必要があります。

図 7-4 GC_FILES_TO_LOCKS の例 7



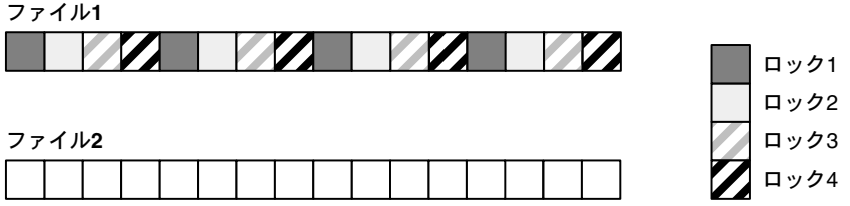
GC_FILES_TO_LOCKS の解放可能な設定例

次の例では、固定ロックと組み合せた解放可能ロックを示します。

例 8 GC_FILES_TO_LOCKS="1=4:2=0"

ファイル 1 には、4 つの固定 PCM ロックがあります。ファイル 2 には、解放可能ロックが要求に応じて割り当てられます。初めから割り当てられているロックはありません。

図 7-5 GC_FILES_TO_LOCKS 例 8



PCM ロックの必要性を分析するワークシートの使用

大きなアプリケーションでは、関連するビジネス・プロセスを慎重に検討します。この項で示すワークシートを使用すると便利です。

システムで日常的に実行する操作の種類を判断します。X ロックを必要とする操作と、S ロックを必要とする操作を区別することが重要です。Oracle が、1 つのロックをあるモードから他のモードに変換するたびに、ロックが必要になります。同じ表における、異なるインスタンス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、表内の行の数、および表の増加率も考慮する必要があります。この分析に基づいて、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 7-6 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% フル・テーブル・スキャン? 単一行?	
B				100%	
C					
D					

図 7-7 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 7-8 PCM ロック・ワークシート 3

表の名前	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

通常、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、Oracle Parallel Server 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は、単一の PCM ロックで処理できます。読込み専用でないデータは、複数の PCM ロックで処理する必要があります。

データが読込み専用で、一度インスタンスが読込み専用の表領域の PCM ロックを所有すると、それを解放することはありません。最初のロックを取得した後は、DLM 操作は不要です。

読込み専用の表領域をパーティション化して、専用の PCM ロックの集合で処理すると、最良の結果を得ることができます。これを行うには、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配置して、その後で、GC_FILES_TO_LOCKS パラメータを使用して、表領域のデータ・ファイルに PCM ロックを割り当てます。

注意： 読込み専用データおよび書込み可能データを、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 ロックのパーティション化

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、インスタンス間で PCM ロックをパーティション化することもできます。

この方法によって、不要な分散ロック管理が最小限で済みます。また、要求されたデータ・ブロックは、他のインスタンスが所有する PCM ロックによって処理されるため、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発生するディスク I/O も最小化されます。

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、あるまとまりの PCM ロックをファイル 1、2 および 3 に割り当て、別のまとまりの PCM ロックをファイル 4 および 5 に割り当てることができます。これによって、各インスタンスは、更新するデータに対する PCM ロック・オーナーシップを取得します。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のみです。

これに対して、データ・ファイル 3 および 4 にあるまとまりの PCM ロックを割り当てた場合は、I/O が増加します。これは、両方のインスタンスが同じまとまりの PCM ロックを定期的に使用するためです。

非 PCM インスタンス・ロック

この項では、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。内容は次のとおりです。

- 非 PCM インスタンス・ロックの概要
- 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（SCN）
- ライブラリ・キャッシュ・ロック（L[A-Z]）、（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）

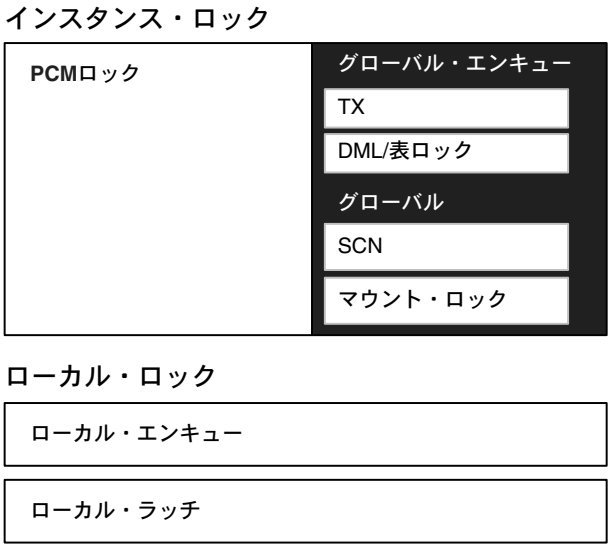
> **参照：** DLM で構成するための非 PCM リソースおよびロックの数の計算方法の詳細は、第 10 章を参照してください。

非 PCM インスタンス・ロックの概要

この項では、Oracle 環境において、Oracle がどのように非 PCM ロックを使用して、トランザクション、表およびその他のエンティティに対するロックを管理するかについて説明します。トランザクション・ロックの TX や表ロックの TM など、各タイプのロックの接頭辞は、Oracle がそれを識別するために使用するネーミング・スキーマに従っています。

図 7-9 に、Oracle で使用されるその他のロックに関連する非 PCM ロックを示します。

図 7-9 Oracle のロック・メカニズム：非 PCM ロック



PCM ロックが静的（アプリケーション設計時にユーザーが割り当てる）であることは対照的に、非 PCM ロックは非常に動的です。システムの初期化パラメータ値が変更されると、非 PCM ロックの数および対応する領域要件も変更されます。

トランザクション・ロック（TX）

行ロックは、選択された行を保護するロックです。トランザクションは、次の文のうちの 1 つによって変更された各行に対して、グローバルなエンキューおよび排他ロックを取得します。

- INSERT
- UPDATE
- DELETE
- FOR UPDATE 句を持つ SELECT

これらのロックはブロックに格納され、各ロックはグローバルなトランザクション・エンキューを参照します。

トランザクション・ロックは、トランザクションが最初の変更を開始したときに、排他モードで取得されます。これは、トランザクションが COMMIT または ROLLBACK を実行するまで保持されます。SMON も、トランザクションをリカバリ（UNDO）するときに、トランザクション・ロックを排他モードで取得します。トランザクション・ロックは、進行中のトランザクションにロックされたオブジェクトが解放されるのを待っているプロセスのための、キューイング・メカニズムとして使用されます。

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 句を持つ SELECT および LOCK TABLE のうちの 1 つによって表が変更されたとき、表ロックを取得します。表ロックは、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）のいずれかのモードで保持できます。

インスタンスがデータベースをマウントするときに、使用するすべてのインスタンスが DML_LOCKS = 0 または DML_LOCKS != 0 のいずれかに設定されていることを確認するために、表ロックが使用されます。このように設定されていない場合、Oracle はエラー・メッセージ ORA-00061 を表示し、マウントの試行は失敗します。表ロックは、トランザクションの実行中に DML 文を使用して参照するときに取得されます。これによってオブジェクトは、トランザクションの実行中に削除または変更されません。これは、DML_LOCKS パラメータが 0（ゼロ）でない場合にのみ行われます。

また、次の文を使用して、特定の表に対して選択的に表ロックをオンまたはオフにできます。

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

DML_LOCKS が 0（ゼロ）に設定された場合、DDL 操作はできません。これは、表ロックが使用禁止に設定された表に対しても同様です。

> **参照：** パフォーマンス向上のためのインスタンス・ロックの最小化および表ロックの使用禁止の詳細は、10-3 ページの「パフォーマンスを最適化する表ロックの最小化」を参照してください。

システム変更番号（SCN）

システム変更番号（SCN）は、単一インスタンス内およびすべてのインスタンスに渡って、イベントを順序付けるために Oracle が使用する、論理的なタイムスタンプです。SCN を生成するために Oracle が使用するスキームの 1 つは、ロック・スキームです。

ロック SCN スキームによって、グローバル SCN が SCN ロックの値ブロックに保持されます。Oracle は、多くのデータベース・イベントに応答して、特に COMMIT の後にこの値を増やします。グローバル SCN を増やすプロセスによって、排他モードで SCN ロックが取得され、SCN が増やされ、ロック値ブロックが作成され、ロックがダウングレードされます。SCN ロック値へのアクセスは、バッチ処理で行われます。Oracle は、グローバル SCN のキャッシュ・コピーをメモリーに保持します。プロセスは、他のプロセスがフェッチした SCN を読み込むことによって、通信オーバーヘッドを発生させずに SCN を取得できます。

SCN の実装は、プラットフォームによって異なる場合があります。ほとんどのプラットフォームでは、MAX_COMMIT_PROPAGATION_DELAY 初期化パラメータがプラットフォーム固有のしきい値（通常は 7）より小さい場合、Oracle がロック SCN スキームを使用します。

Oracle は、MAX_COMMIT_PROPAGATION_DELAY がしきい値より大きい場合、Lamport SCN スキームを使用します。インスタンスが起動された後でアラート・ログを調べ、どの SCN 生成スキームが選択されたかを参照できます。

> **参照：** SCN 実装については、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。

ライブラリ・キャッシュ・ロック（L[A-Z]）、（N[A-Z]）

データベース・オブジェクト（表、ビュー、プロシージャ、ファンクション、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、SQL（DML/DDL）または PL/SQL 文の解析中またはコンパイル中に参照された場合、その文を解析またはコンパイルするプロセスは、適切なモードでライブラリ・キャッシュ・ロックを取得します。Oracle8 では、解析またはコンパイルが完了するまでの間（解析コールの所用時間）のみ、ロックが保持されます。

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、メタデータ・ストアであるデータ・ディクショナリの情報が含まれます。このキャッシュによって、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、そのメタデータをデータ・ディクショナリ・キャッシュから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には、排他モードおよび単一共有モードでラッチを使用します。インスタンス・ロックは、複数共有（パラレル）モードで使用します。

Oracle Parallel Server では、すべてのノードのデータ・ディクショナリ・キャッシュに、1 つのインスタンスで削除される表のメタデータを含めることができます。この表のメタデータは、すべての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される必要があります。これは、インスタンス・ロックによって実行および同期化されます。

データベース・マウント・ロック（DM）

マウント・ロックによって、インスタンスが特定のデータベースをマウントしたかどうかが表示されます。このロックは、Oracle Parallel Server でのみ使用されます。これは、排他モードの Oracle Parallel Server によって使用される唯一の複数インスタンス・ロックであり、他のインスタンスが共有モードでデータベースにマウントするのを防止します。

Oracle Parallel Server の単一共有モードでは、DM ロックは共有モードで保持されます。したがって、別のインスタンスは、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、Oracle Parallel Server の排他モードでは、別のインスタンスはロックを取得できません。

8

空きリスト・グループによるデータのパーティション化

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、データをパーティション化する方法について説明します。空きリストは、単一インスタンスの Oracle に存在します。ただし、空きリスト・グループは、Oracle Parallel Server 環境にのみ存在します。Oracle Parallel Server の空きリスト・グループは、データをパーティション化し、空き領域に対する競合を最小化するために使用します。

空きリスト・グループによるデータのパーティション化は、アプリケーション・プロファイルが表のパーティション化を許可しない場合にのみ使用します。空きリスト・グループと同じ役割を実現するためには、パーティション表およびパーティション索引を使用する方が簡単です。

> **参照：** パーティション表およびパーティション索引の使用方法は、第 5 章を参照してください。

この章の内容は、次のとおりです。

- 空きリストの実装手順の概要
- データベース・オブジェクト用の空き領域のパーティション化方法の決定
- CREATE 文の FREELISTS パラメータおよび FREELIST GROUPS パラメータの使用
- インスタンス、ユーザーおよびロックと空きリスト・グループとの対応付け
- エクステントの事前割当て
- 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

> **参照：** 空きリスト・グループの概要については、『Oracle8*i* Parallel Server 概要』を参照してください。

空きリストの実装手順の概要

空きリストを使用して複数インスタンスの空き領域を管理するには、次の手順に従います。

1. データベース・オブジェクトを分析し、空き領域およびデータのパーティション化方法を決定します。
2. 個々の表、クラスタおよび索引について、CREATE 文の FREELISTS 句および FREELIST GROUPS 句を設定します。
3. インスタンス、ユーザーおよびロックを空きリストに対応付けます。
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、Oracle Parallel Server に最適とはいえないアプリケーションの場合でも、パフォーマンスを向上できます。

データベース・オブジェクト用の空き領域のパーティション化方法の決定

この項では、データベース・オブジェクトを分析し、最適なパフォーマンスを得るための空き領域およびデータのパーティション化方法を決定するために有効なワークシートを示します。

データベース・オブジェクトの特性

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。

オブジェクト読込み専用表

表への挿入アクティビティが高くなく、新しい領域の割当てを必要としない程度の更新であれば、空きリストまたは空きリスト・グループは必要ありません。

パーティション・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切にパーティション化した場合は、表またはセグメントに挿入する必要があるノードは 1 つのみです。このような場合、ユーザーが多ければ空きリストが必要になります。ユーザーがほとんどいない場合、空きリスト・グループは必要ありません。

パーティション・データに関連するオブジェクト

パーティション・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。

ランダム挿入がある表のオブジェクト

同じ表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、書き込む場所を判断する必要があります。したがって、マスター空きリストを含むセグメント・ヘッダーに対する競合が発生します。

空き領域ワークシート

表 8-1 で示すように、表、クラスタ、索引などのデータベース・オブジェクトをそれぞれワークシートでリスト化し、各オブジェクトの空きリストおよび空きリスト・グループを計画します。

表 8-1 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	NA	NA
	NA	NA
	NA	NA
	NA	NA
パーティション・アプリケーションのオブジェクト	NA	
	NA	
	NA	
	NA	
パーティション・データに関連するオブジェクト	NA	NA
	NA	NA
	NA	NA
	NA	NA
ランダム挿入がある表のオブジェクト		

注意： パーティション・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 パラメータおよび FREELIST GROUPS パラメータの使用

この項の内容は次のとおりです。

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 記憶域パラメータを指定します。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。

インスタンスに大量の DML が発生する場合、一般的には Oracle Parallel Server のインスタンスに対して空きリスト・グループを作成します。その後、FREELIST GROUPS の値を、クラスタ内のインスタンスの数と等価に設定します。

注意： 記憶域パラメータを設定した後で、ALTER TABLE 文、CLUSTER 文または INDEX 文でこれらの値を変更することはできません。

FREELISTS パラメータ

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 のデフォルトおよび最小値は 1 です。最大値は、データ・ブロックのサイズによって異なります。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 つ当りに予想される同時挿入の数によって異なります。

FREELIST GROUPS パラメータ

各空きリスト・グループは、起動時に 1 つ以上のインスタンスに対応付けられます。FREELIST GROUPS のデフォルト値は 1 です。これは、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、Oracle Parallel Server 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、データのパーティション化が発生します。 1 つのインスタンスに割り当てられたブ

ロックや、別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： 複数の空きリスト・グループを使用すると、空きリスト構造はセグメント・ヘッダーから切り離されるので、セグメント・ヘッダーに対する競合が減少します。これは、高ボリュームの UPDATE および INSERT トランザクションがある場合に、非常に有効です。

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 );
```

クラスタ化表用の空きリストの作成

クラスタ化表の場合は、CREATE TABLE 文に FREELISTS および FREELIST GROUPS 記憶域パラメータを指定できません。空きリスト・パラメータは、個々の表についてではなく、クラスタ全体について指定する必要があります。これは、クラスタ化表では、CREATE CLUSTER 文の記憶域パラメータが使用されるためです。

クラスタは、共通のキー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行は、物理的に一緒にデータベース内に格納されます。Oracle Parallel Server 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パーティション化されている場合に、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。

注意： TRUNCATE TABLE *table_name* REUSE STORAGE 構文を使用すると、空きリストに対するエクステント・マッピングが削除され、最高水位標が最初のエクステントの開始点にリセットされます。

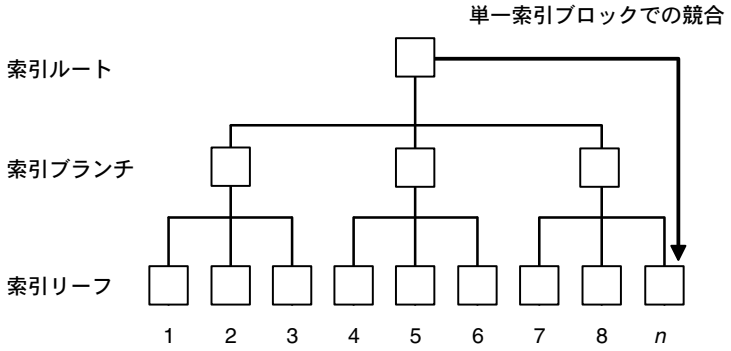
参照： TRUNCATE TABLE 文の REUSE STORAGE 句の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

索引用の空きリストの作成

CREATE INDEX 文の FREELISTS および FREELIST GROUPS 記憶域パラメータを使用して、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できます。これらのパラメータの使用方法は、表の場合と同じです。

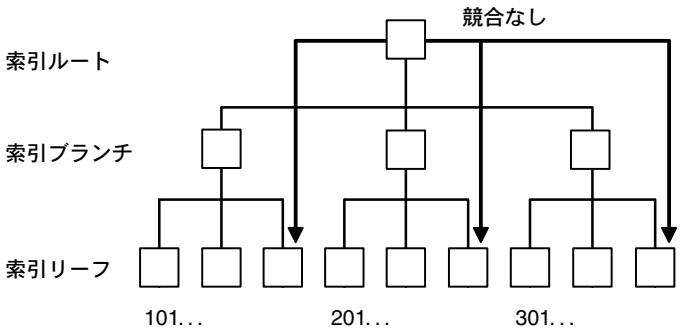
複数のインスタンスが、索引を持つ表に同時に行を挿入する場合、索引値がインスタンス別にパーティション化できる場合を除いて、索引ブロックに対する競合が発生してパフォーマンスが低下します。図 8-1 に、すべてのインスタンスが同じ索引リーフ・ブロック（*n*）に挿入しようとしている状況を示します。

図 8-1 1 つの索引ブロックに対する競合



この問題を回避するには、図 8-2 に示すように、各インスタンスが固有のツリーに挿入するようにします。

図 8-2 競合がない索引



次の式で索引値を計算します。

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・グループとの対応付け

この項では、次の対応付けについて説明します。

- インスタンスと空きリスト・グループとの対応付け
- ユーザー・プロセスと空きリスト・グループとの対応付け
- PCM ロックと空きリスト・グループとの対応付け

インスタンスと空きリスト・グループとの対応付け

次のようにして、インスタンスとエクステントまたは空きリスト・グループを対応付けます。

INSTANCE_NUMBER パラメータ	INSTANCE_NUMBER 初期化パラメータと同時に様々な SQL 句を使用して、データ・ブロックのエクステントをインスタンスに対応付けることができます。
SET INSTANCE 句	ALTER SESSION 文の SET INSTANCE 句を使用して、セッションが接続しているインスタンスに関係なく、そのセッションが特定のインスタンスに対応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できます。次に例を示します。 `ALTER SESSION SET INSTANCE = inst_no`

SET INSTANCE 句は、1 つのインスタンスに障害が発生し、他のインスタンスに接続する場合に有効です。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。インスタンス全体にユーザーが分散され、データが適切にパーティション化され、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。1 つのインスタンスに障害が発生した場合、すべてのユーザーを他のインスタンスに移動しても、データのパーティション化には影響しません。これは、それぞれの新しいセッションは、障害の発生したインスタンスに対応付けられている元の空きリスト・グループを使用できるためです。

ユーザー・プロセスと空きリスト・グループとの対応付け

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 に基づいて、空きリストに自動的に対応付けられます。

(oracle_pid modulo #free_lists_for_object) + 1

ALTER SESSION SET INSTANCE 文を使用することによって、特定のインスタンスに対応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・グループとの対応付け

表の各エクステントが別々のデータ・ファイルにある場合は、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 ロックの集合を、それぞれ 1 つの空きリスト・グループのみに対応付けることができます。

> **参照：** インスタンス、ユーザーおよびロックと空きリストとの対応付けの詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

エクステントの事前割当て

この項では、エクステントの事前割当ての方法を説明します。この方法は有効ですが、エクステント割当てへの静的アプローチには、多少のデータベース管理上のオーバーヘッドが発生します。

- ALLOCATE EXTENT 句
- 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 句

ALTER TABLE 文または ALTER CLUSTER 文の ALLOCATE EXTENT 句を使用すると、エクステントのサイズ、データ・ファイルおよび空きリストのグループを指定するパラメータを使用して、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。

排他モードおよび共有モード 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規則に従います。トランザクションは、このインスタンスに対して、マスター空きリストまたは特定の空きリスト・グループを使用できます。

SIZE パラメータ ALLOCATE EXTENT 句の SIZE パラメータには、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、エクステント・サイズは、記憶域パラメータ NEXT および PCTINCREASE の値に従って計算されます。

SIZE の値は、後続のエクステント割当てを計算するためには使用しません。後続のエクステント割当ては、NEXT および PCTINCREASE に基づいて判断されます。

DATAFILE パラメータ このパラメータは、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。このパラメータを省略すると、該当する表を含む表領域にあるすべてのアクセス可能なデータ・ファイルから領域が割り当てられます。

ファイル名は、大文字 / 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と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 パラメータ このパラメータは、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取得します。最小インスタンス番号は、0（ゼロ）ではなく 1 です。最大値は、オペレーティング・システム固有です。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、割り当てられる空きリスト・グループが判断されます。

$$modulo(n,\#\_freelistgroups) + 1$$

INSTANCE パラメータを指定しないと、新しい領域は表に割り当てられますが、空きリスト・グループには割り当てられません。このような領域は、他の領域が使用できない場合に、必要に応じて空きブロックのマスター空きリストに含められます。

注意： INSTANCE の値には、実際のインスタンス番号ではなく、使用する空きリスト・グループの番号に対応した値を使用してください。

参照： INSTANCE パラメータの詳細は、4-5 ページの「インスタンスの停止」を参照してください。

MAXEXTENTS、MINEXTENTS および INITIAL パラメータの設定

特定のインスタンスに対応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在の数（事前割当てされたエクステントに MINEXTENTS を加えた数）に設定することによって、自動割当てを防止できます。MINEXTENTS を 1（デフォルト）に設定し、INITIAL をその最小値（2 つのデータ・ブロック、またはブロック・サイズが 2048 バイトの場合は 10KB）に設定して表またはクラスタを作成すると、初期割当てを最小化できます。

データ・ブロックに対するインスタンス間の競合を最小化するには、各表に複数のデータ・ファイルを作成し、各インスタンスを異なるファイルに対応付けます。

システムのノード数を増やす場合は、追加インスタンスを考慮して、現行のインスタンス数より多くの空きリスト・グループを持つ表またはクラスタを作成します。これらの空きリスト・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、自動的に割り当てら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のみです。

データ・ブロックを空きリスト・グループに対応付けるには、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って PCTUSED 下に入れるか、または特別にその空きリスト・グループに割り当てる必要があります。したがって、使用されていない空きリスト・グループは、未使用の空きデータ・ブロックから切り離さ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のみを挿入および更新に使用できます。これによって、インスタンスが他のインスタンスに割り当てられた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、別のインスタンスが INSTANCE_NUMBER で再起動されない限り、別の空きリストにあるデータ・ブロックを使用することはできません。また、ALTER SESSION 文を使用して、セッション中にインスタンス番号をオーバーライドできます。

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 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 記憶域パラメータの値より多くのインスタンスを実行する Parallel Server 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 番目に起動されるすべてのインスタンスは、同じ空きリスト・グループに対応付けられます。

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 つ持つ 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（MINEXTENTS と FREELIST GROUPS の値の合計）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 文を使用して MAXEXTENTS を増加させます。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* パラメータを使用して、ロック境界内の最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。内容は次のとおりです。

- データ・ブロック・アドレスからロック名への変換
- ALLOCATE EXTENT 構文での !blocks

データ・ブロック・アドレスからロック名への変換

第 7 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、それを保護するロック名との間の変換を指定します。次に構文を示します。

```
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保護するために、1000 個の異なるロック名を使用する必要が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保護されます。

```
GC_FILES_TO_LOCKS = "1000!25"
```

ALLOCATE EXTENT 構文での !*blocks*

同様に、*!blocks* パラメータによって、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られるブロックの割合を指定できます。一度に 255 ブロックまで指定できます。次に例を示します。

```
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。

参照： ALLOCATE EXTENT 構文の詳細は、第 9 章を参照してください。

未使用領域の識別および割当て解除

この項の内容は次のとおりです。

- 未使用領域の識別
- 未使用領域の割当て解除
- 削除または更新によって解放された領域

未使用領域の識別

DBMS_SPACE パッケージには、表内の空きリスト・グループの使用済領域および未使用領域の量を判断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、領域を割り当て直す必要があるインスタンスを判断することができます。このパッケージは、『Oracle8*i* ユーティリティ・ガイド』で説明されているとおり、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インスタンスに割り当てた未使用領域は、割当て解除できません。これは、未使用領域が最高水位標より下にあるためです。

ただし、領域が最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 を ALTER TABLE 文または ALTER INDEX 文とともに使用して、セグメントを最高水位標まで切り捨てます。

削除または更新によって解放された領域

削除または更新による行サイズの縮小によって解放されたブロックは、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。

9

インスタンス・ロックの設定

この章では、インスタンス・ロックの設定方法について説明します。内容は次のとおりです。

- GC_FILES_TO_LOCKS の設定 : 各データ・ファイルに対する PCM ロック
- GC_FILES_TO_LOCKS の設定に対するヒント
- その他の GC_* パラメータの設定
- PCM ロックのチューニング
- PCM ロックおよび非 PCM ロックの名前および書式

GC_FILES_TO_LOCKS の設定 : 各データ・ファイルに対する PCM ロック

GC_FILES_TO_LOCKS 初期化パラメータを設定して、データ・ファイルまたはデータ・ファイルの集合にあるデータ・ブロックを処理する、パラレル・キャッシュ管理（PCM）ロックの数を指定します。この項の内容は次のとおりです。

注意 : データ・ファイルを追加またはサイズ変更する場合、表領域を作成する場合、あるいは表領域およびそのデータ・ファイルを削除する場合は、必ず、Parallel Server が使用可能な Oracle を再起動する前に、GC_FILES_TO_LOCKS の値を調整してください。

参照 : 単一 PCM ロックで処理されるデータ・ブロック数の決定方法の詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

GC_FILES_TO_LOCKS 構文

GC_FILES_TO_LOCKS パラメータを設定する構文によって、データベース・アドレスとデータベース・ブロックのクラスの間の変換、およびそれを保護するロック名が指定されます。GC_FILES_TO_LOCKS パラメータに指定されていないファイルに対しては、この変換を指定できません。

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	次のように、単一のファイル、ファイルの範囲、またはファイルおよび範囲のリストを指定します。 *fileidA*[-*fileidC*][,*fileidE*[-*fileidG*]] ... データ・ディクショナリ・ビュー DBA_DATA_FILES を問い合せて、ファイル名とファイル ID 番号の間で対応するものを検索します。
#*locks*	*file_list* に割り当てる PCM ロックの数を設定します。#*locks* の値を 0（ゼロ）に設定すると、固定ロックのかわりに解放可能ロックが使用されます。
!*blocks*	各ロックによって処理される、連続するデータ・ブロックの数を指定します。
EACH	#*file_list* 内の各ファイルに割り当てるロックの数を、#*locks* に設定します。
R	ロックが解放可能であることを指定します。ロックが不要になった場合に、インスタンスによって解放されます。解放可能 PCM ロックは、プール GC_RELEASABLE_LOCKS から取り出されます。

注意： GC_ROLLBACK_LOCKS では、同じ構文が使用されます。GC_FILES_TO_LOCKS パラメータの引用符内には、空白を使用しないでください。

データ・ファイルに対する PCM ロックのマッピングの制御に加えて、GC_FILES_TO_LOCKS は、デフォルト・バケット内のロックの数を制御します。Oracle は、GC_FILES_TO_LOCKS に明示的に指定されていないすべてのファイルに対して、デフォルト・バケットを使用します。このパラメータの値には 0（ゼロ）を使用できます。また、デフォルトは "0=0" です。たとえば、"0=100"、"0=100R"、"0-9=100EACH" などに設定できます。デフォルトでは、このバケット内のロックは解放可能です。ただし、固定ロックも使用できます。

GC_FILES_TO_LOCKS パラメータに R オプションを使用して、解放可能 PCM ロックを指定できます。Oracle は、GC_RELEASABLE_LOCKS のプールから、1:N の解放可能 PCM ロックを取り出します。

REACH は、"R" と "EACH" が組み合されたキーワードです。たとえば、GC_FILES_TO_LOCKS="0-9=100REACH" のようになります。EACHR は、有効なキーワードではありません。

EACH および *!blocks* を省略すると、*#locks* PCM ロックが集合的に *file_list* に割り当てられ、個々の PCM ロックによって *file_list* の各ファイルのデータ・ブロックが処理されます。ただし、あるデータ・ファイルに PCM ロックより少ないデータ・ブロックが含まれる場合、いくつかの PCM ロックは、そのデータ・ファイルのデータ・ブロックを処理しません。

!blocks のデフォルト値は 1 です。*blocks* を指定する場合、連続するデータ・ブロックは、*#locks* PCM ロックの各自によって処理されます。*blocks* の値を指定するには、「!」セパレータを使用する必要があります。主に *blocks* を指定し、EACH キーワードは指定しないで、複数のデータ・ファイルを処理する PCM ロックの集合を割り当てます。*blocks* を使用して、単一のデータ・ファイルを処理する PCM ロックの集合を割り当てることができます。この単一データ・ファイルでは、PCM ロックの競合は最小になり、PCM ロック管理が削減されます。

空きリスト・グループの使用によって取得されるデータ・パーティション化との干渉を回避するには、必ず、*!blocks* に値を設定します。通常、ディスク領域を事前に割り当てる必要はありません。行が表に挿入され、新しいエクステントの割当てが必要な場合、GC_FILES_TO_LOCKS の *!blocks* で指定された連続するブロックは、1 つのインスタンスに対応付けられた空きリスト・グループに割り当てられます。

固定ロックの例

インスタンスのパラメータ・ファイルに次の行を追加して、300 のロックをファイル 1 に、100 のロックをファイル 2 に割り当てることができます。

```
GC_FILES_TO_LOCKS = "1=300:2=100"
```

次のエントリによって、合計 1500 のロックが、ファイル 1、2 および 3 に 500 ずつ割り当てられます。

```
GC_FILES_TO_LOCKS = "1-3=500EACH"
```

対照的に、次のエントリでは、3 つのファイルに合計 500 のロックが割り当てられます。

```
GC_FILES_TO_LOCKS = "1-3=500"
```

次のエントリは、1000 の個別のロックを使用して、ファイル 1 を保護する必要があることを示します。ファイル内のデータは、25 ブロックごとに保護されます。

```
GC_FILES_TO_LOCKS = "1=1000!25"
```

解放可能ロックの例

あるグループ要素を持つデータ・ブロックに対して粒度が低い解放可能ロックを指定するには、インスタンスのパラメータ・ファイルに次のエントリを指定します。

```
GC_FILES_TO_LOCKS="1=0!4"
```

これによって、ファイル 1 に対してグループ要素 4 を持つロックが指定されます。

次のエントリは、1000 の解放可能ロックが 25 ブロックごとにファイル 1 を保護することを示します。

```
GC_FILES_TO_LOCKS = "1=1000!25R"
```

GC_FILES_TO_LOCKS の設定に対するガイドライン

次のガイドラインを使用して、GC_FILES_TO_LOCKS パラメータを設定します。

- 必ず、すべてのデータ・ファイルを GC_FILES_TO_LOCKS に指定します。
- GC_FILES_TO_LOCKS には、同じデータベースにアクセスする各インスタンスに対して同じ値を割り当てます。
- 1 つのデータ・ファイルに指定する PCM ロックの数が、データ・ファイル内のブロックの数を超えないようにします。これによって、データ・ファイルのサイズが増加した場合に、新しいブロックが別の PCM ロックで処理されることが保証されます。

  データ・ファイルが AUTOEXTEND 句で定義されているか、または ALTER DATABASE... DATAFILE... RESIZE 文を発行する場合は、データ・ファイルのサイズが増加していないかを、定期的に監視する必要があります。データ・ファイルのサイズが増加している場合、できるだけ早くパラメータ GC_FILES_TO_LOCKS を更新します。その後、Oracle Parallel Server を停止し、再起動します。

  > **注意 :** Oracle Parallel Server を再起動する必要はありませんが、Oracle Parallel Server を停止および再起動しないと、ロックはさらに多くのブロックを処理します。

  file_list に指定された PCM ロックの数が、データ・ファイル内のデータ・ブロックの実際の数より少ない場合、DLM はいくつかの PCM ロックを使用して、指定した数より多くのデータ・ブロックを処理します。これによって、パフォーマンスが低下する可能性があります。したがって、必ず、十分な PCM ロックが使用可能であることを確認してください。
- 新しいデータ・ファイルを追加する場合、必ずそれらのロックを GC_FILES_TO_LOCKS に指定し、予備 PCM ロックの自動割当てを回避します。

  ALTER TABLESPACE... ADD DATAFILE 文を使用して、Oracle Parallel Server の実行中に、データ・ファイルを追加する必要がある場合もあります。これを行う場合、GC_FILES_TO_LOCKS の設定をできるだけ早く更新して、Oracle Parallel Server を停止し、再起動します。
- 異なるインスタンスがアクセスする分割データを作成することによって発生するリソースの競合を削減するには、データ・ファイルを異なるディスクに置きます。

GC_FILES_TO_LOCKS を使用して、個別のデータ・ファイル内のデータ・ブロックを処理する PCM ロックを割り当てます。

- 頻繁に変更されない索引データを含むブロックに対しては、比較的少ない PCM ロックを指定します。索引は、独自の表領域または表領域内の独自のデータ・ファイル内に置きます。これによって、PCM ロックの個別の集合が索引に割り当てられます。読込み専用索引に対しては、1 つのロックのみを使用します。
- GC_FILES_TO_LOCKS に指定されていないファイルは、解放可能ロックを使用します。

GC_FILES_TO_LOCKS の設定に対するヒント

GC_FILES_TO_LOCKS の設定は、Oracle Parallel Server の重要なチューニング作業です。この項では、パラメータ設定が最高のパフォーマンスを提供しているかを確認するために有効な、いくつかの簡単なチェックについて説明します。この項の内容は、次のとおりです。

拡張の余地の提供

継続的に実行しているサイトは、停止してパラメータ値を調整する余裕がありません。したがって、これらのパラメータのサイズを設定する場合、拡大または拡張するファイルに余地を提供します。

また、データ・ファイルを追加またはサイズ変更する場合、表領域を作成する場合、あるいは表領域およびそのデータ・ファイルを削除する場合は、必ず、Oracle Parallel Server を再起動して使用可能にする前に、GC_FILES_TO_LOCKS の値を調整します。

ロックの有効数のチェック

割り当てられたロックの数が、割り当てられたデータ・ブロックの数より多くないことをチェックします。

注意： 表に挿入する直前の場合、現在割り当てられているブロックが、0（ゼロ）である場合があります。

FILE_LOCK データ・ディクショナリ・ビューをチェックし、ファイルごとに割り当てられているロックの数を確認します。V$DATAFILE ビューをチェックし、データ・ファイルの最大サイズを確認します。

> **参照：** FILE_LOCK および V$DATAFILE 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。

有効なロック割当てのチェック

次のようにして、ロック割当ての問題を回避します。

- ロールバック・セグメントのみを保持するファイルに、ロックを割り当てないでください。
- 内部一時表に対する一時データのみを保持するファイルに、ロックを割り当てないでください。
- 読込み専用オブジェクトをグループ化し、1 つのロックのみをそのファイルに割り当てます。これは、ファイルに 1 回も書込みが行われていない場合、またはブロックが変更されていても、ブロックのクリーン・アウト中に実行される場合にのみ有効です。

表領域に対する読込み専用の設定

表領域が読込み専用の場合、Oracle でそれを読込み専用に設定することを考慮します。これによって、データベースに書込みが行われず、PCM ロックが表領域で使用されないことが保証されます。ただし、例外として、表領域が予備ロックに関して競合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。

ファイルの妥当性チェック

次の構文を使用して、各ファイル内のオブジェクトの数を決定します。

```
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 を実行して、EXT_TO_OBJ ビューを使用します。同じファイル内にオブジェクトが共存できることを確認します。つまり、GC_FILES_TO_LOCKS 設定に互換性があることを確認します。

パラメータ値の変更なしでのデータ・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、PCM ロックの配分の結果を考慮します。ロックは、インスタンスを停止してパラメータを変更し、データベースを再起動しなければ、このファイルに割り当てることができません。これは、本番データベースには不可能な場合があります。この場合、Oracle は、残りのロックのプールからデータ・ファイルにロックを提供します。そのファイルは、GC_FILES_TO_LOCKS パラメータに設定しなかったすべてのファイルと競合します。

その他の GC_* パラメータの設定

この項では、次の 2 つの追加 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 に対して、デフォルトの設定を使用することをお薦めします。これは、DB_BLOCK_BUFFERS の値です。通常、この推奨事項によって、最適なパフォーマンスが得られます。ただし、GC_RELEASABLE_LOCKS をデフォルトより小さく設定すると、メモリーを節約できます。GC_RELEASABLE_LOCKS の値が低すぎると、パフォーマンスが低下する場合があります。

V$SYSSTAT ビューの統計情報 global cache freelist waits には、システムの解放可能ロックが不足した回数が表示されます。これが発生し、global cache freelist waits に 0（ゼロ）以外の値が表示された場合、GC_RELEASABLE_LOCKS の値を大きくします。

GC_ROLLBACK_LOCKS の設定

固定ロックを使用している場合、割り当てられたロックの数が割り当てられたデータ・ブロックの数を超えていないことをチェックします。表に挿入する直前の場合、現在割り当てられているブロックが、0（ゼロ）の場合があります。次のように入力して、ロールバック・セグメントに割り当てられているブロックの数を検索します。

```
SELECT S.SEGMENT_NAME NAME,
       SUM(R.BLOCKS) 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;
```

この問合せによって、各ロールバック・セグメントに割り当てられているブロックの数が表示されます。多くの不要な強制読込み / 書込みが UNDO ブロックで発生する場合は、解放可能ロックを使用します。GC_ROLLBACK_LOCKS のデフォルトの設定は次のとおりです。

```
GC_ROLLBACK_LOCKS = "0-128=32!8REACH"
```

これによって、ロールバック・セグメント 0 ～ 128 がロックで保護されます。最初の 129 ロールバック・セグメントは 32 の解放可能ロックを持ち、8 つにグループ化されます。つまり、各ロックは 8 つの連続するブロックを処理します。

パラメータ GC_ROLLBACK_LOCKS は、GC_FILES_TO_LOCKS パラメータとほぼ同様に引数をとります。たとえば次のとおりです。

```
GC_ROLLBACK_LOCKS="0=100:1-10=10EACH:11-20=20EACH"
```

この例では、システム・ロールバック・セグメントであるロールバック・セグメント 0 は、100 のロックを持ちます。ロールバック・セグメント 1 ～ 10 はそれぞれ 10 のロックを持ち、ロールバック・セグメント 11 ～ 20 はそれぞれ 20 のロックを持ちます。

注意： GC_ROLLBACK_LOCKS を使用して、ロールバック・セグメントでロックを共有することはできません。

次の例では、各ロールバック・セグメントは 100 のロックを持つため、1 つ目の例は無効で、2 つ目の例は有効です。

無効

```
GC_ROLLBACK_LOCKS="1-10=100"
```

有効

```
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考慮する問題について説明します。

- false ping の検出
- PCM ロック変換に必要な時間の決定
- PCM ロック変換の完了を待機するセッションの識別

この項の内容は次のとおりです。

false ping の検出

false ping は、異なる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保護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保護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、同じロックが両方のブロックを処理するためです。

false ping アクティビティを表示できる統計情報はありません。false ping の検出は、周囲の状況から判断するしかありません。この項では、対象となるアクティビティを検出する方法を説明します。

次の SQL 文は、書込みの原因となるロック・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 forced writes';
```

表 9-1 に、ping 率の解析方法を示します。

表 9-1 ping 率の解析

ping 率	意味
< 1	false ping が発生している場合がありますが、ping へ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を書き込んでいるため、ロック・アクティビティに対する書込みは発生していません。これは、「soft ping」とも呼ばれます。I/O が、ping に対してではなく、ロック・アクティビティにのみ要求されていることを表します。
= 1	書込みが発生する可能性がある各ロック・アクティビティで、実際に書込みが発生します。false ping が発生している場合があります。
> 1	false ping が確実に発生しています。

次の式を使用して、false ping の確率を計算します。

$$\frac{(ping\_rate - 1)}{ping\_rate} * 100$$

書込みの合計数をチェックし、false ping による書込みの数を計算します。

```
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 forced writes'
AND Y.NAME = 'physical writes';
```

次の SQL 文によって、*ping_rate* が導出されます。

```
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 forced writes';
```

目標は、全体の ping のみでなく、false ping を削減することです。このためには、GC_FILES_TO_LOCKS 内のインスタンス・ロックの配分を確認して、ファイル内のデータをチェックします。

PCM ロック変換に必要な時間の決定

PCM ロックの取得に必要な時間を確実にチェックします。この時間はシステム間で異なります。次の SQL 文を入力して、ロックの取得期間を検索します。

```
SELECT *
FROM V$SYSTEM_EVENT
WHERE EVENT LIKE 'global cache%'
```

次のように出力されます。

```
EVENT                    TOTAL_WAITS TOTAL_TIMEOUTS TIME_WAITED AVERAGE_WAIT
-------------------------------------------------------------------------------
global cache lock open s         743              0         494    .66487214
global cache lock open x        5760              0        5945   1.03211806
global cache lock null to s      263              0         697   2.65019011
global cache lock null to x     2149              0        7804   3.63145649
global cache lock s to x        1427              0        1394   .976874562
global cache cr request        25248              5        4729   .187301965
global cache lock busy            21              0          46   2.19047619
global cache bg acks               2              0           0            0
```

PCM ロック変換の完了を待機するセッションの識別

次の SQL 文を入力して、PCM ロック変換の完了を待機中のセッション、および待機が終わった直後のセッションを判断します。

```
SELECT *
FROM V$SESSION_WAIT
WHERE EVENT LIKE 'global cache%' AND 'wait_time = 0'
```

PCM ロックおよび非 PCM ロックの名前および書式

この項の内容は次のとおりです。

ロック名およびロック名の書式

Oracle は、次の書式で、すべてのエンキューおよびインスタンス・ロックに名前を付けます。

- *type ID1 ID2*
- *type*, *ID1*, *ID2*
- *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 表に記述されています。
ID1	DLM に使用される、1 番目のロック識別子です。この識別子に対する規則は、ロック・タイプごとに異なります。
ID2	DLM に使用される、2 番目のロック識別子です。この識別子に対する規則は、ロック・タイプごとに異なります。

たとえば、領域管理ロックの名前は ST00 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 表には、ローカル・インスタンスによって現在保持されているか、または要求されているローカルおよびグローバルな Oracle エンキューが表示されます。ロック名は、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 ロックの名前

すべての PCM ロックは、バッファ・キャッシュ管理ロックです。バッファ・キャッシュ管理ロックのタイプは BL になります。PCM ロックの名前の構文は、*type ID1 ID2* です。各項目の意味は次のとおりです。

type	PCM ロックはバッファ・ロックであるため、常に BL です。
ID1	ブロック・クラスです。
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取得されたロック要素（LE）索引番号です（V$LOCK_ELEMENT 固定ビューを参照）。解放可能ロックの場合、*ID2* は、ブロックのデータベース・アドレスです。

次に、PCM ロックの名前の例を示します。

BL (1, 100)	ロック要素 100 を持つデータ・ブロックです。
BL (4, 1000)	ロック要素 1000 を持つセグメント・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 を持つロールバック・セグメント・ヘッダーです。ロールバック・セグメントの計算式は、7 + (10 * 2) です。

非 PCM ロックの名前

非 PCM ロックには、多くの異なる名前があります。表 9-2 に、名前のリストを示します。

表 9-2 非 PCM ロックのタイプおよび名前

タイプ	ロック名
CF	制御ファイル・トランザクション
CI	インスタンス間コール起動
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	分散リカバリ
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・リカバリ

表 9-2 非 PCM ロックのタイプおよび名前（続き）

タイプ	ロック名
IS	インスタンス状態
MM	マウント定義
MR	メディア・リカバリ
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	システム変更番号
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・リカバリ
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
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

参照： 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。

10

ロックおよびリソースに対する DLM 容量の確保

共有リソースに対する競合を削減し、Oracle Parallel Server のパフォーマンスを最適化するには、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、分散ロック・マネージャが適切に構成されていることを確認します。この章の内容は次のとおりです。

- 分散ロック・マネージャ容量の計画の概要
- 分散ロック・マネージャ容量の計画
- Oracle 初期化パラメータの調整
- パフォーマンスを最適化する表ロックの最小化

ロックおよびリソースに対するシステム設定の調整を完了している場合は、次の項を参照して、SQL*Loader を使用してデータをデータベースにロードできます。

- SQL*Loader の使用

> **参照：** パラレル・キャッシュ管理および非パラレル・キャッシュ管理のロックの概要は、『Oracle8*i* Parallel Server 概要』を参照してください。

分散ロック・マネージャ容量の計画の概要

システムのロックを管理するには、パラレル・キャッシュ管理（PCM）ロックの割当て計画のみでは不十分です。明示的に PCM ロックを割り当てる他に、必要なすべての PCM ロックと非 PCM ロック、およびリソースに対して、各ノード上に分散ロック・マネージャ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くすると、多くのリソースに対する要求が増加することも考慮する必要があります。

非 PCM ロックには多くの異なるタイプがあり、それぞれ処理方法が異なります。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もることはできます。また、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（あるいはその両方）を調整することもできます。表ロックを最小化して、パフォーマンスを最適化することもできます。

分散ロック・マネージャ容量の計画

分散ロック・マネージャ容量は、LM_RESS および LM_LOCKS パラメータの設定によって決定されます。分散ロック・マネージャは、初期化パラメータ・ファイルの他のパラメータ設定に基づいて、LM_RESS および LM_LOCKS の値を自動的に計算します。Oracle が LM_RESS および LM_LOCKS に対して行った設定は、起動時に `alert.log` ファイルに表示されます。ただし、エンキュー・リソース使用率はアプリケーションに依存するため、これらの設定は単なる見積りです。

共有プールの領域が不足した場合、または V$RESOURCE_LIMIT ビューに表示される最大値の使用率が、Oracle が LM_RESS および LM_LOCKS に設定した値より大きい場合、LM_RESS および LM_LOCKS の設定値を増やし、V$RESOURCE_LIMIT の統計情報を再検査します。そうでない場合は、LM_RESS および LM_LOCKS を設定する必要はありません。

リソースおよびロックの動的割当ての回避

必要なロックまたはリソース数が、Oracle の初期割当て量を上回ると、Oracle は、システム・グローバル領域の共有プールから追加ロックおよびリソースを割り当てます。この機能によって、インスタンスの停止が防止されます。

動的割当てによって、Oracle は、次回のデータベース起動時に初期化パラメータを調整する必要があることを示す警告ファイルに、メッセージを書き込みます。パフォーマンスおよびメモリー使用量は、動的割当てによって悪影響を受ける場合があるため、必要なロックおよびリソースを正確に計算することをお薦めします。

SHARED_POOL_SIZE の推奨設定

SHARED_POOL_SIZE の推奨デフォルト値は、64 ビットのアプリケーションでは 16MB、32 ビットのアプリケーションでは 8MB です。

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

ただし、各インスタンスに必要以上に大きい実パラメータ値は指定しないでください。これらのパラメータを必要以上に高く設定すると、オーバーヘッドが発生します。

パフォーマンスを最適化する表ロックの最小化

挿入、削除および更新に対して、DML ロックなどの表ロックを取得すると、Oracle Parallel Server のパフォーマンスが低下します。表にロックを保持しているすべてのインスタンスがこれらのロックを解放する必要があるため、Oracle Parallel Server での表のロックは望ましくありません。次の項目で説明する 2 つの方法のいずれかを使用して、これらのロックを完全に使用不可にすることを検討してください。

- 表ロックの使用禁止
- DML_LOCKS を 0（ゼロ）に設定

注意： すべての表ロックを使用不可にする場合、インスタンスまたは表のいずれかに対して DDL コマンドを実行することはできません。

表ロックの使用禁止

ユーザーが表ロックを取得しないようにするには、次の文を使用します。

```
ALTER TABLE table_name DISABLE TABLE LOCK
```

表ロックが使用不可のときにユーザーが表をロックしようとすると、エラーになります。

表ロックを再度使用可能にするには、次の文を使用します。

```
ALTER TABLE table_name ENABLE TABLE LOCK
```

一度この構文を実行すると、実行中のすべてのトランザクションがコミットした後で、表ロックが使用可能になります。この文は、ENABLE 文の発行後、新しいトランザクションが開始するのを待つ必要はありません。

表の表ロックが使用可能か使用不可かを判断するには、データ・ディクショナリ表 USER_TABLES の列 TABLE_LOCK を問い合せます。DBA_TABLES または ALL_TABLES に選択権限がある場合、他のユーザーの表の表ロック状況を問い合せることができます。

DML_LOCKS を 0（ゼロ）に設定

表ロックは、初期化パラメータ DML_LOCKS によって設定されます。DROP TABLE、CREATE INDEX および LOCK TABLE 文が必要でない場合、DML_LOCKS を 0（ゼロ）に設定し、ロック変換を最小化して、パフォーマンスを最適化します。

注意： 1 つのインスタンスで DML_LOCKS を 0（ゼロ）に設定した場合、すべてのインスタンスでこれを 0（ゼロ）に設定してください。その他の値を使用する場合は、すべてのインスタンスでこのパラメータを同じ値にする必要はありません。

SQL*Loader の使用

一度分散ロック・マネージャを構成すると、いつでもデータをデータベースにロードできます。効率的にデータをロードするには、SQL*Loader を使用します。

SQL*Loader を使用すると、Oracle Parallel Server データベースに対して、従来型パス・ロードまたはダイレクト・パス・ロードのいずれかを使用できます。ただし、各方法にはそれぞれメリットおよびデメリットがあります。

従来型パスの方法には、Oracle が通常のユーザー・ベースの挿入に適応するものと同じデータ整合性規則が適用されます。REDO ログ、ロールバック・セグメント、索引およびトリガーは、通常の挿入処理の場合と同様に機能します。全体のデータ整合性は処理速度より重要であるため、ロード処理の所用時間の削減が重要でない場合は、データのロードには従来型パスの方法の使用をお薦めします。

従来型パスの方法を使用し、ロードをパラレルに実行する場合、次のことを回避する必要があります。

- 入力ファイルでの読込みの競合
- 影響を受けたデータ・ファイルでの書込みの競合
- すべてのノードの CPU の飽和状態
- 必要なエクステントに対する領域不足

SQL*Loader のダイレクト・パスの方法によって、システム・グローバル領域内のほとんどの処理が省略されます。SQL*Loader によって表に行が追加された場合、Oracle はロールバック・セグメントにその行を記録しません。そのため、Oracle は、新しいデータからの問合せに対する読取り一貫性のブロックを作成できません。ただし、ユーザーは、データがディスクに書き込まれた後で、新しい記録を読み込むことができます。

パラレル・ダイレクト・ロードによって、ブロック・イメージが同じデータ・ファイル・ブロック・アドレスに書き込まれることがあります。これを回避するには、PARALLEL キーワードを使用して、制御ファイルにフラグを設定します。各パラレル SQL*Loader セッションは、フラグをチェックし、同じ表に対して実行する非パラレル・ダイレクト・ロードがないことを確認します。これによって、Oracle は、各セッションに新しいエクステントを作成します。

> **参照：** SQL*Loader の詳細は、『Oracle8*i* ユーティリティ・ガイド』を参照してください。

第IV部

Oracle Parallel Server のパフォーマンスの監視およびチューニング

第 IV 部では、パフォーマンス統計情報を監視する方法およびパラメータを調整して Oracle Prallel Server のパフォーマンスを向上させる方法について説明します。第 IV 部に含まれる章は、次のとおりです。

- 第 11 章「一般的なチューニングの推奨事項」
- 第 12 章「Oracle Parallel Server およびインスタンス間パフォーマンスのチューニング」

11

一般的なチューニングの推奨事項

この章では、ご使用の Oracle Parallel Server アプリケーションをチューニングする一般的な方法を示し、Oracle Parallel Server のチューニング問題の概要について説明します。内容は次のとおりです。

- Oracle Parallel Server のチューニングの概要
- Oracle Parallel Server のパフォーマンスの監視に対する統計情報
- 同期化コストの決定
- グローバルおよびローカルな作業の割合の計測
- ロック競合によるグローバル・キャッシュの同期化コストの計算
- Oracle Parallel Server ベースのアプリケーションにおける問題の解決

> **参照：** Oracle Parallel Server 管理におけるパフォーマンスの監視およびチューニングの詳細は、『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。

Oracle Parallel Server のチューニングの概要

過去の経験から、Oracle Parallel Server アプリケーションを配置する前に、多くのパフォーマンスの問題を予測できます。Oracle Parallel Server アプリケーションをチューニングする場合、単一インスタンスのチューニングが有効な場合もあります。ただし、Parallel Server 固有の目標を考慮して、バッファ・キャッシュ、共有プールおよび共有ディスク・サブシステムを効率的にチューニングする必要があります。

Oracle Parallel Server では、単一インスタンス環境では使用されないパラメータが導入されています。このようなパラメータの多くはチューニング可能なパラメータであり、正しく設定すると、Parallel Server のパフォーマンスが大幅に向上します。ただし、最も効果的なチューニングでも、分析またはデータベースとアプリケーション設計のミスが引き起こした問題は解決できません。

Oracle Parallel Server のチューニングには、いくつかのビューの監視および Parallel Server 固有の統計情報の収集も必要です。これを行うには、この章で説明する方法を使用してください。

Oracle Parallel Server のパフォーマンスの監視に対する統計情報

この項では、アプリケーションを監視およびチューニングするために、具体的に使用可能な統計情報について説明します。この項の内容は次のとおりです。

- V$SYSSTAT および V$SYSTEM_EVENT の内容
- チューニングに対する統計情報の記録

Oracle は、ローカルなシステム・グローバル領域に多くの統計情報を保持しています。この章で説明されているように、動的パフォーマンス・ビューまたは V$ 表に対して SQL を使用して、これらの統計情報にアクセスします。

また、UTLBSTAT や UTLESTAT などのユーティリティを使用して、ある期間、統計情報を記録し、一定間隔で統計情報を獲得することもできます。統計情報は、メッセージ要求カウンタまたは定期的な統計情報として使用できます。メッセージ・カウンタには、特定の種類のロック変換の数を示す統計情報が含まれます。たとえば、Oracle Parallel Server の定期的な統計情報は、特定の操作での読込みおよび書込み I/O に対する、合計または平均待機時間を示します。

Oracle Parallel Server において最も重要な統計情報を、次に示します。

- consistent gets、db block gets、db block changes、waits for busy buffers などのキャッシュ関連の統計情報
- physical reads、physical writes、cross-instance writes、wait times for reads and writes などの I/O 統計情報
- global cache lock gets、global cache converts、waits for events such as null-to-X conversions などの、グローバル・キャッシュ・ロックの要求および待機時間

Oracle Parallel Server 固有の統計情報を表示する最も重要なビューは、V$SYSSTAT および V$SYSTEM_EVENT です。

V$SYSSTAT および V$SYSTEM_EVENT の内容

この項では、V$SYSSTAT および V$SYSTEM_EVENT の内容を説明し、その他の Oracle Parallel Server 固有のビューを示します。

V$SYSSTAT の統計情報

V$SYSSTAT ビューには、次の統計情報が含まれます。

- consistent gets
- db block gets
- db block changes
- physical reads
- physical writes
- DBWR cross-instance writes
- global cache get
- global cache converts
- global cache get time
- global cache convert time
- global cache cr blocks received

- global cache cr blocks receive time
- global cache cr timeouts
- global cache cr convert timeouts
- CPU used by this session

V$SYSTEM_EVENT の統計情報

V$SYSTEM_EVENT ビューには、次の統計情報が含まれます。

- db file sequential read
- db file scattered read
- db file parallel write
- log file sync
- global cache lock null to x
- global cache lock null to s
- global cache lock s to x
- global cache lock open x
- global cache lock open s
- global cache cr request
- global cache lock busy
- buffer busy
- buffer busy due to global cache
- enqueue
- row cache
- library cache pin
- lkmgr wait for remote messages

その他の Parallel Server 固有のビュー

その他の重要な統計情報は、次のパフォーマンス問題に関連します。

- バッファ・キャッシュの使用量
- ロック変換のタイプ
- ブロック・クラスおよびファイルに関連するロック・アクティビティ
- 分散ロック・マネージャが送受信するメッセージ

これらの統計情報は、次のビューで表示されます。

- V$CACHE
- V$LOCK_ACTIVITY
- V$CLASS_PING
- V$FILE_PING
- V$DLM_MISC

これらのビューの他に、この章で説明している手順を使用して、CPU 使用方法、ディスク I/O およびバックグラウンド・プロセス（LCK、LMD、BSP、DBW など）の CPU 使用率を示すオペレーティング・システムの統計情報を分析する必要があります。

チューニングに対する統計情報の記録

あるイベントが発生する割合についての統計情報を記録することをお薦めします。また、Oracle Parallel Server 環境内の特定のトランザクションに関する情報を記録する必要もあります。そのためには、1 秒ごとおよびトランザクションごとの統計件数を計算する、UTLBSTAT や UTLESTAT などのパフォーマンス指向のユーティリティを使用できます。また、アプリケーションが送信する、文の実行数およびビジネス・トランザクションの数を測定することもできます。

たとえば、保険のアプリケーションでは、トランザクションが保険の見積もりとして定義される場合があります。このトランザクションは、複数の DML 操作および問合せで構成される場合があります。一定の時間内にシステムが処理する見積りの数がわかっていれば、その値をその時間内に完了した見積もりの数で割ります。これを一定時間行い、パフォーマンスを評価します。

これによって、トランザクションごとに使用されるリソースおよびアプリケーションの作業負荷の観点から、アプリケーション・プロファイルが示されます。トランザクションごとの件数は、作業負荷のパターンの変更を検出する場合に有効です。また、割合は作業負荷の程度を示します。

Oracle Parallel Server の作業負荷のパフォーマンスおよび効率

Oracle Parallel Server では、アプリケーションのパフォーマンスおよびスケーラビリティは、ノード間の同期化の割合およびコストによって決定されます。トランザクションが CPU リソースをどれだけ効率的に使用しているかを識別することによって、コストを測定できます。トランザクションは、次の目的のために CPU を使用します。

- メッセージの送受信
- グローバル・キャッシュの一貫性を保証するために必要なロックおよびリソースの保持
- ping 処理に対応付けられたキャッシュ・バッファの無効性による、追加の I/O 要求の処理

ロックのオープン要求と変換要求および I/O イベントの待機によって追加のコストが発生します。データベース・リソースでロックをオープンまたは変換中に、ローカル・トランザクションとリモート・トランザクション間のリソースが競合すると、同期化のコストが増加します。

一般的な例として、次の計算式を使用して、トランザクションまたは要求コストを概算します。

$$CPUapps + CPUsyncio + CPUipc + CPUlocks + WAITsyncio + WAITipc + WAITlocks$$

次の値も計算できます。

- IPC レイヤーで使用された CPU 時間
- ロック要求の処理に必要な CPU 時間
- SQL 文の解析、行のフェッチ、ソートなど、アプリケーション処理に使用される CPU
- ノード間同期化によって発生する読込みおよび書込み I/O の処理に使用される CPU 時間

> **注意：** これらのイベントに対応付けられたディレイは、「waits」といわれます。

これらのイベントに関する統計情報は、V$SYSTEM_EVENT などの V$ 表に表示されます。各トランザクションの応答時間は、I/O およびロックの要求の数、指示を処理するのに必要な時間、および各要求のディレイまたは待機時間によって異なります。

あるリソースでの競合によって、測定可能な各コンポーネントのコストが増加します。たとえば、次のことが原因でディスク・サービスまたはキュー時間が増加します。

- 特定のディスクに対する頻繁な I/O 要求
- ローカルおよびリモート・トランザクションによる同じデータまたは索引ブロックの競合

これは、ビジー・バッファの待機の原因になります。これによって、各トランザクションのコストが増加し、オペレーティング・システムのオーバーヘッドに追加されます。

第 5 章で説明しているように、同期化の割合またはコストを最小限にすることによって、よりスケーラブルで、ユーザー・コミュニティのサービス・レベルの要件を満たすために十分な Oracle Parallel Server アプリケーションを作成できます。つまり、ノード間同期化およびノード間通信の要件を最少限に抑えることが必要です。たとえば、他のトランザクションが干渉しないトランザクションのグループによって処理できるアプリケーションの一部を分離するパーティション化によって、グローバル同期化のコストおよび割合を削減できます。

第 7 章で説明しているように、ロック・メカニズムおよびロック割当ての選択も、グローバル・キャッシュの一貫性の割合およびコストに影響します。キャッシュ・フュージョンなどの Oracle Parallel Server の最新の最適化では、共有または排他的バッファ・アクセスに対する要求のディレイを減らすことによって、インスタンス間の操作のコストが削減されます。

同期化コストの決定

この項では、追加の CPU 時間、I/O やグローバル・ロック・プロセス、および競合による、インスタンス間での同期化および一貫性によって発生するコストの決定方法について説明します。これを行うには、Oracle 統計情報の次のグループを調べます。

- 必要な CPU サービス時間の計算
- I/O の同期化コストの見積り
- グローバル・キャッシュの一貫性および競合の測定
- グローバルおよびローカルな作業の割合の計測
- ロック競合によるグローバル・キャッシュの同期化コストの計算

必要な CPU サービス時間の計算

トランザクションごとに必要な CPU サービス時間を導出するには、V$SYSSTAT に表示されているセッションが使用している CPU を、ユーザーのコミット数またはビジネス・トランザクションの数で割ります。これは、ユーザー・プロセスがユーザー・モードまたはカーネル・モードで実行するのに必要な時間であることに注意してください。これには、トランザクションのかわりにオペレーティング・システムのカーネルが使用する時間は含まれていません。

この測定は、排他モードで実行している単一インスタンス環境でのアプリケーションの動作を、Oracle Parallel Server 環境で実行しているアプリケーションと比較する場合に有効です。また、異なる作業負荷およびアプリケーション設計変更の影響を比較する場合にも有効です。

I/O の同期化コストの見積り

次の要求の件数は、V$SYSSTAT ビューを参照してください。

- DBWR cross-instance writes
- physical writes
- physical reads

次のアクションの待機時間および平均待機時間は、V$SYSTEM_EVENT ビューを参照してください。

- db file parallel write
- db file sequential read
- db file scattered read

別のインスタンスからの要求によって、ディスクに書き込まれるデータ・ブロックの再読込みによって発生する読込みの待機時間を見積もるには、統計情報（たとえば、db file swquential reads の待機時間）に、次の式で計算される、前回のキャッシュのフラッシュによって発生した読込み I/O の割合を掛けます。

$$\frac{\textit{lock buffers for read}}{\textit{physical reads}}$$

lock buffers for read は N から S へのロック変換の値であり、V$LOCK_ACTIVITY ビューから導出されます。また、physical reads は V$SYSSTAT ビューから導出されます。

同様に、ping によって発生したデータベース・ファイルのパラレル書込みの待機時間の割合は、V$SYSTEM_EVENTS に表示される db file parallel write time に、次の式を掛けて見積もることができます。

$$\frac{\textit{DBWR cross-instance writes}}{\textit{physical writes}}$$

グローバル・キャッシュの一貫性および競合の測定

表 11-1 に、グローバル・キャッシュの一貫性に関するビューおよびそれに含まれる統計情報の種類を示します。

表 11-1 グローバル・キャッシュの一貫性と競合の統計情報およびビュー

ビュー	統計情報
右に示すアクションの要求数をカウントするには、V$SYSSTAT を参照してください。	global cache gets（オープンされている新しいロックの件数） global cache converts（既存のロックに対する変換の件数） global cache cr blocks received（BSP から受信された一貫読込みバッファの件数） **注意 :** V$LOCK_ACTIVITY の変換タイプ固有の行も参照してください。
右に示すアクションの待機時間は、V$SYSSTAT を参照してください。	global cache get time（待機を含む合計処理時間） global cache convert time（待機を含む合計処理時間） global cache cr block receive time（待機を含む）
右に示すイベントの待機時間は、V$SYSTEM_EVENT を参照してください。	global cache lock null to X global cache lock null to S global cache lock S to X global cache lock open X global cache lock open S global cache cr request

高い競合または過剰なディレイの標識は、次の統計情報およびビューを参照してください。

- V$SYSSTAT にある global cache cr timeouts
- V$SYSSTAT にある global cache covert timeouts

- V$SYSTEM_EVENT にある global cache lock busy
- V$SYSTEM_EVENT にある buffer busy due to global cache

前述したように、トランザクションごとおよび時間単位ごとのアプリケーション・プロファイルのメンテナンスは有効です。これによって、2 つの個別の作業負荷を比較したり、作業負荷の変更を検出できます。割合は、容量の決定およびスループット問題の識別にも有効です。パフォーマンス・モニター・スクリプトに、次の統計情報の割合を取り込むことをお薦めします。

- トランザクションごとの lock gets（たとえば、トランザクションごとの global cache gets）
- トランザクションごとの lock converts（たとえば、トランザクションごとの global cache converts）
- トランザクションごとの cr request（たとえば、トランザクションごとの global cache cr blocks received）
- トランザクションごとの lock convert waits（V$SYSTEM_EVENT の TIME_WAITED 列に示される値）
- トランザクションごとの global cache lock null to x
- トランザクションごとの global cache lock null to s
- トランザクションごとの global cache lock s to x
- トランザクションごとの lock open waits（たとえば、V$SYSTEM_EVENT の TIME_WAITED 列）
- トランザクションごとの global cache lock open x
- トランザクションごとの global cache lock open
- トランザクションごとの lock busy waits（たとえば、V$SYSTEM_EVENT の TIME_WAITED 列）
- トランザクションごとの global cache lock busy

合計件数または待機時間を計測間隔で割って、1 秒ごとまたは 1 分ごとに同じ統計情報を計算します。

注意： 定期的に統計情報を記録するには、TIMED_STATISTICS パラメータを TRUE に設定してください。Oracle は、これらの統計情報を数百分の 1 秒ごとに記録します。

グローバルおよびローカルな作業の割合の計測

グローバル作業に対してアクセスされたバッファの割合、またはインスタンス間同期化によって発生したI/Oの割合は、アプリケーション・プロセスがどの程度データを効率的に共有しているかを測定する場合に重要です。また、データベースがスケーラビリティを最適化するように設計されているかどうかもわかります。

次の式で、ローカルな操作に対するバッファ・アクセス（ロック変換の原因とならないデータベース・バッファの読込みおよび変更）の割合を計算します。

$$\frac{((\textit{consistent gets} + \textit{db block gets}) - (\textit{global cache gets} + \textit{global cache converts}) * 100)}{(\textit{consistent gets} + \textit{db block gets})}$$

同様に、次の式で、ローカルな操作に対する読込みおよび書込みI/Oの割合を計算します。

$$\frac{(\textit{physical writes} - (\textit{DBWR cross-instance writes})) * 100}{\textit{physical writes}}$$

この計算は、ローカルな作業に対するDBWR読込みの時間の割合を示します。

次のように計算することもできます。

$$\frac{(\textit{physical reads} - (\textit{lock buffers for read})) * 100}{\textit{physical reads}}$$

この計算は、ローカルな作業のみに対するユーザー・プロセスによるパーセント読込みの数を示します。強制読込みは含みません。

前述の式では、V$SYSSTATのphysicl read統計情報が、V$LOCK_ACTIVITYのlock buffers for read値と組み合せて使用されています。ローカルな書込みの割合は、V$SYSSTATの対応する値に基づいて計算できます。

ローカルおよびグローバルな作業の比率（パーティション化の程度）を決定する他に、これらの割合を使用して、作業負荷のパターンの変更を検出できます。

また、これらの割合は、データ・ブロック・アクセスがグローバルまたはローカルである可能性も示します。そのため、この情報を、スケーラビリティを計算する場合の概算として使用できます。

これらの割合の他に、使用不可のバッファまたはロックによるディレイは、次の式で簡単に計算できます。

$$\frac{100 * (\textit{buffer busy due to global cache})}{\textit{buffer busy + buffer busy due to global cache}}$$

次の式でも計算できます。

$$\frac{100 * (\textit{buffer busy due to global cache})}{\textit{consistent gets+db block gets}}$$

ロックのオープンおよび変換に対して、ビジー・ロック（取得中または解放されているロック）によって発生した待機の割合は、ロックのオープンまたは変換でのディレイ、またはバッファの小さい集合における高い同時実行性を示します。

$$\frac{100 * (\textit{time waited for global cache lock busy})}{\textit{sum(time waited for global cache opens and global cache converts)}}$$

問題の領域（高い割合の global busy waits、convert and consistent read timeouts、高い割合の DBWR cross-instance writes など）がわかれば、次のビューを参照することによって問題の詳細がわかります。

- V$FILE_PING
- V$CACHE
- V$CLASS_PING

これらのビューの統計情報は、両方のインスタンスが共有しているファイルおよびブロックを識別する場合に有効です。これらの共有ファイルが、インスタンス間同期化およびグローバル・キャッシュの一貫性処理における多くのコストの原因である場合があります。

ロック競合によるグローバル・キャッシュの同期化コストの計算

インスタンス間同期化のコストが増加することによって、スループットが低下し、トランザクションの応答時間が増大します。この項では、次に示すコスト増大の原因について説明します。

- 同じデータ・ブロックの競合
- セグメント・ヘッダーの競合および空きリスト・ブロック
- データベース・ブロック以外のリソースの競合
- ロックの不足

同じデータ・ブロックの競合

同じデータ・ブロックの競合は、複数のインスタンスによって一般的にアクセスされる行が、ブロック内の限られた範囲に分散されている場合に発生します。これが発生する確率は、アプリケーションの動作による、表内でのデータのアクセス分散によって異なります。競合の可能性は、ブロックのサイズにも依存します。たとえば、4KB のブロックより 8KB のブロックの方が、多くの行を持つことができます。特定の表に対して定義された PCTFREE は、競合のレベルに影響します。実際、データベースのブロック・サイズおよび PCTFREE は、Oracle Parallel Server の設計方法の一部として使用することができます。目標は、ブロックごとの行数（アクセス競合の可能性）を削減することです。

グローバルで頻繁にアクセスされるブロックの標識には、次のものが含まれます。

- 高い割合の buffer busy waits または buffer busy due to global cache waits
- 頻繁な convert timeouts または consistent read timeouts

トランザクションごとの global cache lock waits の割合が高い場合は、両方のノードからどのファイルおよびブロックが頻繁にアクセスされているかを判断してください。次に、競合ファイルを識別する 1 つの方法を説明します。

V$CACHE、V$PING および V$BH を使用した競合オブジェクトの識別

動的パフォーマンス・ビュー V$CACHE、V$PING および V$BH には、2 つの列 FORCED_READS および FORCED_WRITES があり、これらの列によって、両方のノードが使用しているオブジェクトおよびブロックを識別できます。

FORCED_WRITE 列には、あるブロックがローカル・バッファ・キャッシュから ping out された回数（別のインスタンスによって現在のバージョンが要求されたためにディスクに書き込まれた回数）が含まれます。

それに対応して、FORCED_READ 列は、以前に別のインスタンスからの排他要求によって無効にされたブロックを、ディスクから再読込みする必要がある頻度を追跡します。また、V$FILE_PING を使用して、最も多く ping されたファイルを識別することもできます。

V$FILE_PING を使用した過度の ping が発生するファイルの識別

V$FILE_PING ビューを使用して、ping によって最も頻繁に書き込まれたブロックを含むファイルを識別します。最も高い ping 率を持つファイルを識別したら、これらのファイルに対するロック方針（固定または解放可能）を再検討してください。また、これらのファイルを物理的に分散させて、I/O ディレイを削減すべきかどうかについても検討する必要があります。最適な方法は、ファイルに含まれるオブジェクトでのローカル作業の割合を増加させること（ping を全体的に回避すること）です。

注意： 頻繁に ping されるブロックは、クラスタ内にあるすべてのインスタンスのローカル・バッファ・キャッシュで参照できます。

セグメント・ヘッダーの競合および空きリスト・ブロック

セグメント・ヘッダーの競合は、表または索引のヘッダーが、多くのトランザクションによって同時に読込みおよび更新される必要がある場合に発生します。通常、これは、挿入または更新するデータのサイズを保持するために十分な空き領域を持つブロックをトランザクションが検索しているときに発生します。また、新しいエクステントまたは追加の空きブロックが表に追加された場合は、セグメント・ヘッダーが更新されます。

非常に大量のデータを複数のノードから挿入する新しいアプリケーションは、パフォーマンスの重大なボトルネックになる可能性があります。これは、Oracle が、空きリストを含むセグメント・ヘッダー・ブロックを別のインスタンス・キャッシュにコピーする必要があるためです。これは、競合の原因の 1 つになります。

問題の表および索引に対して空きリスト・グループを作成することによって、この状況を大幅に改善できます。空きリスト・グループを使用するメリットは、アクセスをインスタンスに基づいたセグメントの空きリストに分割できることです。これによって、INSERT および DELETE の割合が高い場合のインスタンス間の競合が削減されます。

データベース・ブロック以外のリソースの競合

データベース・ブロック以外のリソースの競合は、頻繁には発生しません。ただし、競合が発生した場合、パフォーマンスが低下します。通常、このような問題が発生する可能性があるレイヤーは次の 2 つです。

- 行キャッシュまたはデータ・ディクショナリ・キャッシュ
- ライブラリ・キャッシュ

キャッシュされていない Oracle 順序を使用したり、表または表領域に対して不完全に設定された領域パラメータを使用すると、データ・ディクショナリが頻繁に変更される場合があります。データ・ディクショナリの行は、別々のキャッシュに格納され、データ・バッファ・キャッシュの形式とは異なるバッファ形式を使用します。これらのオブジェクトが両方のノードから頻繁に読込みまたは更新される場合、Oracle は、これらのオブジェクトを無効にし、ディスクに書き込む必要があります。これらのオブジェクトは、他のロックが保持されている間は再帰的な内部トランザクションで使用されます。このような内部データ・ディクショナリ・トランザクションが複雑であることが原因で、この処理によって重大なディレイが発生する場合があります。

ディレイが発生すると、V$SYSTEM_EVENT にある row cache lock 待機件数および time waited 統計情報の値が増加します。これらは、最も長いイベント待機のうちの 2 つになります。time waited for row cache locks の割合は、合計待機時間の 5% を超えないようにしてください。ディレイの原因となる行キャッシュのオブジェクトを判断するには、V$ROWCACHE ビューを問い合せます。この問合せに対する Oracle の応答は、行キャッシュのオブジェクト型名（DC_SEQUENCES、DC_USED EXTENTS など）、およびこれらのオブジェクトに対する DLM 要求と DLM 競合です。競合が要求の 10 〜 15% を超え、行キャッシュ・ロック待機時間が長すぎる場合は、順序番号のキャッシュ、表領域の非断片化または領域パラメータのチューニングを行って競合を防止します。

最も頻繁に問題が発生するのは、Oracle が CACHE オプションなしで順序を作成した場合です。この場合、DC_SEQUENCES の値に高い割合の DLM 競合が示されます。断片化された表領域または不十分なエクステント・サイズによる頻繁な領域管理操作によって、データ・ディクショナリ・キャッシュの DC_USED_EXTENTS および DC_FREE_EXTENTS オブジェクトに対して ping が発生する可能性があります。

データ・ディクショナリが頻繁に変更される場合は、データ・ディクショナリ表（通常、ファイル番号 1）の多数のデータ・ブロックが、V$CACHE または V$PING を問い合せるときに、各インスタンスのバッファ・キャッシュ内で見つけられます。

Oracle Parallel Server では、ライブラリ・キャッシュのパフォーマンスの問題はほとんどありません。通常、これらの問題は、library cache pin に対する高い待機時間から判断でき、頻繁なカーソルの再解析、またはプロシージャおよびパッケージのロードが原因で発生します。この問題の詳細は、V$LIBRARYCACHE ビューの DLM 列を問い合せてください。ライブラリ・キャッシュ・オブジェクトのインスタンス間の無効化は、ほとんど行われません。ただし、2 つのインスタンスで実行されているカーソルが参照しているオブジェクトを削除した場合、このような無効化が行われる場合があります。

ロックの不足

データベースおよびバッファ・キャッシュが大きい場合、およびメモリー上の制約がある場合、競合ロックまたはリソースが不足することがあります。パラメータ GC_RELEASABLE_LOCKS を、デフォルトより低い値に設定できます。この値は、DB_BLOCK_BUFFERS によって決まるバッファ・キャッシュのサイズと同じにする必要があります。これによって、ロック用のメモリーを節約し、リソースを節約できます。

ただし、多くのリソースがグローバル・キャッシュの一貫性を保持するために使用され、ほとんどの空きリストが使用されている場合は、さらに頻繁にロックを解放および再オープンする必要があります。これは、global cache lock free list waits および global cache gets または global cache lock open events の件数の増加によって識別できます。ロックの解放を待機したり、または高い割合でロックがオープンされている場合、インスタンスの同期化によって発生するコストが増加し、その結果、トランザクションの応答時間および CPU の使用量が増加します。

Oracle Parallel Server ベースのアプリケーションにおける問題の解決

この項では、Oracle Parallel Server ベースのアプリケーションにおける問題の識別方法および解決方法について説明します。この項の内容は、次のとおりです。

問合せのチューニングのヒント

問合せ集中型のアプリケーションでは、I/O 要求ごとのデータ量を最大にするチューニング・テクニックによって効果が得られます。このチューニングを行う前に、このチューニング・テクニックを実装する前後のパフォーマンスを監視し、このチューニングの効果を測定および判断します。

大きいブロック・サイズ

ブロック・サイズを大きくして、各操作によって取り出される行数を増加させます。また、ブロック・サイズを大きくすると、アプリケーションの索引ツリーの階層が浅くなります。

データベースが主に問合せ処理に使用されている場合は、ブロック・サイズを 8KB 以上にする必要があります。

DB_FILE_MULTIBLOCK_READ_COUNT の値の増加

DB_FILE_MULTIBLOCK_READ_COUNT の値を、できるだけ大きい値に設定する必要もあります。これによって、表をスキャンするのに必要な読込み数が削減され、フル・テーブル・スキャンがより高速になります。システム I/O は、読み込まれたブロック数をブロック・サイズに掛けた値に制限されていることに注意してください。

オペレーティング・システムのストライプ化を使用している場合、DB_BLOCK_SIZE の 2 倍の値を DB_FILE_MULTIBLOCK_READ_COUNT に掛けた値にストライプ・サイズを設定します。システムが索引ストライプと表データ・ストライプを区別する場合は、DB_BLOCK_SIZE に 2 を掛けたストライプ・サイズを索引に使用します。

また、次のことに注意してください。

- データおよび索引に対して、読込み専用表領域を使用します。
- 一時セグメントを保持している表領域を TEMPORARY 型として定義し、頻繁にフル・スキャンされる表を含むデータ・ファイル上の PCM ロックに対して、大きいブロッキング要素を使用します。

アプリケーションのチューニングのヒント

一般的に、トランザクション・ベースのアプリケーションは、その他のタイプのアプリケーションよりも多くのデータをディスクに書き込みます。トランザクション・ベースのアプリケーションを最適化するには、いくつかの方法がありますが、これらの方法を、すべてのタイプのシステムで使用できるわけではありません。これらの方法でチューニングを開始した後、アプリケーションのパフォーマンスを監視して、この方法が受け入れられることを確認してください。

データベース・ライター・プロセス（DBWn）の機能を改善して大量のデータをより高速に書き込ませるには、非同期化 I/O を使用します。Oracle8*i* は、複数の DBWn プロセスを使用して、パフォーマンスを向上させます。粒度が低いロックを使用して、false ping を回避することもできます。

インスタンスごとにパーティション化されたユーザーがあり、複数のリストを保持するために十分な領域がある場合は、空きリスト・グループを使用します。これによって、アプリケーションの動的なデータのパーティション化を回避できます。また、表を手動で値ごとにパーティション化することもできます。アクセスがランダムな場合は、より小さいブロック・サイズを使用することを考慮します。次の点に注意してください。

- 関連する索引およびシーケンス・ジェネレータでの競合に注意します。
- パーティション化およびフェイルオーバーのために、マルチティア・アーキテクチャを使用します。

パフォーマンス問題の診断

アプリケーションが正常に実行しない場合は、そのアプリケーションの各コンポーネントを分析して、問題の原因となるコンポーネントを識別します。そのためには、次の項で説明するように、オペレーティング・システムおよび DLM 統計情報をチェックして、競合または過剰な CPU 使用率を識別します。決まった手順で測定可能な過剰なロック変換によって、DLM コンポーネントによる過剰な読込み / 書込みアクティビティまたは高い CPU 要件を判断できます。

> **参照：** Oracle Parallel Server のパフォーマンスのチューニングの詳細は、第 11 章を参照してください。

競合および CPU 使用率を監視するための DLM 統計情報

アプリケーションのパフォーマンスが最適でない場合は、次に示すように、統計情報を調べることを検討します。

- UTLBSTAT、UTLESTAT を実行し、V$SQL_AREA ビューを問い合せて、標準のチューニング・テクニックを使用します。このビューから統計情報を調べて、共有プールおよびバッファ・キャッシュでのヒット率を分析します。
- CATPARR.SQL を実行するときに作成された、動的パフォーマンス表の統計情報を調べます。
- V$LOCK_ACTIVITY 表を使用して、ロックの割合を監視します。
- V$BH 表を使用して、どのブロックが ping されているかを識別します。この表では、各ブロックの PCM ロックが排他から NULL へダウングレードされた回数の合計が示されます。
- V$PING ビューには、排他から NULL への件数が 0（ゼロ）ではない V$CACHE 表の行が示されます。

注意： オブジェクトが新しく取得したブロックは、CATPARR.SQL を再実行するまで、これらの表には表示されません。

Parallel Server 環境に固有の競合問題

Oracle Parallel Server 環境で回避できる重大な競合問題があります。これらの問題は、複数のインスタンスが主キー値に対してシーケンス・ジェネレータを共有する場合に、索引ブロックへの挿入によって発生します。これらの問題は、次のものを使用することによって最小化できます。

- 順序番号の乗数の使用
- Oracle 順序の使用

順序番号の乗数の使用

インスタンスが新しいエントリを同じ索引に挿入しないように、SEQUENCE_NUMBER * INSTANCE_NUMBER * 1,000,000,000 などの乗数を使用する必要がある場合があります。

Oracle 順序の使用

CACHE 句を使用しないで順序を作成すると、多くのオーバーヘッドが発生する場合があります。両方のインスタンスが同じ順序を使用する場合は、同期化のオーバーヘッドの原因になる場合もあります。

12

Oracle Parallel Server およびインスタンス間パフォーマンスのチューニング

この章では、Oracle Parallel Server およびキャッシュ・フュージョン関連の統計情報について説明し、これらの統計情報を使用してパフォーマンスを監視およびチューニングする方法を説明します。また、この章では、キャッシュ・フュージョンによる Oracle Parallel Server の読込み / 書込み競合の解消についても簡単に説明します。さらに、ほとんどのシステムおよびアプリケーションに適用される一般的な用語を使用して、キャッシュ・フュージョンの利点を説明します。

内容は次のとおりです。

- キャッシュ・フュージョンによる一貫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スケーラビリティの改善
- Oracle Parallel Server 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
- パフォーマンスへの影響
- キャッシュ・フュージョンおよびインスタンス間パフォーマンスの監視

> **参照：** キャッシュ・フュージョン処理の概要は、『Oracle8*i* Parallel Server 概要』を参照してください。

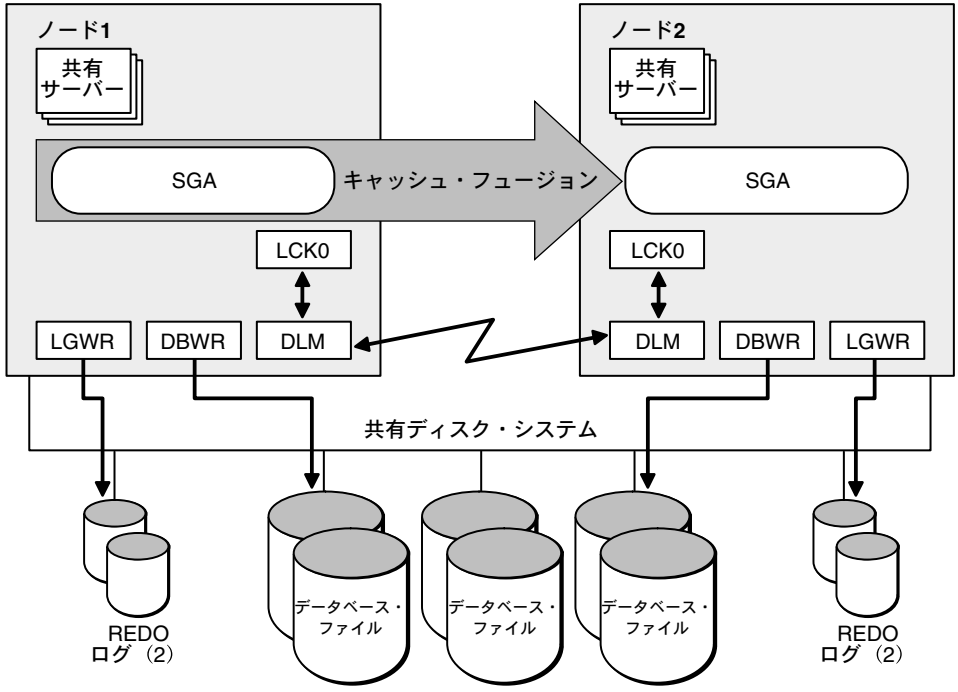
キャッシュ・フュージョンによる一貫読込みブロックの作成方法

1 つの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・キャッシュ内にある場合、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・アクセスを使用して、読込み / 書込み競合を解消します。要求側のインスタンスは、ブロックの一貫読込みの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ブロック・サーバー・プロセス（BSP）は、要求されたブロックの一貫読込みイメージを、高速インターコネクト経由で、保持側のインスタンスのバッファ・キャッシュから要求側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 12-1 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・ブロックを直接送信できます。そのため、パラレル・キャッシュ管理における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・テクノロジを活用して、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。これによって、ノード間メッセージに対するオペレーティング・システムのコンテキスト切替えが削減され、CPU 使用率が低下します。Oracle は、従来のディスク・ベースのパラレル・キャッシュ管理（PCM）を使用して、書込み / 書込み競合を管理します。

図 12-1 キャッシュ・フュージョンによるインターコネクト経由のキャッシュ間ブロック転送



注意： キャッシュ・フュージョンは常に使用可能です。

書込み / 書込み競合の解消を改善するためのデータのパーティション化

キャッシュ・フュージョンによって、多くの読込み / 書込み競合が発生するアプリケーションのスケーラビリティが改善されますが、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。書込み / 書込み同時実行性の高い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、書込み / 書込み競合の可能性を減少させる必要があります。

キャッシュ・フュージョンによるスケーラビリティの改善

キャッシュ・フュージョンは、次のことによって、アプリケーション・トランザクションのスループットおよびスケーラビリティを改善します。

- コンテキスト切替えを削減します。これによって、読込み / 書込みキャッシュ一貫性による競合中の CPU 使用率を削減します。
- ユーザー・モードの IPC プラットフォームでは、CPU 使用率をさらに削減します。
- ブロック・ピングのための I/O を削減し、X-to-S ロック変換を削減します。
- 高速インターコネクトを経由して一貫読込みブロックを送信します。

ディスク・ベースの PCM で読込み / 書込み競合の発生率が高いアプリケーションが、キャッシュ・フュージョンの最も大きい効果を得られます。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、より効率的に拡張できます。キャッシュ・フュージョンは、OLTP およびレポート作成機能が別々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 機能によって変更され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経由してそのバージョンのデータ・ブロックを受信します。これによって、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、およびそれに伴う X-to-S ロック変換に対するディスク I/O の減少によるものです。

さらに、別のインスタンスから同じブロックに対する読込み要求を受信する前に、キャッシュ・データ・ブロックを変換していたインスタンスは、後続の変更に対して、ブロックへの排他的アクセスを再要求する必要はありません。これは、そのブロックが読込み側のインスタンスに送信された後も、そのインスタンスが排他ロックおよびバッファを保持しているためです。

注意： キャッシュ・フュージョンによって、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。どの程度向上するかは、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体のシステム構成によって異なります。

高速インターコネクトを経由した一貫読込みブロックの転送

キャッシュ・フュージョンは高速 IPC を使用します。そのため、クラスタ・インターコネクトを経由した待ち時間の短い通信を実現する最新テクノロジによって、Oracle Parallel Server のパフォーマンスが向上します。さらに、仮想インタフェース・アーキテクチャ（VIA）やユーザー・モードの IPC などのより効率的なプロトコルによって、さらにパフォーマンスの向上が期待できます。

キャッシュ・フュージョンは、ユーザー・モードの IPC（「メモリー・マップド IPC」ともいわれます）を活用して、UNIX および NT ベースのプラットフォームでの CPU 使用率を削減します。ハードウェアが適切に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切替えは、キャッシュ・フュージョンのみによる標準的な削減以上に最小化されます。また、高コストのデータ・コピーおよびシステム・コールも排除されます。

ユーザー・モードの IPC がハードウェアのサポートによって効果的に実装されている場合、ユーザー・プロセスがオペレーティング・システムのカーネルを使用しないで通信できるため、CPU 使用率が削減されます。つまり、ユーザー実行モードからカーネル実行モードへ切り替える必要はありません。

ブロック・ピングのための I/O の削減および 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・キャッシュに一貫読込みブロックを直接送信することによって、データ・ブロックおよびロールバック・セグメント・ブロックに対する高コストのロック・オペレーションおよびディスク I/O を削減します。これによって、読込み / 書込み競合を解消するために必要な待ち時間を最大 90 パーセント短縮できます。

キャッシュ・フュージョンは、ディスク I/O を使用しないで、ディスク・ベースの PCM に必要な処理量の約 10 分の 1 の処理量で、読込み / 書込み同時実行性を解決します。キャッシュ・フュージョンによって発生するオーバーヘッドは、一貫読込み要求を処理し、要求されたブロックの一貫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側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。1000 分の 1 秒もかからないプラットフォームもあります。

Oracle Parallel Server 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及ぼす主な要素は、インターコネクトおよび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読込みブロック要求を処理する速度が決定されます。

インターコネクト処理への影響

一度インターコネクトが動作可能になると、インターコネクトのパフォーマンスを大きく変更す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。

> **参照：** 詳細は、ベンダー固有のインターコネクトのドキュメントを参照してください。

サポートされているインターコネクト・ソフトウェア

Oracle Parallel Server およびキャッシュ・フュージョンをサポートしてい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 / インターコネクト・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

Oracle Parallel Server では、これらのプロトコルをサポートしているすべてのインターコネクト製品を使用できます。インターコネクト製品は、Oracle Parallel Server 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があります。

パフォーマンスへの影響

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およびスループットは、アプリケーションによって異なります。システムが処理するトランザクションの種類および組合せによっても、パフォーマンスがさらに影響を受けます。

キャッシュ・フュージョンによるパフォーマンス向上の程度は、作業負荷によって異なります。また、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によっても、パフォーマンスが影響を受けます。

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ない可能性があります。ただし、アプリケーションで一貫読込み競合の結果として非常に多くのロック変換および大量のディスク I/O が発生した場合は、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。

Oracle8.1 と Oracle8.0 のロックおよび I/O 統計情報を比較すると、排他から共有へのロック要求および物理書込み I/O が大幅に減少していることがわかります。次の項「キャッシュ・フュージョンおよびインスタンス間パフォーマンスの監視」では、キャッシュ・フュージョンのパフォーマンスの評価方法を詳しく説明します。

参照： ロック・タイプの詳細は、第 7 章を参照してください。

キャッシュ・フュージョンおよびインスタンス間パフォーマンスの監視

この項では、Oracle Parallel Server およびキャッシュ・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。この項の内容は、次のとおりです。

- キャッシュ・フュージョンおよび Oracle Parallel Server のパフォーマンス監視の目標
- Oracle Parallel Server およびキャッシュ・フュージョンを監視するための統計情報
- V$SYSTEM_EVENTS ビューを使用したパフォーマンス問題の識別

キャッシュ・フュージョンおよび Oracle Parallel Server のパフォーマンス監視の目標

キャッシュ・フュージョンおよび Oracle Parallel Server のパフォーマンスを監視する主な目標は、グローバル処理のコストを判断し、一貫性の保持およびインスタンスの同期化に必要なリソースを明確にすることです。これは、次の項で説明するように、いくつかのビューからパフォーマンス統計情報を分析することによって行います。これらの監視を継続して行い、処理の傾向を観察して、処理を最適なレベルで保持します。

多くの統計情報は、キャッシュ・レイヤー、トランザクション・レイヤー、I/O レイヤーなど、データベース・カーネルの様々なコンポーネントが行う作業を計測するために使用できます。さらに、定期的な統計情報によって、ある要求を処理する時間または特定のイベントを待機する時間を正確に判断できます。

これらの統計情報ソースから、作業割合、待機時間および効率を取り出すことができます。

参照： 収集する統計情報およびその統計情報を使用してパフォーマンス割合を計算する方法の詳細は、第 11 章を参照してください。

Oracle Parallel Server およびキャッシュ・フュージョンを監視するための統計情報

Oracle は、バッファ・キャッシュおよび DLM 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、Oracle は、ロック要求およびロック待機に対する Oracle Parallel Server 統計情報も収集します。いくつかのビューを使用して、これらの統計情報を調べることができます。

システム・パフォーマンスの履歴を十分にメンテナンスすると、統計情報の変更が表す傾向を識別しやすくなります。これによって、応答時間の増大およびスループットの低下の原因を識別しやすくなります。また、作業負荷の変更およびピーク時の処理要件を識別する場合にも有効です。

この項の手順では、次の項目に従ってグループ化した統計情報を使用しています。

- グローバル・キャッシュおよびキャッシュ・フュージョンの統計情報の分析
- グローバル・ロック統計情報の分析
- DLM リソース、ロック、メッセージおよびメモリー・リソース統計情報の分析
- DLM メッセージの統計情報
- Oracle Parallel Server の I/O 統計情報の分析
- ラッチ、Oracle Parallel Server および DLM の統計情報の分析

第 11 章で説明するとおり、1 秒ごとおよびトランザクションごとの V$SYSSTAT ビューおよび V$SYSTEM_EVENT ビューからの統計情報をメンテナンスして、作業負荷の一般的なプロファイルを取得することを検討してください。これらのビューから、次のものを観察します。

- トランザクションごとの要求または件数（たとえば、トランザクションごとの physical reads やトランザクションごとの logical reads）
- トランザクションごとの待機時間または経過時間（たとえば、トランザクションごとの読込み I/O 待機時間やトランザクションごとの lock convert time）
- 1 秒ごとの要求または件数
- 要求ごとの平均時間（たとえば、別インスタンスからの一貫読込みバッファ受取り平均時間）

これらの統計情報をメンテナンスすることによって、ある種類の操作によるトランザクションの応答時間に対するコストの増加の影響を正確に見積もることができます。また、作業割合または平均ディレイが大きく増加したことによっても、容量の問題を識別できます。

この項の手順で説明しているほとんどのビューに対する統計情報を収集するには、TIMED_STATISTICS パラメータを TRUE に設定します。

この章で説明しているビューの定期的な統計情報は、100 分の 1 秒単位で表示されます。

注意： CATPARR.SQL を実行して、統計情報をソートおよび参照するための Oracle Parallel Server 関連のビューおよび表を作成する必要もあります。これについては、次の項で説明します。

CATPARR.SQL による Oracle Parallel Server 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、Parallel Server のデータ・ディクショナリ・ビューが作成されます。このスクリプトを実行するには、SYSDBA 権限が必要です。

CATALOG.SQL は、次のビューの他に、標準 V$ 動的ビューを作成します。

- GV$CACHE
- GV$PING
- GV$CLASS_PING
- GV$FILE_PING
- GV$ROWCACHE
- GV$LIBRARYCACHE

エクステントを追加した後、CATPARR.SQL を再実行して、EXT_TO_OBJ 表に最新情報を格納できます。CATPARR.SQL を再実行せずにオブジェクトを削除すると、EXT_TO_OBJ に誤った情報が表示される場合があります。

> **参照：** 動的ビューの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

グローバル動的パフォーマンス・ビュー

Oracle データベースのチューニング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。Oracle Parallel Server では、グローバル動的パフォーマンス（GV$）・ビューに問い合せて、条件に合ったすべてのインスタンスから V$ ビュー情報を取り出せます。グローバル固定ビューは、V$ROLLNAME、V$CACHE_LOCK、V$LOCK_ACTIVITY および V$LOCKS_WITH_COLLISIONS 以外のすべての既存の動的パフォーマンス・ビューにあります。

グローバル・ビューには、ローカル・ビューのすべての列の他に、INST_ID 列（データ型 INTEGER）が含まれます。この列には、対応する V$ 情報が取得されたインスタンス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$ 情報を取り出すことができます。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 および 5 の V$ ビューから情報が取り出されます。

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・ビューの内容を取り出す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 制限に達している状態で、GV$ ビューを問い合せると、この目的のために追加の Parallel Server プロセスが起動されます。この追加プロセスは、GV$ 問合せ以外のパラレル操作には使用できません。

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 Parallel Server プロセスは起動しません。

インスタンスがすでに PARALLEL_MAX_SERVERS 制限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、最初の問合せ以外は正常に実行されません。ほとんど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、エラーになるか、または問合せコーディネータによって問合せが順次実行されます。

参照：

- 第 2 章を参照してください。
- GV$ ビューの制限事項、および関連するすべてのパラメータと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。

グローバル・キャッシュおよびキャッシュ・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・キャッシュ統計情報を収集します。これらの統計情報には、グローバル・リソースに対する問合せの件数および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・ロック要求は、要求側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 DLM 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システム・グローバル領域に割り当てられます。これらの構造を「ロック要素」といいます。

グローバル・キャッシュ統計情報を監視するには、次の手順で説明するように、V$SYSSTAT ビューを問い合せ、その出力を分析します。

グローバル・キャッシュ統計情報の監視手順

グローバル・キャッシュ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$SYSSTAT を問い合せます。

```
SELECT * FROM V$SYSSTAT WHERE NAME LIKE 'global cache%';
```

次のように出力されます。

```
NAME                                                                  VALUE
---------------------------------------------------------------------------
global cache cr blocks received                                        7372
global cache cr block receive time                                     2293
global cache cr blocks served                                          7882
global cache cr block serve time                                         60
global cache cr block send time                                         239
global cache cr block log flushes                                       119
global cache cr block log flush time                                    140
global cache cr timeouts                                                  2
global cache cr requests blocked                                          0
```

2. 次の式で、一貫ブロック要求の平均待ち時間（ラウンドトリップ時間）を計算します。

$$\frac{\textit{global cache cr block receive time}}{\textit{global cache cr blocks received}}$$

結果は、システム構成およびボリュームによって異なりますが、通常、約 15ms（ミリ秒）になり、一貫読込み要求のラウンドトリップ（要求側のインスタンスから保持側のインスタンスに送られて、要求側のインスタンスへ戻る）の平均待ち時間になります。CPU のアイドル時間が制限されていて、システムが、通常、長時間実行の問合せを処

理する場合は、待ち時間が長くなる場合があります。ただし、平均待ち時間を 1ms 未満にできます。

DB_MULTI_BLOCK_READ_COUNT パラメータ値が大きい場合、一貫読込みサーバーの要求待ち時間も影響されます。これは、このパラメータの設定によって、要求側のプロセスがブロックに対して複数の要求を発行する場合があるためです。そのため、要求側のプロセスの待ち時間が長くなる場合があります。

3. 特にレポート集中型のアプリケーションで受信要求の数が多い場合、または BSP での要求の送信元であるノードが複数ある場合、BSP のサービス時間が増加するため、ラウンドトリップ時間は増加します。ディレイの長さが BSP によって発生したかどうかを判断するには、要求ごとの平均サービス時間を計算します。

$$\frac{\textit{global cache cr block serve time}}{\textit{global cache cr blocks served}}$$

この手順に示されているように、要求ごとの平均 BSP サービス時間および要求ごとの合計ラウンドトリップ時間を追跡します。

サービス時間のどの部分が合計サービス時間と最も関連があるかを調べるには、次の 2 つの式を使用して、ログ・フラッシュの待機時間および完了した要求の送信にかかった時間を計算します。

$$\frac{\textit{global cache cr log flush time}}{\textit{global cache cr log flushes}}$$

$$\frac{\textit{global cache cr block send time}}{\textit{global cache cr blocks served}}$$

これらの平均を計算することによって、一貫読込みブロック要求のほとんどすべての処理手順を評価できます。要求ごとの合計ラウンドトリップ時間と BSP サービス時間のその他の違いは、LMD プロセスでの処理時間およびネットワーク IPC 時間です。

4. 次のいずれかの式を使用して、平均変換時間および平均取得時間を計算します。

$$\frac{\textit{global cache convert time}}{\textit{global cache converts}} \quad \textit{or} \quad \frac{\textit{global cache get time}}{\textit{global cache gets}}$$

変換時間が大きい場合は、過剰なグローバル競合が発生しています。global cache gets、global cache converts の値が大きく、平均変換時間および平均取得時間の急速な増加がある場合、過剰な競合が発生しています。別の原因としては、ロック・オペレーションの待ち時間が、システム全体の作業負荷またはシステム問題によって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がオープンされると、global cache gets が増加します。変換は、すでにオープンされているロックがあり、Oracle がそのロックを別のモードへ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および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が長すぎる場合は、システムでタイムアウトが発生している場合があります。

global cache convert time または global cache get time の値が大きい場合は、V$SYSTEM_EVENTS ビューの統計情報を参照して、TIME_WAITED 統計情報の値が大きい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 出力がこの統計情報で 0（ゼロ）以外の値を示している場合は、システムに負荷がかかっていないか、または過剰な競合がないかをチェックしてください。一般に、変換タイムアウトは発生しません。変換タイムアウトが発生するのは、重大なパフォーマンス問題がある場合です。

6. V$SYSSTAT 出力で global cache consistent-read timeouts の値を調べて、分析します。一貫読込み要求の完了をシステムが待機する時間が長すぎると、Oracle はこの統計情報を増やします。この統計情報が 0（ゼロ）以外の値を示している場合は、一貫読込み要求の開始から長時間が経過しており、タイムアウトが発生しています。

この状況が発生した場合、通常、一貫読込み要求が完了するまでの平均時間も増加しています。タイムアウトが発生しており、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、またはデータ・ブロックに対する過剰な競合が発生している可能性があります。これは、IPC またはネットワーク問題の標識としても使用される場合があります。

その他の有効なキャッシュ・フュージョンの統計情報

次に、その他のキャッシュ・フュージョン統計情報について説明します。これらの統計情報は、グローバル・キャッシュおよびキャッシュ・フュージョン操作を診断する場合に有効です。これらの統計情報を使用して、一貫ブロック要求のすべての操作を監視します。

global cache cr blocks received あるプロセスが、ローカル・キャッシュからでは満たされないデータ・ブロックの一貫読込みを要求する場合、プロセスは、要求を別のインスタンスへ送信します。一度要求が完了すると（バッファが受信される）、Oracle は要求の件数を減らします。

global cache cr block receive time この統計情報は、一貫読込み要求の完了にかかった合計時間（一貫読込みブロックに対するすべての要求のラウンドトリップ時間の累計）を記録します。

global cache cr timeouts この統計情報は、ディレイが長く、タイムアウトした一貫読込みブロックに対する要求を識別します。これは、システムのパフォーマンス問題、低速インターコネクト・ネットワークまたはネットワーク・パケットの削除が原因の場合があります。この統計情報の値は、常に 0（ゼロ）にする必要があります。

global cache cr blocks served これは、BSP が行った一貫読込みブロックに対する要求の数です。Oracle は、ブロックが送信されたときにこの統計情報を増やします。

global cache cr block serve time この統計情報は、BSP が受信するすべての要求を処理するために必要な累計時間を示します。各要求に対して、BSP が要求キューから要求を取り出した後、すぐに開始時間が記録されます。この間隔は、ブロックが送信された後に計算されます。

global cache cr block send time これは、BSP が一貫読込みブロックを送信するために必要な時間です。各要求に対して、タイミングはブロックが送信されたときに開始し、送信が完了したときに停止します。これはサービス時間の一部です。この統計情報は、送信するために必要な時間のみを測定することに注意してください。ブロックが要求側に到着するまでの経過時間は測定されません。

global cache cr block log flushes BSP が作成したデータ・ブロックのバージョンを含むバッファへの変更では、ログ・フラッシュが開始される必要があります。

BSP は、完了キューを管理することによって、非同期式で待機を処理します。一度 LGWR がログ・フラッシュ・キューにあるバッファへの変更のフラッシュを完了すると、BSP は、これを送信できます。これによって、キューは定期的にチェックされます。ログ・フラッシュがキューされた場合、Oracle はこの統計情報を増やします。

global cache cr block log flush time これは、ログ・フラッシュに対する待機時間で、サービス時間の一部です。

グローバル・ロック統計情報の分析

グローバル・ロック統計情報には、PCM および非 PCM ロック・アクティビティの件数およびタイミングが含まれます。Oracle は、DLM 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。DLM 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 つであるため、DLM のすべての Oracle クライアントは、このレイヤーを介して DLM 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生したロック要求が含まれ、グローバル・キャッシュの統計情報は、バッファ・キャッシュの Oracle Parallel Server アクティビティに関連しています。

この項の手順に従って、V$SYSSTAT ビューのデータを監視し、平均、待ち時間および件数を導出します。これによって、インスタンスで生成される Oracle Parallel Server 作業負荷の大まかな標識を作成できます。

グローバル・ロック統計情報の分析手順

次の手順に従って、グローバル・ロック処理に対する V$SYSSTAT ビューの統計情報を参照および分析します。

1. 次の構文を使用して、V$SYSSTAT を問い合せます。

```
SELECT * FROM V$SYSSTAT WHERE NAME LIKE 'global lock%';
```

次のように出力されます。

```
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
```

この出力を基にして、残りの手順で説明する計算および分析を行います。

2. 次の式で、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 20 ～ 30ms を超える場合は、DESCEND キーワードを使用して V$SYSTEM_EVENTS ビューの 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いるロック・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 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 数を増やします。変換は、すでにオープンされているロックがあり、Oracle がそのロックを別のモードへ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および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長すぎる場合は、システムでタイムアウトが発生している場合があります。

global lock convert times または global lock get times の値が大きい場合は、V$SYSTEM_EVENTS ビューの統計情報を参照して、TIME_WAITED 統計情報の値が大きいイベントを識別します。

3. 次の式で、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 を超える場合は、DESCEND キーワードを使用して V$SYSTEM_EVENTS ビューの TIME_WAITED 列を問い合せ、ディレイの原因であるイベントを識別します。

4. 前述したように、グローバル・ロックの統計情報は、バッファ・キャッシュのロック・オペレーションおよびデータ・ブロック以外のリソースに対するロック・オペレーションに適用されます。パフォーマンスの問題になっているリソース・タイプを判断するには、global lock get および global lock convert を、次の 2 つのカテゴリに分割します。

– 同期操作

同期ロック取得には、global lock sync gets などが含まれます。これらは、通常、キャッシュされたデータ・ブロック以外のリソースのロック要求に対して実行されます。同期ロック取得に必要な時間の割合を判断するには、global lock get time または global lock convert time を、対応する同期操作の数で割ります。

– 非同期操作

非同期ロック・オペレーションには、global lock async gets などが含まれます。これらは、通常、グローバル・キャッシュ・ロックに対するロック・オペレーションです。同期操作と同じ計算を使用して、合計時間の割合を導出できます。この方法で、作業の割合、およびグローバル・キャッシュ・ロック要求およびその他のロック要求のコストを計算できます。

通常、グローバル・キャッシュ・ロック要求以外のリソースに対するグローバル・ロック要求の割合が、すべてのロック・オペレーションのコストの大半を占める場合、V$SYSTEM_EVENTS ビューには、row cache lock、enqueue または library cache pin に対する長い待機時間が示されます。

5. V$LIBRARYCACHE ビューおよび V$ROWCACHE ビューを分析して、非 PCM リソースに対する DLM アクティビティを観察します。これらのビューには、DLM リソース使用率を識別する DLM 固有の列があります。これらのビューの分析を行うのは、library cache pin、enqueue、DFS lock hadle に対する長時間の待機が頻繁に発生する場合です。

DLM リソース、ロック、メッセージおよびメモリー・リソース統計情報の分析

Oracle は、DLM リソース、ロックおよびメッセージの統計情報を DLM レベルで収集します。これらの統計情報を使用して、DLM 待ち時間および作業負荷を監視します。これらの統計情報は、V$DLM_CONVERT_LOCAL ビューおよび V$DLM_CONVERT_REMOTE ビューに表示されます。

これらのビューは、ロック要求のタイプごとに、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$DLM_CONVERT_LOCAL ビューは、ローカルなロック・オペレーションの統計情報を表示します。V$DLM_CONVERT_REMOTE ビューは、リモート変換の値を表示します。これらのビューの平均変換時間は、100 分の 1 秒単位で表されます。

注意： これらのビューの件数情報は、インスタンスの存在期間中の累積です。

DLM の作業負荷によるパフォーマンスへの影響

DLM の作業負荷は、Oracle Parallel Server およびキャッシュ・フュージョンのパフォーマンスにとって重要です。これは、一貫読込み要求ごとに、ロック要求が発生するためです。要求の割合が高いために DLM の作業負荷が大きくなると、パフォーマンスに悪影響を及ぼします。

DLM 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、それらのノードからの応答を待機する必要があります。ただし、DLM ではほとんどの下位変換がローカル操作です。

次に示す DLM 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、DLM リソースおよびロックの監視方法について説明します。2 番目のグループでは、メッセージ統計情報の監視方法について説明します。

DLM リソースおよびロックの統計情報の分析手順

次の手順に従って、DLM リソース処理に必要な V$DLM_CONVERT_LOCAL ビューおよび V$DLM_CONVERT_REMOTE ビューの統計情報を取得および分析します。

V$DLM_CONVERT_LOCAL ビューおよび V$DLM_CONVERT_REMOTE ビューへ移入するために、イベント 29700 を使用可能にする必要があります。次のように入力します。

```
EVENT="29700 TRACE NAME CONTEXT FOREVER"
```

1. 次の構文を使用して、V$DLM_CONVERT_LOCAL ビューを問い合せます。

```
SELECT CONVERT_TYPE,
AVERAGE_CONVERT_TIME,
CONVERT_COUNT
FROM V$DLM_CONVERT_LOCAL;
```

次のように出力されます。

```
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 ビューを問い合せます。

```
SELECT * FROM V$DLM_CONVERT_REMOTE;
```

V$DLM_CONVERT_LOCAL ビューと同じ形式の出力が戻されます。

V$DLM_CONVERT_LOCAL ビューおよび V$DLM_CONVERT_REMOTE ビューの出力を基に、次の手順で説明する計算を行います。

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. 作業負荷およびロック変換の待ち時間の履歴をメンテナンスすると有効です。トランザクションごとのロック要求が変化していて平均待ち時間が増加しない場合、通常、それはアプリケーションの作業負荷のパターン変更によるものです。

要求の割合および待ち時間が悪化している場合、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がわかります。LMD プロセスの CPU 消費率が高い場合（CPU 消費率が 20% を超えていて、全体のシステム・リソースの消費率が通常どおりの場合）、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。

待ち時間が増加した場合は、CPU データ、および UNIX の sar、vmstat、netstat など、または Windows NT の Perfmon などのユーティリティを使用して取得できる、他のオペレーティング・システムの統計情報も調べてください。

5. V$SYSTEM_EVENT ビューから、LMD に対する CPU ビジー時間の推定値を導出します。

   LMD が使用する CPU 時間を簡単に見積もるには、V$SYSTEM_EVENT ビューに表示される LMD の待機時間イベントを変換できます。これを行うには、LMD プロセスがアイドル状態である時間を表す lkmgr wait for remote messages というイベントを調べます。TIME_WAITED 列には、LMD のアイドル時間の累計が 100 分の 1 秒単位で表されます。

   ビジー時間を導出するには、TIME_WAITED の値を秒に換算します。つまり、17222 センチ秒は 172.22 秒です。この時間が LMD プロセスのアイドル時間になります。この時間を計測時間で割ると、LMD プロセスのアイドル時間の割合になります。この結果を 1 から引くと、その結果が LMD プロセスのビジー時間の割合になります。これは、プロセスごとの CPU 使用率を指定するオペレーティング・システムのユーティリティと比較すると、非常に正確な推定値になります。

   注意： スナップショットを開始および終了させて、正確な計算を行ってください。

DLM メッセージの統計情報

DLM は、直接またはフロー制御を使用して、メッセージを送信します。どちらの方法でも、DLM は各メッセージにチケットというマーカーを付けます。各 DLM のチケット割当てには制限があります。ただし、DLM はチケットを無期限に再使用できます。

DLM 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。DLM がチケットを使用しきった場合、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加するまで、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 プロセスによって管理されます。

チケットの数を制限することによって、負荷が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、負荷が小さい他のノードがあるにもかかわらず、リモート一貫読込みブロック要求の負荷が大きいノードがクラスタ全体のメッセージ・リソースの制御権限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
- DLM messages flor controlled
- DLM messages received
- DLM total incoming message queue length

DLM メッセージの統計情報の分析手順

次の手順に従って、V$DLM_MISC ビューのメッセージの統計情報を取得および分析します。

1. 次の構文を使用して、V$DLM_MISC ビューを問い合せます。

```
SELECT NAME, VALUE FROM V$DLM_MISC;
```

次のように出力されます。

```
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 の出力情報は、オラクル社カスタマ・サポート・センターでデバッグを行うために必要な場合があります。

V$DLM_MISC ビューの出力を基にして、次の手順に従います。

2. 次の式で、2 つのスナップショット間の受信キューの平均の長さを計算します。

$$\frac{\textit{total incoming message queue length}}{\textit{messages received}}$$

Oracle は、新しい要求によって LMD プロセスのメッセージ・キューが開始されると、incoming message queue length の値を増やします。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ると、Oracle は messages received の値を増やします。

多くの要求が同時に LMD に届くと、キューのサイズが増加する場合があります。これは、ロック・アクティビティの量が多い場合、または LMD が大量の一貫読込み要求を処理する場合に発生します。通常、受信キューの平均の長さは 10 より小さくなります。

Oracle Parallel Server の I/O 統計情報の分析

前述したグローバル・キャッシュ統計情報およびグローバル・ロック統計情報の他に、V$SYSSTAT ビューの統計情報を使用して、グローバル・キャッシュ同期化に関連する I/O の作業負荷も計測できます。V$SYSSTAT ビューには、このための 3 つの重要な統計情報があります。

- DBWR forced writes
- Remote instance undo header writes
- Remote instance undo block writes

DBWR forced writes は、データ・ブロックを要求した要求側のノードがブロックを使用する前に、Oralcle がそのブロックをディスクに書き込むことによって、インスタンス間のデータ・ブロック競合を解消するときに発生します。

キャッシュ・フュージョンによって、一貫読込みに対するディスク I/O が最小化されます。これによって、各インスタンスが実行する物理書込みおよび読込みが大幅に削減されます。キャッシュ・フュージョンを導入する前は、リモート・インスタンスからデータが要求される一貫読込みでは、リモート・インスタンスに対して最大 3 回の書込み I/O、および要求側のインスタンスに対してそれに対応する最大 3 回の読込み I/O（データ・ブロック用、ロールバック・セグメント・ヘッダー用、ロールバック・セグメント・ブロック用に 1 つずつ）が必要でした。

V$SYSSTAT ビューにある Oracle Parallel Server の I/O 統計情報の分析

次の統計情報を取得して、グローバル・キャッシュ同期化に必要な書込み I/O の量を明確にできます。

1. 次の構文を使用して、V$SYSSTAT ビューを問い合せます。

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME IN ('DBWR forced writes',
'remote instance undo block writes',
'remote instance undo header writes',
'physical writes');
```

次のように出力されます。

```
NAME                                                          VALUE
------------------------------------------------------------ ----------
physical writes                                               41802
DBWR cross-instance writes                                     5403
remote instance undo block writes                                 0
remote instance undo header writes                                2
4 rows selected.
```

統計情報 physical writes は、特定のインスタンスから発生した DBWR が実行するすべての物理書込みを示します。また、DBWR cross-instance writes の値は、別のインスタンスが変更を行うために要求したデータ・ブロックを含む使用済バッファを書込むことによって発生する、すべての書込みを示します。また、クロス・インスタンス書込みも DBWR によってハンドルされるため、DBWR cross-instance writes はすべての physical writes のサブセットになります。

また、remote instance undo block writes および remote instance undo header writes は、別のインスタンスが一貫読込みバージョンのデータ・ブロックを作成しようとしたが、ブロックをロールバックするために必要な情報がインスタンスのキャッシュにないため、Oracle がロールバック・セグメント・ブロックをディスクに書き込んだ回数を示します。両方とも、DBWR cross-instance writes のサブセットです。Oracle8*i* では、キャッシュ・フュージョンによって、ロールバック情報をエクスポートおよびインポートする必要性が削減されているため、これらの統計情報は、パフォーマンスにとってあまり重要ではありません。ほとんどの場合、インスタンスは、インターコネクトを経由して完全なバージョンのデータ・ブロックを要求側のインスタンスへ送信します。

排他（X）から NULL（N）、または排他（X）から共有（S）へのすべてのロック変換は、ロックのバッファが使用済の場合、ディスクへの書込みに対応付けられていることに注意してください。ただし、Oracle8*i* では、キャッシュ・フュージョンは X-to-S ロック変換を必要としないため、X-to-S ロック変換の数が削減されています。ほとんどの場合、保持側インスタンスは、X ロックを保持します。

2. 次の式で、物理 I/O 全体に対する Oracle Parallel Server 関連 I/O の割合を計算します。

$$\frac{DBWR\ forced\ writes}{physical\ writes}$$

この計算結果とキャッシュ・フュージョン導入前の統計情報の間では、割合が大幅に減少しています。

3. 次の式で、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込む必要がある場合に、ロールバック・セグメントへの書込みが発生した回数を計算します。

$$\frac{(remote\ instance\ undo\ header\ writes + remote\ instance\ undo\ block\ writes)}{DBWR\ forced\ writes}$$

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 の量を示します。キャッシュ・フュージョンによって、この値は非常に小さくなります。

4. グローバル・キャッシュ同期化による読込みの数または割合を見積もるには、NULL（N）から共有モード（S）への変換に対するロック要求数を使用します。この要求数は、V$LOCK_ACTIVITY または V$SYSSTAT の physical reads 統計情報にカウントされます。

次の式では、ローカル作業のみに関する読込みの割合が計算されます。

$$\frac{(physical\ reads - (lock\ buffers\ for\ read)) * 100}{physical\ reads}$$

ここで、lock buffers for read は、N-to-S ロック変換を示します。

これらは「強制読込み」といわれ、ローカル・インスタンスによって変更されたキャッシュ・データ・ブロックが、別のインスタンスからの ping によってディスクに書き込まれる必要があり、その後ローカル・インスタンスが読込みのためにブロックを再取得する場合に発生します。

参照： ローカルまたはグローバル作業の割合、および Oracle Parallel Server クラスタの割合の見積りの詳細は、第 11 章を参照してください。

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、ロック変換の量を種類別に明確にする方法を説明します。この項で説明する作業およびビューは次のとおりです。

- V$LOCK_ACTIVITY ビューを使用したロック変換の分析
- V$CLASS_PING ビューを使用したブロック・クラス別 ping の識別
- V$PING ビューを使用したホット・オブジェクトの識別

V$LOCK_ACTIVITY ビューを使用したロック変換の分析

V$LOCK_ACTIVITY ビューには、インスタンスの存続期間中に発生したロックの下位変換および上位変換の数が示されます。X-to-N 下位変換は、別のインスタンスがリソースを変更するためにロックが下位変換された回数を示します。

下位変換には、X-to-S という種類の変換もあります。この種類の下位変換は、インスタンスが、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を読み込むときに発生します。どちらのロック変換も I/O が伴います。ただし、キャッシュ・フュージョンでは、X-to-S 下位変換はバッファ・ロックが必要ないため削減されます。

V$CLASS_PING ビューを使用したブロック・クラス別 ping の識別

V$CLASS_PING ビューには、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表示することによって、ロック変換アクティビティが示されます。

- 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 および 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示されます。

V$PING ビューを使用したホット・オブジェクトの識別

V$PING ビューは、ホット・ブロックおよびホット・オブジェクトを識別する場合に有効です。各列（FORCED_READS および FORCED_WRITES）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を示します。パフォーマンスの原因が ping または Oracle Parallel Server 自体であると想定される場合は、V$LOCK_ACTIVITY ビューを監視して、Oracle Parallel Server 全体の作業負荷プロファイルを生成します。V$LOCK_ACTIVITY ビューの情報を使用して、ロック変換が発生する割合を記録します。

詳細を取得するには、V$CLASS_PING ビューを使用して、ロック変換および ping が発生しているブロックのタイプを識別します。そのクラスを識別したら、V$PING ビューを使用して、特定の表または索引、および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。

応答時間またはスループット要件が満たされない場合は、通常、V$LOCK_ACTIVITY ビュー、V$CLASS_PING ビュー、V$CACHE ビュー、V$PING ビューまたは V$FILE_PING ビューを調べます。さらに、次のものも調べます。

- V$SYSSTAT。トランザクションごとのロック要求の増加を識別します。
- V$SYSSTEM_EVENT。トランザクションごとのグローバル・キャッシュ・ロックまたは一貫読込みサーバー要求の待機時間の増加を識別します。
- 第 11 章で説明した、完了したグローバル作業およびローカル作業。パフォーマンスの割合に大きい変更があるかどうかを確認します。

通常、アプリケーション・プロファイルおよび作業割合の変更によって、前述のビューを使用した分析の詳細が保証されます。既存のアプリケーションのパフォーマンス問題の診断とは別に、これらのビューは、アプリケーションの設計時、またはパーティション化方法の決定時に有効です。

ラッチ、Oracle Parallel Server および DLM の統計情報の分析

ラッチは、システム・グローバル領域の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。

V$DLM_LATCH ビューおよび V$LATCH_MISSES ビューを使用して、DLM 内のラッチ競合を監視します。これらのビューは、特定のラッチ、そのラッチの統計情報、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。

通常の操作では、ラッチの値の統計情報には制限があります。複数のラッチによって、あるレイヤーでのパフォーマンスがわずかに向上する場合があります。多くの場合、次のいずれかが原因でラッチの競合が高くなります。

- さらに高レベルなパフォーマンスの問題または適切にチューニングされていないシステム
- Oracle 内部の非効率性またはパフォーマンス上のエラー

有効な情報を得るには、次の手順に従います。これらの統計情報を定期的に監視し、ラッチ問題のみに基づく結果を導出することはお薦めしません。ただし、この情報を収集しておくと、オラクル社カスタマ・サポート・センターまたはオラクル社の開発担当に問い合せるときに有効な場合があります。また、ラッチ・チューニングは拡張チューニング・アクティビティの対象になりますが、ほとんどの場合、ラッチ・チューニングが実際のパフォーマンス問題になることはありません。

一方、latch free 待機イベントの TIME_WAITED 値が非常に大きく、V$SYSTEM_EVENT ビューの中で最長時間を示した場合は、これらの手順で得る情報を記録しておきます。

ラッチ、Parallel Server および DLM の統計情報の分析手順

次の手順に従って、ラッチ、Oracle Parallel Server および DLM 関連の統計情報を分析します。

1. 次の構文を使用して、V$LATCH ビューを問い合せます。

   ```
   SELECT * FROM V$LATCH;
   ```

   次のように出力されます。列は、左から statistic name、gets、misses、sleep を表示します。

   ```
   cache buffer handle             46184          1          0
   cache buffers chained        84139946     296547      29899
   cache buffers lru             4760378      11718        227
   channel handle pool                 1          0          0
   channel operations                  1          0          0
   dlm ast latch                  542776        494       1658
   ```

```
dlm cr bast queue              37194          1          0
dlm deadlock list              32839          0          0
dlm domain lock la                 1          0          0
dlm domain lock ta             49164          1          0
dlm group lock latt                1          0          0
dlm group lock tab             25239          1          0
dlm lock table fre            325306        270        327
dlm process hash l              6346          0          0
dlm process table                  2          0          0
dlm recovery domai              2014          0          0
dlm resource hash             683031       1709      41342
dlm resource scan                188          0          0
dlm resource table            182093         70          2
dlm shared communication      190766        211        313
dlm timeout list              113294         40          3
dml lock allocation              261          0          0
22 rows selected.
```

注意： この出力例が 5 列になる場合は、左から、gets、hits、misses、sleeps、sleeps-to-misses ratio です。

2. 前述の手順の出力で、sleeps-to-misses ratio が高い場合は、スリープが発生している場所を判断してください。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;
```

次のように出力されます。

```
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$LATCH および V$LATCH_MISSES の出力を基に、次の手順に従います。

3. 次の式で、この項の手順 1 の V$LATCH 出力を基に、misses に対する gets の割合を計算します。

$$\frac{gets}{misses}$$

misses の数値が大きい場合、通常は同じリソースおよびロックの競合があることを表します。割合の許容範囲は、90 ～ 95% です。

4. この項の手順 1 の V$LATCH_MISSES 出力を基に、misses に対する sleeps の割合を分析します。この割合は、プロセスがラッチをすぐに取得できず、そのラッチを待機しているときに、そのプロセスがスリープ状態になる頻度を表示します。

割合が 2 であれば、misses のたびに、プロセスがラッチを実際に取得する前に、2 回取得しようとしたことを意味します。sleeps-to-misses の数が大きい場合は、通常、プロセス・スケジュールのディレイ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表します。また、1 つのリソースに対する内部の非効率性または同時実行性の高さも表します。たとえば、同じリソースに対して多くのロックが同時にオープンされた場合、プロセスはリソース・ラッチを待機する必要があります。

V$LATCH_MISSES ビューでは、ラッチを取得する関数が WHERE 列に示されます。この情報は、内部パフォーマンス問題を判断する場合に有効です。通常、長期間スリープ状態であったラッチは、V$SESSION_WAIT ビューまたは V$SYSTEM_EVENT ビューの latch free 待機イベントのカテゴリに示されます。

次の項では、V$SYSTEM_EVENTS ビューの使用方法をさらに詳しく説明します。

V$SYSTEM_EVENTS ビューを使用したパフォーマンス問題の識別

V$SYSTEM_EVENT ビューには、キャッシュ・フュージョンおよび Oracle Parallel Server イベントに関するデータが表示されます。プロセスが最も長時間待機したイベントを識別するには、DESCENDING キーワードを使用して、V$SYSTEM_EVENT ビューの TIME_WAITED 列を問い合せます。TIME_WAITED 列には、各システム・イベントの待機時間の合計が示されます。

イベント待機の順序リストを作成すると、パフォーマンスのボトルネックを簡単に特定できます。各 COUNT は、非強制コンテキスト切替えを示します。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時間の累計です。TOTAL_TIMEOUT 列および AVERAGE_WAIT 列の値には、システム効率に関する追加情報が表示されます。

第 11 章で説明したように、TOTAL_WAITS 列および TIME_WAITED 列の値の合計を、トランザクション数で割ることをお薦めします。トランザクションは、保険の見積もり、注文入力などのビジネス・トランザクションとして定義できます。または、予想に従って、ユーザー・コミットまたは実行に基づいて、これらのトランザクションを定義できます。

目標は、トランザクション応答時間に影響する主なイベントの種類を判断することです。一般に、次の式で計算されます。

$$\frac{\textit{response time}}{\textit{number of transactions}} = \frac{\textit{CPU time}}{\textit{number of transactions}} + \frac{\textit{wait time}}{\textit{number of transactions}}$$

このため、合計待機時間は、次のような待機時間のサブコンポーネントに分割できます。

$$\frac{\textit{total wait time}}{\textit{number of transactions}} = \frac{\textit{(db file sequential read tm)}}{\textit{number of transactions}} + \frac{\textit{(global cache cr request tm)}}{\textit{number of transactions}} + \cdots$$

ここで、*tm* は待機時間です。

個別のイベントを追加して合計待機時間を導出し、各イベントの待機時間の割合を観察して、トランザクション応答時間の主なコスト要因を導出する場合に有効です。待機の割合が最も大きい時間を削減することによって、応答時間に最も大きい効果が得られます。

V$SYSTEM_EVENTS の Parallel Server イベント

V$SYSTEM_EVENT 出力に表示される次のイベントは、Oracle Parallel Server イベントの待機を表します。

- 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 リソース関連のイベント

パフォーマンス問題は Oracle Parallel Server に関連している場合があるため、前述のイベント以外のイベントも監視できます。たとえば、次のイベントです。

- Row cache locks
- Enqueues
- Library cache pins
- DFS lock handle

まとめ

グローバル・キャッシュ・イベントの待機時間が他の待機時間に比べて長い場合は、V$SYSSTAT の統計情報およびオペレーティング・システムの Performance Monitor 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どうかを確認してください。global cache busy または buffer busy waits の値が大きい場合は、バッファ・キャッシュ内で競合が増加していることを表します。

データ・ブロック・アドレスのロックが発生し、競合の程度が高い OLTP システムでは、global cache waits イベントが合計待機時間の合計に対して高い割合を示すことは普通です。

row cache lock、enqueue および library cache pin の各行の統計情報に示されるように、非バッファ・キャッシュ・リソースの待機によって大量の待機時間が発生している場合は、V$ROWCACHE ビューおよび V$LIBRARYCACHE ビューで Oracle Parallel Server 関連の問題を監視します。

特に、これらのビューの DLM 列の値を調べます。

一般的な Oracle Parallel Server 問題は、適切に管理されていない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・ロックおよびエンキューに対する待機がプロセスに発生し、V$ROWCACHE ビューには、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が示されます。

第 V 部

Oracle Parallel Server のメンテナンス

第 V 部では、バックアップおよびリカバリについて説明します。第 V 部に含まれる章は、次のとおりです。

- 第 13 章「データベースのバックアップ」
- 第 14 章「データベースのリカバリ」

13

データベースのバックアップ

データを保護するために、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・ファイルのバックアップを取ってください。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・ファイルのバックアップも取ってください。この章では、これらの作業の実行方法について説明します。内容は次のとおりです。

- 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
- オンライン・バックアップおよび Oracle Parallel Server

Oracle Parallel Server 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・バックアップおよびクローズ・バックアップの両方の方法でバックアップを取ることができます。

参照：

- バックアップおよびリカバリの一般的な情報は、『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。
- Recovery Manager（RMAN）の詳細は、『Oracle8*i* Recovery Manager ユーザーズ・ガイドおよびリファレンス』を参照してください。

バックアップ方法の選択

次の 2 つの方法を使用して、バックアップおよびリカバリ操作を実行できます。

- Recovery Manager（RMAN）の使用
- オペレーティング・システムの使用

この章で説明する内容は、特に指定がない限り、どちらの方法にも有効です。

> **参照：** Oracle Enterprise Manager の Backup Wizard の使用方法については、『Oracle Enterprise Manager 管理者ガイド』を参照してください。

この章では、オンラインおよびオフラインのデータ・ファイルおよび表領域間の混同を回避するために、「オープン」および「クローズ」という用語を使用して、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可能であるかを示します。「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のバックアップが取られたことを示します。「全体バックアップ」および「増分バックアップ」という用語は、RMAN によって提供される特定の種類のバックアップのみを示します。

> **参照：** RMAN に関するバックアップおよびリカバリの操作および用語の詳細は、『Oracle8*i* Recovery Manager ユーザーズ・ガイドおよびリファレンス』を参照してください。

REDO ログ・ファイルのアーカイブ

この項では、Oracle Parallel Server の各インスタンス用の REDO ログ・ファイルのアーカイブ方法について説明します。

- アーカイブ・モード
- 自動アーカイブまたは手動アーカイブ
- アーカイブ・ファイルのフォーマットおよび接続先
- 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 では、2 つのアーカイブ・モードが提供されます。ARCHIVELOG モードおよび NOARCHIVELOG モードです。ARCHIVELOG モードでは、REDO ログが上書きされる前に、ログが一杯になると同時にインスタンスによってアーカイブする必要があります。これによって、障害が発生しても Oracle はログ・ファイルをリカバリできます。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生成できます。NOARCHIVELOG モードでは、クローズド・バックアップしか生成できません。

注意： アーカイブは、2 つの方法のいずれかで処理できるインスタンスごとの操作です。

- Oracle Parallel Server 上の各インスタンスは、インスタンス自体の REDO ログ・ファイルをアーカイブできます。
- 次の項で説明するように、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することもできます。

参照： 13-16 ページの「オープンおよびクローズ・データベース・バックアップ」を参照してください。

アーカイブ・モードの変更

オンライン REDO ログ・ファイルのグループを保存するアーカイブ・ロギングを使用するかどうかを判断してください。アーカイブ・ロギングを使用しない場合、一度 REDO ログ・ファイルが再使用可能になると、Oracle が REDO ログ・ファイルを上書きします。

一杯になったオンライン REDO ログ・ファイルのアーカイブを使用可能にするかどうかは、アプリケーションの可用性および信頼性要件に依存します。ディスク障害発生時にどのようなデータも失うことができない場合、ARCHIVELOG モードを使用します。一杯になったオンライン REDO ログ・ファイルをアーカイブするには、追加の管理操作が必要な場合があることに注意してください。

参照： Oracle Parallel Server のアーカイブ・ログ設定方法の詳細は、『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。

Oracle Parallel Server 環境でアーカイブ・ロギングを使用可能にするには、データベースをマウントする必要がありますが、オープンする必要はありません。次に、使用禁止の状態で Parallel Server を起動します。次の手順に従います。

1. すべてのインスタンスを停止します。
2. あるインスタンス上で PARALLEL_SERVER を FALSE にリセットします。

3. PARALLEL_SERVER を FALSE に設定したインスタンスを起動します。
4. 次の文を入力します。

```
ALTER DATABASE ARCHIVELOG
```

5. インスタンスを停止します。
6. PARALLEL_SERVER の値を TRUE に変更します。
7. インスタンスを再起動します。

同じ手順でアーカイブ・ロギングを使用禁止にすることもできます。ただし、ALTER DATABASE 文の NOARCHIVELOG 句を使用してください。

自動アーカイブまたは手動アーカイブ

アーカイブは、LOG_ARCHIVE_START 初期化パラメータの設定値によって、任意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。

- LOG_ARCHIVE_START を TRUE に設定すると、Oracle は一杯になった REDO ログを自動的にアーカイブします。
- LOG_ARCHIVE_START を FALSE に設定すると、Oracle はアーカイブを指示されるまで待機します。

LOG_ARCHIVE_START は、各インスタンスごとに異なる値を設定できます。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 文を使用して、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、オンライン REDO ログ・ファイルは、アーカイブを実行するインスタンスにしかコピーされません。

スレッドがクローズされている場合、アクティブ・インスタンスのアーカイブ・プロセスでは、ログ・スイッチおよびクローズされたスレッドのアーカイブが実行されます。これが実行されるのは、すべての使用可能スレッドのアーカイブ・ログで SCN の範囲をほぼ同じにするために、すべてのスレッドにログ・スイッチを適用する場合です。

手動アーカイブ

LOG_ARCHIVE_START が FALSE に設定されている場合、次のいずれかの方法で手動アーカイブを実行できます。

- SQL 文 ALTER SYSTEM の ARCHIVE LOG 句を使用します。
- SQL 文 ALTER SYSTEM ARCHIVE LOG START を使用して、自動アーカイブを使用可能にします。

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・プロセスによって実行されます。インスタンスの ARCH プロセスによっては実行されません。

手動アーカイブ用の ALTER SYSTEM ARCHIVE LOG 句

ALTER SYSTEM ARCHIVE LOG 手動アーカイブ句には、次のものが含まれます。

ALL	一杯になった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	すべての使用可能スレッドの現行 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、次に一杯になる 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 句を使用して、現行インスタンス以外のインスタンスに対応付けられているスレッドにある REDO ログ・ファイルをアーカイブできます。

参照：

- スレッドおよびログ・スイッチの詳細は、13-15 ページの「ログ・スイッチの強制」を参照してください。
- ALTER SYSTEM ARCHIVE LOG 文の構文の詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。
- 手動および自動アーカイブの詳細は、『Oracle8*i* Recovery Manager ユーザーズ・ガイドおよびリファレンス』および『Oracle8*i* 管理者ガイド』の「アーカイブ REDO ログの管理」を参照してください。

アーカイブ・プロセスの監視

GV$ARCHIVE_PROCESSES ビューおよび V$ARCHIVE_PROCESSES ビューには、それぞれ、データベースおよびインスタンスの様々な ARCH プロセスの状態に関する情報が表示されています。GV$ARCHIVE_PROCESSES ビューは、10*n 行を表示します。n はデータベースのオープン・インスタンス数です。V$ARCHIVE_PROCESSES ビューは、10 行（各 ARCH プロセスごとに 1 行）を表示します。

> **参照：** 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および接続先

アーカイブ REDO ログには、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 つ以上の変数およびファイル名拡張子を含むことができます。表 13-1 に、LOG_ARCHIVE_FORMAT の指定方法を示します。この表の例では、LOG_ARCHIVE_FORMAT= arch%*parameter*、およびすべてのパラメータの上限を 10 文字と想定しています。

表 13-1 アーカイブ 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・パラメータ %t および %T は、Oracle Parallel Server のみで使用できます。たとえば、REDO スレッド番号 7 に対応付けられたインスタンスが LOG_ARCHIVE_FORMAT を LOG_%s_T%t.ARC に設定すると、そのアーカイブ REDO ログ・ファイルの名前は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。

参照：

- アーカイブ REDO ログ・ファイル名のフォーマットおよび接続先の指定の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。
- デフォルト・ログ・アーカイブのフォーマットおよび接続先の詳細は、システム固有の Oracle マニュアルを参照してください。

制御ファイルの REDO ログ履歴

CREATE DATABASE 文または CREATE CONTROLFILE 文の MAXLOGHISTORY 句を使用して、インスタンスが書き込んだ REDO ログ・ファイルの履歴を制御ファイルに保持できます。データベース作成後は、新しい制御ファイルの作成によって、ログ履歴を増減することのみできます。CREATE CONTROLFILE を使用すると、現行の制御ファイル内にあるすべてのログ履歴が破壊されます。

MAXLOGHISTORY 句では、アーカイブ履歴に記録できる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・システムによって異なります。MAXLOGHISTORY が 0（ゼロ）を超える値に設定されている場合、インスタンスがオンライン REDO ログ・ファイルを切り替えるたびに、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（順序における次のログの低 SCN）

注意： LGWR は、REDO ログ・ファイルのアーカイブ時ではなく、ログ・スイッチ中にログ履歴データを制御ファイルに書き込みます。

ログ履歴記録が小さく、ログ履歴が MAXLOGHISTORY で設定された制限を超える場合、古い履歴から順に上書きされます。

リカバリ中、SQL*Plus によって適切なファイル名を付けるためのプロンプトが表示されます。RMAN は、必要な REDO ログを自動的にリストアします。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・ログ・ファイル名を再作成でき、複数の REDO ス

レッドを持つ Parallel Server の自動メディア・リカバリを実行します。使用可能スレッドを 1 つのみ持ち、排他モードでデータベースにアクセスする Oracle インスタンスには、ログ履歴は必要ありません。ただし、1 つのスレッドのみがオープンされている場合でも、複数スレッドが使用可能であるときはログ履歴が有効です。

V$LOG_HISTORY ビューのログ履歴情報を問い合せることができます。V$RECOVERY_LOG には、メディア・リカバリを完了するために必要なアーカイブ・ログに関する情報も表示されます。この情報は、ログ履歴記録から導出されます。

多重 REDO ログ・ファイルには、ログ履歴の複数エントリは必要ありません。各エントリは、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を識別します。

参照：

- リカバリ中に表示される SQL*Plus プロンプトの詳細は、14-8 ページの「REDO ログ・ファイルのリストアおよびリカバリ」を参照してください。
- MAXLOGHISTORY のデフォルト値の詳細は、システム固有の Oracle マニュアルを参照してください。

アーカイブ・ログのバックアップ

一般に、アーカイブ・ログには、それらを作成したノードのみがアクセスできます。Oracle Parallel Server には、次の 3 つのバックアップ・オプションがあります。

- すべてのアーカイブ・ログの書込み先の場所を、すべてのノードで共有します。
- 各ノードのアーカイブ・ログのバックアップを、それぞれのノードで取ります。
- 1 つのノードにアーカイブ・ログを移動してバックアップを取ります。

RMAN を使用して 1 番目および 2 番目の方法を、また、オペレーティング・システム・ユーティリティを使用して 3 番目の方法を実装できます。

RMAN によるアーカイブ・ログのバックアップ

すべてのアーカイブ・ログをクラスタ内のすべてのノードと共有する場合、どのノードもすべてのログを読み込むことができるため、バックアップは非常に簡単になり、どのノードからでも実行できます。次の例では、ノード 1 によってすべてのノードにあるすべての REDO ログのバックアップが取られます。『Oracle8*i* Parallel Server セットアップおよび構成ガイド』を参照して、ディレクトリが共有用に構成されていることを確認してください。

```
rman TARGET INTERNAL/sys@node1 catalog rman/rman@rman

  RUN {
        ALLOCATE CHANNEL t1 type 'sbt_tape' FORMAT 'al_t%t_s%s_p%p';
```

```
SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
BACKUP ARCHIVELOG ALL DELETE INPUT;
RELEASE CHANNEL t1;
}
```

ALTER SYSTEM ARCHIVE LOG CURRENT 文を使用して、すべてのノードでそれ自体の現行ログ・ファイルのバックアップを取ります。

すべてのアーカイブ・ログを共有しない場合、どのノード上でもログのバックアップをローカルに取ることができます。ただし、リカバリする場合、リカバリを開始するノードから、すべてのノードにあるすべてのアーカイブ・ログにアクセスする必要があります。このため、ネットワークを介してアーカイブするメディア管理システムまたは共有ディレクトリ・サービスを使用して、ログ・ファイルのリストアを簡素化することをお薦めします。次の RMAN スクリプトは、CONNECT 句および LIKE 句を使用して、すべてのノードのローカル・バックアップを開始します。

注意： THREAD のかわりに LIKE 句を使用することをお薦めします。これによって、1 つのインスタンスが、別のスレッドが停止しているときに、そのスレッドにかわってログをアーカイブできます。また、LIKE によって識別されるように、様々なインスタンスのアーカイブ・ログの接続先が異なっている必要があります。

```
rman TARGET internal/sys@node1 catalog rman/rman@rman

  RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_n1_t%t_s%s_p%p'
        CONNECT internal/sys@node1;
        SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
        BACKUP ARCHIVELOG LIKE '%/arch1/%' delete input;
        RELEASE CHANNEL t1;
        }

  RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_n2_t%t_s%s_p%p'
        CONNECT internal/sys@node2;
        BACKUP ARCHIVELOG LIKE '%/arch2/%' DELETE INPUT;
        RELEASE CHANNEL t1;
        }

  RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_n3_t%t_s%s_p%p'
        CONNECT internal/sys@node3;
```

```
        BACKUP ARCHIVELOG LIKE '%/arch3/%' DELETE INPUT;
        RELEASE CHANNEL t1;
        }
```

すべてのアーカイブ・ログを、各ノードから別々にアーカイブするかわりに、1 つのノードから 1 つのバックアップ・アーカイブにバックアップを取ります。これによって、リカバリ中にすべてのバックアップを検索しやすくなります。共有ディレクトリを使用せずにアーカイブ・ログをバックアップおよびリストアする場合、オペレーティング・システム・ツールを使用して、アーカイブ・ログをコピーまたは移動します。ログをバックアップまたはリストアする前に、このジョブを実行するためのスクリプトを簡単に作成できます。

すべてのアーカイブ・ログをノード 1 上のローカル・ディレクトリにコピーするには、次のようなスクリプトを使用します。

```
#!/bin/sh
sqlplus system/manager@node1 @switchlog.sql
rcp node2:/u01/app/oracle/product/815/admin/ops/arch2/*
/u01/app/oracle/product/815/admin/ops/arch2
rcp node3:/u01/app/oracle/product/815/admin/ops/arch3/*
/u01/app/oracle/product/815/admin/ops/arch3
```

switchlog.sql スクリプトは、リカバリに必要なすべてのログを確実に取得するために使用します。このスクリプトは、次のようなものです。

```
#!/bin/sh
ALTER SYSTEM ARCHIVE LOG CURRENT;
EXIT
```

RMAN を使用してノード 1 からアーカイブ・ログのバックアップを取る場合、ALTER SYSTEM ARCHIVE LOG CURRENT 文がシェル・スクリプトから実行されること以外は、コマンドは 13-8 ページの例と似ています。

```
rman TARGET internal/sys@node1 catalog rman/rman@rman

  RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_t%t_s%s_p%p';
        BACKUP ARCHIVELOG ALL DELETE INPUT;
        RELEASE CHANNEL t1;
        }
```

RMAN によるアーカイブ・ログのリストア

RMAN は、すべてのバックアップに同時にアクセスする場合、リカバリに必要なすべてのアーカイブ・ログを以前のバックアップから自動的にリストアします。Oracle Parallel Serve 環境では、アーカイブ・ログのバックアップに使用するオプションによって、リストア手順が異なります。

アーカイブ・ログ・ディレクトリを共有する場合は、SET 句を使用してアーカイブ・ログの自動リストアの接続先を変更し、リカバリを開始するノードのローカル・ディレクトリにファイルをリストアできます。

ノード 1 から USERS 表領域をリストアするには、次の RMAN コマンド構文を使用します。

```
rman TARGET internal/sys@node1 catalog rman/rman@rman

  run {
        allocate channel t1 type 'sbt_tape';
        set archivelog destination to
'/u01/app/oracle/product/815/admin/ops/arch1';
        recover tablespace users;
        sql 'alter tablespace users online';
        release channel t1;
        }
```

メディア集中管理システムを使用して各ノードのログ・ファイルをバックアップした場合は、SET コマンドの RMAN AUTOLOCATE オプションを使用できます。リカバリに対していくつかのチャネルを使用する場合、RMAN は、要求されたファイルを最初のチャネルで検索できなければ、すべてのチャネルで検索します。この機能によって、リモート・ノードのローカル・テープ・ドライブを使用して、データベースをリカバリできます。

```
rman TARGET internal/sys catalog rman/rman@rman
  RUN {
        ALLOCATE CHANNEL t1 type 'sbt_tape' parms 'ENV=(NSR_CLIENT=node1)';
        ALLOCATE CHANNEL t2 type 'sbt_tape' parms 'ENV=(NSR_CLIENT=node2)';
        ALLOCATE CHANNEL t3 type 'sbt_tape' parms 'ENV=(NSR_CLIENT=node3)';
        SET AUTOLOCATE ON;
        RECOVER TABLESPACE users;
        SQL 'ALTER TABLESPACE users ONLINE';
        RELEASE CHANNEL t1;}
```

メディア集中管理システムを使用しないで、各ノードからログのバックアップを取った場合、まず、リモート・ノードからすべてのログ・ファイルをリストアし、リカバリを開始するホストにそれらを移動する必要があります。

つまり、次の 3 つのステップでリカバリを実行する必要があります。

1. データ・ファイルをリストアします。
2. アーカイブ・ログをリストアします。
3. リカバリを開始します。

```
rman target internal/sys catalog rman/rman@rman
RUN {
         ALLOCATE CHANNEL t1 TYPE 'sbt_tape' connect internal/sys@node1;
         RESTORE TABLESPACE users;
         RELEASE CHANNEL t1;
         }

         RUN {
         ALLOCATE CHANNEL t1 TYPE 'sbt_tape' connect internal/sys@node2;
         RESTORE ARCHIVELOG
            # this line is optional if you don't want to restore ALL archive logs:
            FROM TIME "to_date('05.09.1999 00:00:00','DD.MM.YYYY HH24:Mi:SS')"
            LIKE '%/2_%';
         RELEASE CHANNEL t1;
         }
RUN {
         ALLOCATE CHANNEL t1 TYPE 'sbt_tape' connect internal/sys@node3;
         RESTORE ARCHIVELOG
            # this line is optional if you don't want to restore ALL archive logs:
            FROM TIME "to_date('05.09.1999 00:00:00','DD.MM.YYYY HH24:Mi:SS')"
            like '%/3_%';
         RELEASE CHANNEL t1;
         }

         EXIT

rcp node2:/u01/app/oracle/product/815/admin/ops/arch2
/u01/app/oracle/product/815/admin/ops/arch2
rcp node3:/u01/app/oracle/product/815/admin/ops/arch2
/u01/app/oracle/product/815/admin/ops/arch2

rman TARGET internal/sys catalog rman/rman@rman

         RUN {
         ALLOCATE CHANNEL t1 TYPE 'sbt_tape';
```

```
ALLOCATE CHANNEL d1 type disk;
RECOVER TABLESPACE users;
SQL 'ALTER TABLESPACE USERS ONLINE';
}
```

注意： このリカバリ手順では、ノード 1 上に生成されたアーカイブ・ログが自動的にリストアされるように、sbt_tape チャネルが割り当てられています。

すべてのアーカイブ・ログを 1 つのノードに移動してバックアップを取った場合、共有ディレクトリを使用した場合と同様にリカバリは簡単です。すべてのログ・ファイルを確実にリカバリできるように、次のようなシェル・スクリプトを使用して、すべてのリモート・ログ・ファイルをコピーします。

```
/rcp_all_logs.sh
rman TARGET internal/sys@node1 catalog rman/rman@rman
RUN {
        ALLOCATE CHANNEL t1 type 'sbt_tape' format 'al_t%t_s%s_p%p';
        BACKUP ARCHIVELOG ALL DELETE INPUT;
        RELEASE CHANNEL t1;
        }
```

チェックポイントおよびログ・スイッチ

この項の内容は次のとおりです。

チェックポイント

Oracle では、チェックポイント取得が一貫して自動的に実行されます。チェックポイント取得を実行すると、Oracle がすべての使用済バッファをディスクに書き込んだ後、チェックポイントを進める必要があります。

> **参照：** チェックポイントの詳細は、『Oracle8*i* パフォーマンスのための設計およびチューニング』を参照してください。

チェックポイントの強制

SQL 文 ALTER SYSTEM CHECKPOINT によって、現行インスタンスまたはすべての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。チェックポイントの強制によって、データベース・バッファへのすべての変更が、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 句はデフォルトです。これは、データベースをオープンしたすべてのインスタンスに強制的にチェックポイント取得を実行させます。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。

リカバリが必要なすべてのインスタンスがリカバリされるまで、グローバル・チェックポイントは終了しません。ただし、グローバル・チェックポイント中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リカバリされる前にチェックポイントが終了する場合があります。

リモート・ノード上で実行中のインスタンスでチェックポイントを強制するには、CONNECT 文を使用して、現行インスタンスを変更します。

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。

> **参照：** リモート・ノード指定の詳細は、4-3 ページの「インスタンスの設定および接続」を参照してください。

ログ・スイッチの強制

Parallel Server は、ある期間オンライン REDO ログ・ファイルのアーカイブに失敗したインスタンスに、ログ・スイッチを強制できます。ログ・スイッチを強制できるのは、そのインスタンスが多くの REDO エントリを生成していないか、または停止したためです。これによって、インスタンスの REDO ログ（REDO ログのスレッド）が長期間アーカイブされないままでいることを防止します。

たとえば、あるインスタンスが停止すると、別のインスタンスがそのインスタンスにログ・スイッチを強制し、その現行 REDO ログ・ファイルをアーカイブできます。SQL 文 ALTER SYSTEM SWITCH LOGFILE は、現行 REDO ログ・ファイルが一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・ファイルへの書込みを強制します。

実行するすべてのインスタンスがログ・スイッチを強制されることを、グローバル・ログ・スイッチといいます。グローバル・スイッチを実行するためには、SQL 文 ALTER SYSTEM ARCHIVE LOG CURRENT 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機します。この文を使用して、単一インスタンスにログ・スイッチを強制します。また、THREAD キーワードを指定して、そのオンライン REDO ログ・ファイルをアーカイブします。

各インスタンスには、INSTANCE FORCE LOG SWITCH 句を使用します。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、削除および改名できるように、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。

クローズ・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に、クローズ・スレッドを強制できます。これは、スレッドの現行ログを削除する場合に有効です。この手順は、スレッドをオープンしているインスタンスが停止した状態でも、オープン・スレッド（現行スレッドを含む）では有効ではありません。たとえば、スレッドがオープン状態の間にインスタンスが異常終了した場合、スレッドのログ・スイッチを強制することはできません。

クローズ・スレッドにログ・スイッチを強制するには、SQL 文 ALTER SYSTEM で ARCHIVE LOG 句を指定して、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して、ログ・スイッチを強制するには、SYSOPER 権限または SYSDBA 権限で接続する必要があります。

参照： SYSDBA 権限または SYSOPER 権限での接続の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

データベースのバックアップ

この項では、Oracle Parallel Server でのバックアップ操作の問題について説明します。この項の内容は次のとおりです。

- オープンおよびクローズ・データベース・バックアップ
- RMAN のバックアップ問題
- オペレーティング・システムのバックアップ問題

オープンおよびクローズ・データベース・バックアップ

Oracle Parallel Server のどのノードからも、すべてのバックアップ操作を実行できます。オープン・バックアップでは、データベースの実行中、その全部または一部のバックアップを取ることができます。ユーザーは、オープン・バックアップ中、データベースのどの部分のデータも更新できます。Oracle Parallel Server 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・ファイルおよび現行の制御ファイルのコピーが含まれます。また、メディア障害発生時までリカバリできるように、後続のアーカイブ 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・バックアップは、データベースがクローズされている間に実行されます。RMAN を使用する場合は、クローズ・バックアップを実行するには、1 つのインスタンスをオープンしないで起動およびマウントする必要があります。クローズ・バックアップを作成する前に、Oracle Parallel Server のすべてのインスタンスを停止してください。データベースがクローズ状態のときは、異なるノードからパラレルにファイルのバックアップを取ることができます。クローズ・データベースの全体バックアップには、すべてのデータ・ファイルおよび現行の制御ファイルのコピーが含まれます。

REDO ログ・ファイルをアーカイブする場合、クローズ・バックアップによってメディア障害発生時までのリカバリが可能になります。NOARCHIVELOG モードでは、クローズ・バックアップによってリストアできるのは、データベースのバックアップ時点までであるため、完全なリカバリはできません。

警告： 全体クローズ・バックアップを実行しない場合は、オペレーティング・システム・ユーティリティを使用して、ARCHIVELOG モードで制御ファイルのバックアップを取らないでください。

オープンまたはクローズ・モードで別の（できれば 2 つの）全体バックアップを作成するまで、アーカイブ REDO ログ・ファイルを消去、再利用または破棄しないでください。

参照：

- 『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。
- 『Oracle8*i* 概要』も参照してください。

オンライン・バックアップおよび Oracle Parallel Server

Oracle Parallel Server でのオンライン・バックアップは、キャッシュを使用しないため効率的です。つまり、ping を発生させずに、クラスタ内の単一インスタンスからオンライン・バックアップを実行できます。

バックアップは主に CPU リソースを使用するため、あまりビジー状態でないインスタンスを使用できます。ただし、ディスク使用を監視して、バックアップによって I/O が飽和していないことを確認する必要があります。I/O がバックアップによって飽和していると、オンライン・ユーザーに悪影響を与える場合があります。

注意： ALTER TABLESPACE...BEGIN BACKUP 文を使用すると、追加の REDO ログが生成されます。

RMAN のバックアップ問題

この項では、次の RMAN のバックアップ問題について説明します。

- RMAN のスナップショット制御ファイルの準備
- RMAN を使用したオープン・バックアップの実行
- ノードの親和性の認識

RMAN のスナップショット制御ファイルの準備

Oracle Parallel Server では、RMAN を使用して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を準備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、このようなスナップショット制御ファイルの接続先ディレクトリがあ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ように指定するには、次のように入力します。

```
SET SNAPSHOT CONTROLFILE TO '/ORACLE/DB_FILES/snapshot/snap_prod.cf';
```

次に、/oracle/db_files/snapshot ディレクトリがバックアップを実行するすべてのノードにあることを確認する必要があります。

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、Oracle Parallel Server の他のデータ・ファイルと同様に、クラスタ内のすべてのノードで共有されます。

RMAN を使用したオープン・バックアップの実行

アーカイブ・ログのバックアップも取る場合は、バックアップの完了後に、ALTER SYSTEM ARCHIVE LOG CURRENT 文を発行します。これによって、バックアップ内のファイルの一貫性を保つために、すべての REDO データを持つことが保証されます。

次のサンプル・スクリプトでは、Parallel Server 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・バックアップが分散されます。この場合、次のことを想定しています。

- データベース内には、20 以上のファイルがある
- 各ノードに 2 台ずつ、計 4 台のテープ・ドライブがある
- スレッド 2 が生成するアーカイブ・ログ・ファイルは、ノード 1 から読み込むことができる

サンプル・スクリプトは、次のとおりです。

```
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);
```

参照： RMAN を使用したオープン・バックアップの詳細は、『Oracle8*i* Recovery Manager ユーザーズ・ガイドおよびリファレンス』を参照してください。

ノードの親和性の認識

クラスタ・プラットフォームによっては、クラスタの特定ノードから一部のデータ・ファイルに対して、他のデータ・ファイルより早くアクセスできる場合があります。RMAN は、この種類の親和性を自動的に検出します。特定のデータ・ファイルのバックアップを取るチャネルを決定する場合、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を優先します。この機能を使用するには、バックアップを取るデータ・ファイルへの親和性を持つクラスタの様々なノードに、RMAN 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';
  ...
}
```

参照： ALLOCATE 文の CONNECT 句の詳細は、『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・システムのバックアップ問題

この項では、オペレーティング・システムにおける次のバックアップ問題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および終了
- 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および終了

オペレーティング・システムによるバックアップでは、あるインスタンスで表領域のオープン・バックアップを開始し、同じかまたは別のインスタンスで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 data files...

....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前に、ALTER TABLESPACE ... BEGIN BACKUP 文を発行しない場合、または処理が完了しない場合、バックアップを取ったデータ・ファイルは、後続のリカバリ操作には使用できません。このようなバックアップをリカバリすることは危険です。また、エラーの原因になり、データの一貫性を損なう可能性があります。

オープン・バックアップの作成時には、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 句は、ユーザーの表領域へのアクセスには影響しません。

完全または不完全メディア・リカバリ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 文の実行からリカバリ終了時点までの間の、すべてのアーカイブ REDO ログを保持します。

オープン・バックアップの作成後、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。この文では、アーカイブが必要なすべて

のオンライン REDO ログ・ファイルがアーカイブされます。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すべてのインスタンスの、すべての使用可能スレッドおよびクローズ・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。

> **参照：** ALTER TABLESPACE 文の BEGIN BACKUP 句および END BACKUP 句の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

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

Oracle Parallel Server で、オペレーティング・システム・ユーティリティを使用してオープン・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 文を発行します。

   これによって、現在起動されていないスレッドを含む、Oracle Parallel Server 環境にある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。

2. ALTER TABLESPACE *tablespace* BEGIN BACKUP 文を発行します。
3. ALTER TABLESPACE 文が正常に終了するまで待機します。
4. オペレーティング・システム環境で、適切な文を発行して表領域のデータ・ファイルのバックアップを取ります。
5. オペレーティング・システム・バックアップが正常に終了するまで待機します。
6. ALTER TABLESPACE *tablespace* END BACKUP 文を発行します。
7. ALTER DATABASE BACKUP CONTROLFILE TO *filename* を使用して、制御ファイルのバックアップを取ります。

ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS 文を使用して、トレース・ファイルに制御ファイルのバックアップを取り、そのトレース・ファイルを識別しバックアップを取ります。

アーカイブ・ログのバックアップも取る場合は、END BACKUP の後に ALTER SYSTEM ARCHIVE LOG CURRENT 文を発行します。これによって、END BACKUP マーカーまでロールバックするためのすべての REDO を持っていることが保証されます。

14

データベースのリカバリ

この章では、Oracle Parallel Server での Oracle リカバリ機能について説明します。内容は次のとおりです。

- インスタンス障害のリカバリ
- メディア障害のリカバリ
- パラレル・リカバリ
- パラレル・インスタンス・リカバリ
- 災害時保護計画

> **参照：** Oracle リカバリの一般的な情報は、『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。

3 つのタイプのリカバリ

この章では、次の 3 つのタイプのリカバリについて説明します。

- インスタンス障害のリカバリ
- メディア障害のリカバリ
- パラレル・リカバリ

インスタンス障害のリカバリ

インスタンス障害は、ソフトウェアまたはハードウェア問題によってインスタンスの動作継続が妨げられる場合に発生します。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に実行されるリカバリについて説明します。

- 単一ノード障害
- 複数ノード障害
- ファスト・スタート・チェックポイント処理
- ファスト・スタート・ロールバック
- インスタンス・リカバリのためのデータ・ファイルへのアクセス
- Oracle インスタンス・リカバリのステップ

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リカバリを実行します。排他モードで実行している単一インスタンスでは、インスタンスは、障害発生または異常終了後に起動されるとすぐにリカバリされます。

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に障害が発生すると、オンライン・インスタンスのリカバリが自動的に実行されます。他のノード上で続行し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影響を受けません。インスタンスが書込みを実行しようとすると、トランザクションは停止します。障害が発生したインスタンスのキャッシュ・リカバリが完了するまで、データーベースに対するすべての操作は停止されます。

> **参照：** 『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。

単一ノード障害

Oracle Parallel Server は、他の実行中インスタンスの SMON プロセスを介してリカバリ操作を調整することによって、インスタンスのリカバリを実行します。あるインスタンスに障害が発生すると、別のインスタンスの SMON プロセスがその障害を認識し、障害が発生したインスタンスに対してインスタンス・リカバリを自動的に実行します。

インスタンス・リカバリでは、障害が発生したインスタンスまたはそのインスタンス上で実行していたアプリケーションの再起動は実行されません。実行中のアプリケーションは、14-2 ページの「インスタンス障害のリカバリ」に示すように、フェイルオーバーによって実行を継続する場合があります。

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リカバリを実行する場合、障害が発生しなかったインスタンスは、障害が発生しているインスタンスによって生成された REDO ログ・エントリを読み込み、その情報を使用して、コミットされたすべてのトランザクションが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。リカバリを実行中のインスタンスは、障害発生時にアクティブだったすべてのトランザクションをロールバックし、それらのトランザクションによって使用されていたリソースを解放します。

> **参照：** アプリケーション・フェイルオーバーの詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

複数ノード障害

あるインスタンスが実行し続けている限り、その SMON プロセスは障害が発生した別のインスタンスにもインスタンス・リカバリを実行します。

Oracle Parallel Server のすべてのインスタンスに障害が発生した場合、インスタンス・リカバリは、次にインスタンスがデータベースをオープンするときに自動的に実行されます。そのインスタンスは、障害が発生したインスタンスの 1 つである必要はありません。そのインスタンスは、Oracle Parallel Server のどのノードからでも、共有または排他モードでデータベースをマウントできます。このリカバリ手順は、1 つのインスタンスが、障害が発生したすべてのインスタンスにインスタンス・リカバリを実行すること以外は、共有モードで実行している Oracle でも排他モードで実行している Oracle でも同じです。

ファスト・スタート・チェックポイント処理

ファスト・スタート・チェックポイント取得の実行は、Oracle のファスト・スタート・リカバリの基礎です。ファスト・スタート・チェックポイント取得の実行は継続的に発生し、Oracle がブロックをディスクに書き込んだときにチェックポイントを進めます。ファスト・スタート・チェックポイント取得を実行すると、常に最も古い変更ブロックが最初に書き込まれ、すべての書込みでチェックポイント時間を進めることができるようにします。これによって、従来のチェックポイント取得の実行で発生するバルク書込みおよびその結果発生する I/O スパイクがなくなり、継続して一定の効率的なパフォーマンスが得られます。

ファスト・スタート・チェックポイント取得のロールフォワード・フェーズの所用時間に対する制限を指定できます。Oracle 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書込み率を自動的に調整し、発行する書込みの数を最小化します。

> **参照：** 詳細は、『Oracle8*i* パフォーマンスのための設計およびチューニング』を参照してください。

ファスト・スタート・ロールバック

Oracle におけるシステム障害のリカバリのロールバック・フェーズでは、非ブロック化ロールバック・テクノロジを使用しています。これは、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ることを意味します。新しいトランザクションが、停止したトランザクションによってロックされている行にアクセスする場合、新しいトランザクションはそ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、停止したトランザクション全体を Oracle がロールバックするまで待機する必要はないため、長時間実行トランザクションによるリカバリ時間への影響はありません。ファスト・スタート・テクノロジによって、データの可用性が最大化され、リカバリ時間が予測可能になります。

さらに、データベース・サーバーは、停止したトランザクションをパラレルでロールバックできます。このテクニックは、停止したトランザクションのロールバックを実行する方が、それをシリアルに実行するよりコストがかからない場合にのみ、新しいトランザクションをブロックしない行に対して使用されます。

> **参照：** ファスト・スタート・ロールバックの詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

インスタンス・リカバリのためのデータ・ファイルへのアクセス

別のインスタンスのリカバリを実行するインスタンスには、障害が発生した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・ファイルに対するアクセス権限が必要です。データ・ファイルが検証に失敗したためにインスタンス・リカバリが失敗した場合、リカバリを実行しようとしたインスタンスには障害は発生しませんが、アラート・ログ・ファイルにメッセージが書き込まれます。

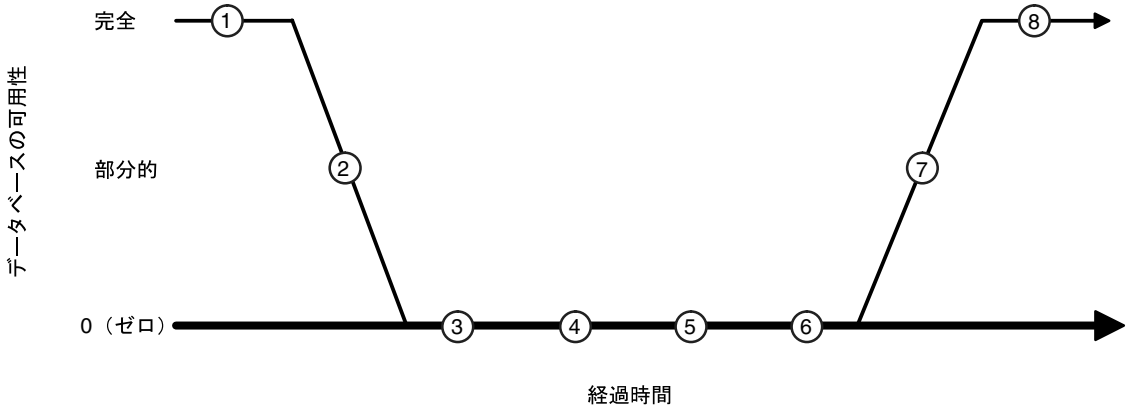
データベース・ファイルへのアクセスを妨げていた問題を修正した後、SQL 文 ALTER SYSTEM CHECK DATAFILES を使用してインスタンスがファイルを使用できるようにします。

> **参照：** 6-2 ページの「データ・ファイル」を参照してください。

Oracle インスタンス・リカバリのステップ

図 14-1 に、インスタンス・リカバリの各ステップでのデータベースの可用性の程度を示します。

図 14-1 Oracle インスタンス・リカバリのステップ



リカバリは、ステップで行われます。

1. Oracle Parallel Server が複数ノード上で実行しています。
2. ノード障害が検出されます。
3. LM が再構成されます。リソースおよびロック・マネージメントが、障害が発生していないノードに再分散されます。1 回のコールで持続リソースが取得されます。ロック値ブロックには、排他モードまたはプロテクト書込みモードで保持されるロックを表す不確定マークが付きます。ロック要求はキューされます。
4. LCK*n* プロセスが、すべての無効ロック要素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、すべての無効ロック要素を有効にします。
7. ロールバックします。ロールバック・セグメントが、コミットされていないすべてのトランザクションのデータベースに適用されます。
8. インスタンス・リカバリが完了し、すべてのデータがアクセス可能になります。

ステップ 5（REDO ログのフォワード適用）の間、データベース・アクセスは、バッファ・キャッシュの推移状態によって制限されます。高粒度または低粒度のロック、あるいは特定

の機能を使用するかどうかにかかわらず、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・アクセス上の制限があります。

- アクセスが制限されている間は、障害が発生していないバッファ・キャッシュにも書込みを行うことはできません。
- バッファ・キャッシュおよびダイレクト・パスを経由して行われるディスク I/O は、どのような I/O であっても、障害が発生していないインスタンスからは実行できません。
- DLM には、ユーザー・データに対するロック要求は行われません。

Oracle は、グローバル・ロックが正常に適用されているキャッシュ内にすでにあるバッファを読み込むことができます。これは、この読込みが、I/O またはロック・オペレーションを必要としないためです。

バッファ・キャッシュの推移状態は、最初のロック・スキャン・ステップが終了すると開始します。このときに、デッド REDO スレッドのスキャンによってインスタンス・リカバリが最初に開始されます。新しいデッド・スレッドが発見されると、後続のロック・スキャンが実行されます。この状態は、REDO ログが適用されている間（キャッシュ・リカバリ中）は継続し、REDO ログが適用され、ファイル・ヘッダーが更新されたときに終了します。キャッシュ・リカバリ操作は、無効ロックの妥当性チェックによって終了します。これは、バッファ・キャッシュ状態が正規化された後に行われます。

メディア障害のリカバリ

メディア障害は、Oracle ファイルの記憶メディアがダメージを受けた場合に発生します。これによって、1 つ以上のデータベース・ファイルが消失したメディア障害の後、Oracle は、データの読込みまたは書込みができなくなります。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リカバリします。Recovery Manager（RMAN）を使用する場合は、増分バックアップ、アーカイブ REDO ログ・ファイルおよび制御ファイルのバックアップを適用する必要がある場合があります。オペレーティング・システム・ユーティリティを使用する場合は、アーカイブ 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必要がある場合があります。

この項の内容は次のとおりです。

> **参照：** 様々なタイプのメディア障害のリカバリ手順については、『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。

完全メディア・リカバリ

完全メディア・リカバリは、排他モードまたは共有モードのいずれでも実行できます。表 14-1 に、特定のデータベース・オブジェクトをリカバリするときに必要なデータベースの状態を示します。

表 14-1 メディア・リカバリに必要なデータベースの状態

リカバリ対象	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、どのインスタンスからもオープンされないようにします。
SYSTEM 表領域以外の表領域	データベースは、リカバリを実行する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である必要があります。
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されているか、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ある必要があります（SYSTEM 表領域内のデータ・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、オープンされないようにします）。

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同時にリカバリできます。

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・リカバリ

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・リカバリを共有モードで実行できます。これには、RECOVER TABLESPACE 文または RECOVER DATAFILE 文を使用します。

RECOVER DATABASE 文を使用して、共有モードでマウントされた、オープンされていないデータベースをリカバリできます。Oracle Parallel Server では、1 つのインスタンスのみがこの文を発行できます。

注意： データベースのリカバリには、RMAN を使用することをお薦めします。できるだけ SQL 文 ALTER DATABASE RECOVER は使用しないでください。

不完全メディア・リカバリ

不完全メディア・リカバリは、データベースが共有モードまたは排他モードでマウントされており、それがインスタンスによってオープンされていない場合に実行できます。これには、次のデータベース・リカバリ・オプションを使用します。

RMAN の場合、リストアおよびリカバリ前に、SET 文で次のいずれかの句を使用します。

- 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、バックアップをリストアしてから、RECOVER DATABASE 文で次のいずれかの句を使用します。

- UNTIL CANCEL
- UNTIL CHANGE *integer*
- UNTIL TIME *date*

> **参照：** SET 文および RECOVER DATABASE 文の使用方法の詳細は、『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。

REDO ログ・ファイルのリストアおよびリカバリ

Oracle Parallel Server によってアクセスされたデータベースのメディア・リカバリには、複数のアーカイブ・ログ・ファイルの同時オープン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ため、リカバリには、スレッドごとに 1 つのアーカイブ・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 ログに十分なリカバリ情報が含まれる場合、そのスレッドにはアーカイブ・ログ・ファイルのリストアは必要ありません。

RMAN を使用したリカバリ

RMAN は、必要なアーカイブ・ログを自動的にリストアして適用します。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 ディレクトリにアーカイブ・ログをリストアします。複数のノードを使用してリストアおよびリカバリしている場合は、アーカイブ・ログがそのリストア / リカバリを実行するノードにリストアされることがあります。

リストアされたログを読み込み、ロールフォワードを実行するノードは、最初に接続されたターゲット・ノードです。次のプラットフォーム固有の方法を使用して、そのノードからログを読み込むことができることを確認する必要があります。

すべてのノードに対してログを読込み可能にする方法 この手順の詳細は、『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。

オペレーティング・システム・ユーティリティを使用したリカバリ

リカバリ中にアーカイブ・ログ・ファイルが必要になると、プロンプトが表示されます。必要なファイルに関する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、ファイル名のフォーマットが LOG_T%t_SEQ%s の場合（%t はスレッド、%s はログ順序番号を示します）、スレッド 8 の SCN 9523 を使用してリカバリを開始するために、次のメッセージが表示される場合があります。

```
ORA-00279: 変更 9523（27/09/91 11:42:54 で生成）にはスレッド番号 8 が必要です。
ORA-00289: 検討すべきログ・ファイル : LOG_T8_SEQ438
ORA-00280: 変更 9523（スレッド 8）は順序番号 438 に存在します。
Specify log: {<RET> = suggested | filename | AUTO | FROM | CANCEL}
```

ALTER DATABASE 文に RECOVER 句を指定する場合、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。Oracle Parallel Server 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があります。ログ・ファイルが不要になると、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るか、リカバリが終了しない限り、そのスレッド用の次のログ・ファイルが要求されます。

リカバリが、追加スレッドが使用可能となった時点に到達すると、Oracle はそのスレッド用のアーカイブ・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するときは、必ず、変更を記録する REDO エントリが書き込まれます。このため、スレッドに関するすべての必要な情報は、リカバリ中、REDO ログ・ファイルから使用できます。

リカバリが、スレッドが使用禁止となった時点に到達すると、Oracle は、そのスレッドのログ・ファイルが不要になったことをユーザーに通知し、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 REDO ログ・ファイル名を再構成する場合、LOG_ARCHIVE_FORMAT でリカバリを実行中のインスタンスに対して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のフォーマットと同じである必要があります。Oracle Parallel Server では、すべてのインスタンスが同じ LOG_ARCHIVE_FORMAT 値を使用する必要があり、リカバリを実行しているインスタンスも同じ値を使用する必要があります。アーカイブ REDO ログ・ファイルが元のアーカイブ先にない場合、リカバリ中に LOG_ARCHIVE_DEST の異なる値を指定できます。

災害時リカバリ

この項では、RMAN 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リカバリについて説明します。災害時リカバリは、災害によってサイト全体が使用不能になった場合に使用されます。この場合、かわりのサイトで、オープンまたはクローズ・データベース・バックアップを使用してリカバリできます。

> **注意：** 最新の時点までリカバリするには、すべてのログがリモート・サイトで使用可能である必要があります。そうでない場合、コミットされたトランザクションがいくつか失われる場合があります。

RMAN を使用した災害時リカバリ

次の場合を想定します。

- データベース全体、すべての制御ファイルおよびオンライン REDO ログを失いました。
- リストアを 2 ノードに分散します。
- 4 台のテープ・ドライブ（各ノード上に 2 つずつ）があります。
- リカバリ・カタログを使用します。

> **注意：** 以前のすべてのバックアップは無効になるため、データベース・リセット・ログのオープン直後にデータベースをバックアップすることをお薦めします。このステップについては、この例では示しません。

データベース構造が最新のバックアップで変更され、その時点までリカバリする場合、SET UNTIL 文を使用します。この方法では、特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、RMAN によってリストアされます。

リストア開始前 : データベースのリストアを開始する前に、次の処理を行う必要があります。

- 最新のバックアップから初期化ファイルおよびリカバリ・カタログをリストアします。
- ディスク上にあってもリカバリ・カタログには登録されていないアーカイブ・ログ、データ・ファイル・コピーまたはバックアップ・セットをカタログに追加します。

  リストアされているログ順序番号までのアーカイブ・ログは、リカバリ・カタログに追加されている必要があります。追加されていない場合、RMAN にはアーカイブ・ログの場所がわかりません。

リカバリ・カタログを頻繁に再同期化する場合、また、リストアしたものからの最新コピーがある場合は、カタログに追加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： リカバリ・カタログを失い、そのリストアおよび失われた時点までのリカバリをすでに実行した場合のみ、この処理を実行する必要があります。リカバリ・カタログが元のままであれば、この処理を実行する必要はありません。ただし、カタログが元のままであっても、いくつかのアーカイブ・ログをカタログに追加する必要がある場合があります。この場合は、最後のカタログ再同期化以降に作成されたアーカイブ・ログのみを再作成します。カタログ再同期化とは、バックアップ、コピーおよびアーカイブ・ログに関する情報が、RMAN によってターゲット・データベース制御ファイルからリカバリ・カタログにコピーされる処理です。

サンプル・スクリプトの実行内容 : 次のスクリプトでは、データベースが最新の使用可能アーカイブ・ログ（ログ 124 スレッド 1）の状態までリストアおよびリカバリされます。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 ユーザーのみに制限します。
- 制御ファイルを指定された場所にリストアします。
- この制御ファイルを CONTROL_FILES 初期化パラメータで指定したその他すべての場所にコピー（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・ログをカタログに追加します。
- データベース・ファイルを（元の場所に）リストアします。

  ボリューム名が変更されている場合、リストア前に SET NEWNAME FOR... 文を使用し、リストア後に切り替える必要があります。これによって、制御ファイルがデータ・ファイルの新しい場所で更新されます。
- 増分バックアップおよび REDO を組み合せて使用するか、または REDO のみを使用して、データ・ファイルをリカバリします。

  RMAN は、指定されたログ順序番号に到達したときにリカバリを完了します。
- データベース・リセット・ログをオープンします。

> **注意：** 他に適用するアーカイブ・ログが確実にない場合にのみ、次の処理を実行してください。

- リセット・ログの後で、データベースのバックアップを取ることをお薦めします。これについては、この例では示しません。

リストア / リカバリ・サンプル・スクリプト :

次のようにして、SQL*Plus を起動します。

```
CONNECT scott/tiger AS SYSDBA
```

Oracle は、次のように応答します。

```
Connected.
```

次の STARTUP 構文を入力します。

```
STARTUP NOMOUNT RESTRICT
```

RMAN を起動し、スクリプトを実行します。

> **注意：** ターゲット・パラメータに指定されるユーザーには、SYSDBA 権限が必要です。

```
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リカバリ

これを実行するには、次の手順を使用してください。

1. 『Oracle8*i* バックアップおよびリカバリ・ガイド』を参照して、最後の全体バックアップを他のサイトでリストアします。
2. SQL*PLus を起動します。
3. SYSDBA として接続します。
4. STARTUP MOUNT 文でデータベースを起動およびマウントします。
5. 適切な UNTIL 句を指定した RECOVER 文を使用して、不完全リカバリを開始します。

   次に例を示します。

   ```
   RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL
   ```

6. 特定のスレッドに対する REDO ログ・ファイル名が提案されたプロンプトが表示された場合は、そのファイル名を使用します。

   提案されたアーカイブ・ログがアーカイブ・ディレクトリにない場合は、ファイルを検索する場所を指定します。スレッドに REDO 情報が必要であり、ファイル名が提案されない場合は、問題のスレッドにアーカイブ・ログを使用してみます。
7. すべてのアーカイブ・ログが適用されるまで、手順 6 を繰り返します。
8. CANCEL 文を使用してリカバリ操作を停止します。
9. ALTER DATABASE OPEN RESETLOGS 文を発行します。

注意： 分散データベース・アクションを使用する場合、リカバリ手順に、調整済分散データベース・リカバリが必要かどうかを確認してください。確認しないと、分散データが論理的に破損する原因となる場合があります。

パラレル・リカバリ

パラレル・リカバリ機能の目標は、計算および I/O のパラレル化を使用して、キャッシュ・リカバリ、単一インスタンス・リカバリまたはメディア・リカバリの実行にかかる時間を削減することです。パラレル・リカバリは、複数のディスク上の複数のデータ・ファイルが同時にリカバリする場合のリカバリ時間の削減に最も有効です。

RMAN では、リストアおよび増分バックアップの用途が、チャネル割当てを使用してパラレル化されます。RECOVERY_PARALLELISM パラメータによって、リカバリに関する同時処理の数が決定されます。

RECOVERY_PARALLELISM を 0（ゼロ）または 1 に設定すると、シリアル・リカバリが起動されます。

RMAN を使用したパラレル・リカバリ

RMAN の RESTORE 文および RECOVER 文を使用すると、Oracle は、次に示すリカバリの 3 段階すべてを自動的にパラレル化できます。

データ・ファイルのリストア： データ・ファイルをリストアする場合、RMAN リカバリ・スクリプトに割り当てたチャネル数によって、RMAN が使用するパラレル化が効果的に設定されます。たとえば、5 つのチャネルを割り当てると、最大 5 つのパラレル・ストリームでデータ・ファイルをリストアできます。

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、割り当てるチャネル数によって潜在的なパラレル化が決定されます。

REDO ログの適用： RECOVERY_PARALLELISM パラメータに設定した値によって決定された数のパラレル・プロセスで、REDO ログが適用されます。

RECOVERY_PARALLELISM 初期化パラメータによって、インスタンス・リカバリまたはメディア・リカバリに関連する REDO アプリケーション・サーバー・プロセスの数が指定されます。パラレル・リカバリ中、1 つのプロセスによって、ログ・ファイルが順次読み込まれ、REDO 情報がいくつかのリカバリ処理にディスパッチされて、ログ・ファイルからの変更がデータ・ファイルに適用されます。値が 0（ゼロ）または 1 の場合は、リカバリがシリアル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。

パラレル・インスタンス・リカバリ

パラレル実行によっても、リカバリ処理は改善されます。リカバリにパラレル実行を使用するには、インスタンス起動時にパラレル実行プロセスが実行されている必要があります。

PARALLEL_MIN_SERVERS パラメータを設定して、パラレル・リカバリに使用可能なパラレル実行サーバーの数を指定します。残りの Oracle 処理にパラレル実行を使用しない場合にも、このパラメータを設定します。PARALLEL_MIN_SERVERS パラメータを使用して、リカバリに使用可能なパラレル実行プロセス数の制限を設定します。

メディア・リカバリ

RECOVER DATABASE 文の PARALLEL 句で、メディア・リカバリの並列度が決定されます。このパラメータの値が 0（ゼロ）以外の場合、また、RECOVER DATABASE 文の RECOVERY_PARALLELISM に値が指定されていない場合、メディア・リカバリでは、並列度のデフォルト値として RECOVERY_PARALLELISM の値が使用されます。RECOVER DATABASE 文の PARALLEL 句を使用して、この並列度をオーバーライドできます。

インスタンス・リカバリの場合、RECOVERY_PARALLELISM パラメータに、リカバリ中に SMON を支援するパラレル実行サーバーの数と等しい値を設定します。

オペレーティング・システム・ユーティリティを使用したパラレル・リカバリ

次の 2 つの方法で、インスタンス・リカバリおよびメディア・リカバリをパラレル化できます。

- RECOVERY_ PARALLELISM パラメータの設定
- RECOVER 文オプションの指定

Oracle Parallel Server は、1 つのプロセスを使用して、ログ・ファイルを順次読み込み、REDO 情報をいくつかのリカバリ処理にディスパッチして、ログ・ファイルからの変更をデータ・ファイルに適用できます。Oracle は、リカバリ処理を自動的に開始するため、1 つ以上のセッションを使用してリカバリを実行する必要はありません。

RECOVERY_ PARALLELISM パラメータの設定

RECOVERY_PARALLELISM 初期化パラメータによって、インスタンス・リカバリまたはメディア・リカバリに関する REDO アプリケーション・サーバー・プロセスの数が指定されます。1 つのプロセスによって、ログ・ファイルが順次読み込まれ、REDO 情報がいくつかのリカバリ処理にディスパッチされます。このリカバリ処理では、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（ゼロ）または 1 の場合は、リカバリが 1 つのプロセスによってシリアルに実行されることを示します。このパラメータの値が、PARALLEL_MAX_SERVERS パラメータの値を超えることはできません。

RECOVER 文オプションの指定

RECOVER 文を使用してインスタンス・リカバリおよびメディア・リカバリをパラレル化する場合、インスタンスへのリカバリ処理の割当ては、オペレーティング・システム固有です。PARALLEL 句の DEGREE キーワードによって、Parallel Server のインスタンス上の処理数またはすべてのインスタンスに分散する処理数が指定されます。

参照 :

- ファスト・スタート・パラレル・ロールバックの詳細は、『Oracle8*i* 概要』を参照してください。
- インスタンスへのリカバリ処理の割当ての詳細は、システム固有の Oracle マニュアルを参照してください。

Oracle Parallel Server でのファスト・スタート・パラレル・ロールバック

初期化ファイル・パラメータ FAST_START_PARALLEL_ROLLBACK を LOW または HIGH に設定すると、ファスト・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、ファスト・スタート・パラレル・ロールバックに関連するサーバー・プロセスの最大数を決定する際に有効です。値が FALSE に設定されると、ファスト・スタート・パラレル・ロールバックは使用禁止になります。

FAST_START_PARALLEL_ROLLBACK の値を LOW に設定すると、ファスト・スタート・ロールバックに使用する処理数は、CPU_COUNT の値の 2 倍になります。値が HIGH の場合は、ファスト・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・サーバー数は、最大で CPU_COUNT の値の 4 倍になります。

Oracle Parallel Server では、複数のパラレル・リカバリ処理は、それらを生成したインスタンスが所有し、そのインスタンス内のみで操作されます。
FAST_START_PARALLEL_ROLLBACK を適切に設定するには、V$FAST_START_SERVERS および V$FAST_START_TRANSACTIONS の内容を調べてください。

ファスト・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。ただし、ファスト・スタート・パラレル・ロールバックでは、各インスタンスがそれ自体のリカバリ処理グループを起動することができるため、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・セグメント処理が改善されます。

災害時保護計画

スタンバイ・データベース、プライマリ / セカンダリ・インスタンス機能の使用および Oracle Parallel Server の高可用性によって、Oracle Parallel Server を障害から保護できます。

スタンバイ・データベースの管理を簡素化するには、管理スタンバイ・データベースの使用を検討してください。

参照 :

- 管理スタンバイ・データベース機能の詳細は、『Oracle8*i* スタンバイ・データベース 概要と管理』を参照してください。
- 高可用性およびプライマリ / セカンダリ・インスタンス機能の詳細は、『Oracle8*i* Parallel Server 概要』を参照してください。

第 VI 部

Oracle Parallel Server の参照情報

第 VI 部では、Oracle Parallel Server についての参照情報を示します。第 VI 部に含まれる章は、次のとおりです。

- 付録 A「Parallel Server 用のデータベース設計の事例」

A

Parallel Server 用のデータベース設計の事例

この付録では、Oracle Parallel Server 用に最適化されたシステムを設計する手順を示す事例について説明します。

事例の概要

この付録で説明する事例では、Oracle Parallel Server で使用する新しいアプリケーションを設計する方法について説明します。これらの方法を使用して、既存のアプリケーションを評価し、Oracle Parallel Server への移行にどれだけ適しているかを判断できます。

注意： 競合を最小化することが目標であることに注意してください。競合の最小化によって、パフォーマンスを最適化できます。

この事例では、初期データベース設計をすでに行っていると想定しています。Parallel Server 用の設計を最適化するには、次の手順に従います。

1. 初期データベース設計を開発します。
2. 表へのアクセスを分析します。
3. トランザクション・ボリュームを分析します。
4. ユーザーおよびデータのパーティション化方法を決定します。
5. 必要に応じて、索引のパーティション化方法を決定します。
6. 高粒度ロックまたは低粒度ロックを選択します。
7. 設計を実装およびチューニングします。

> **参照：** この手順の詳細は、第 III 部「Oracle Parallel Server の設計および配置」を参照してください。

事例 : 初期データベース設計から Oracle Parallel Server へ

この事例では、実際の分析方法を説明します。この事例とは異なるアプリケーションを使用している場合でも、プロセスの理解に有効です。この項の内容は次のとおりです。

- 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

この事例では、「Eddie Bean」という架空のカタログ販売会社を取り上げます。この会社では、多くの注文入力オペレータが様々な製品の電話注文を処理しています。発送処理オペレータが注文に対応し、売掛処理オペレータが請求書を作成します。買掛処理オペレータは、この会社の内部で使用する備品およびサービスの注文を処理します。セールス・マネージャおよび財務アナリストは、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、単一データベース上で操作されるビジネス処理が 3 つあります。

- 注文入力
- 買掛金
- 売掛金

表

Eddie Bean データベースには、次のような表があります。

表 A-1 「Eddie Bean」のサンプル表

表	内容
ORDER_HEADER	注文番号、カスタマの名前および住所
ORDER_ITEMS	注文された製品、数量および価格
ORGANIZATIONS	顧客およびサプライヤの名前、住所、電話番号
ACCOUNTS_PAYABLE	備品およびサービスに対する会社内部の購入および支払いを追跡します。
BUDGET	会社の支出および歳入の貸借対照表
FORECASTS	将来の販売を予想し、現在の業績を記録します。

ユーザー

次のような様々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 日中ほぼ一貫しています。つまり、注文入力、注文処理および出荷が、1 日中発生します。これらの機能は、1 時間単位などで分割されるものではありません。

約 500 の注文が毎日入力されます。各注文ヘッダーは、存続期間中に約 4 回更新されます。したがって、挿入回数の 4 倍の更新が行われることが予想できます。販売業務、会計業務、発送、注文状態のトレースなどの作業を行う多くの従業員が注文ヘッダーを問い合せるため、多くの選択が行われます。

1 つの注文に対して、平均 4 つの項目があります。注文項目が更新されることはありません。ある項目が削除され、他の項目が入力されることはあります。ORDER_HEADER 表には、4 つの索引があります。他の表にはそれぞれ、主キーのみがあります。

予算および予測アクティビティのボリュームは、注文表よりもかなり小さくなります。これらのアクティビティは頻繁に読み込まれますが、更新はあまり頻繁には行われません。予測は予算より頻繁に更新され、一度実際に行われると削除されます。

大量の削除処理は、夜間のバッチ・ジョブとして実行されます。したがって、このメンテナンス・アクティビティを、通常のアプリケーション機能の分析に含める必要はありません。

表へのアクセスの分析

ご使用のデータベース内の表への、既存の（または予測される）アクセス・パターンを分析します。その後、表のパーティション化方法、およびアクセス・パターンによるグループ化方法を決定します。

- 表アクセス分析ワークシート
- 事例 : 表アクセス分析

表アクセス分析ワークシート

表 A-2 に示すワークシートに、アクティビティが多いすべてのデータベース表を記入します。

表 A-2 表アクセス分析ワークシート

表名	1 日のアクセス・ボリューム							
	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

このワークシートに記入するために、各種類の操作のボリュームを見積もります。その後、操作に伴う読込み / 書込みの数を計算します。

操作ボリュームの見積り

表に対して実行する操作の種類ごとに、1 日の発生予測ボリュームを反映した値を入力します。

注意 : この分析では、相対的な値（アプリケーションの通常の使用を示す合計値）が重要です。アプリケーションがまだ存在しない場合も、ユーザーのタイプを想定し、アクティビティの相対レベルを見積もることができます。表に対するメンテナンス・アクティビティは、通常、この分析には関連しません。

操作ごとの I/O の計算

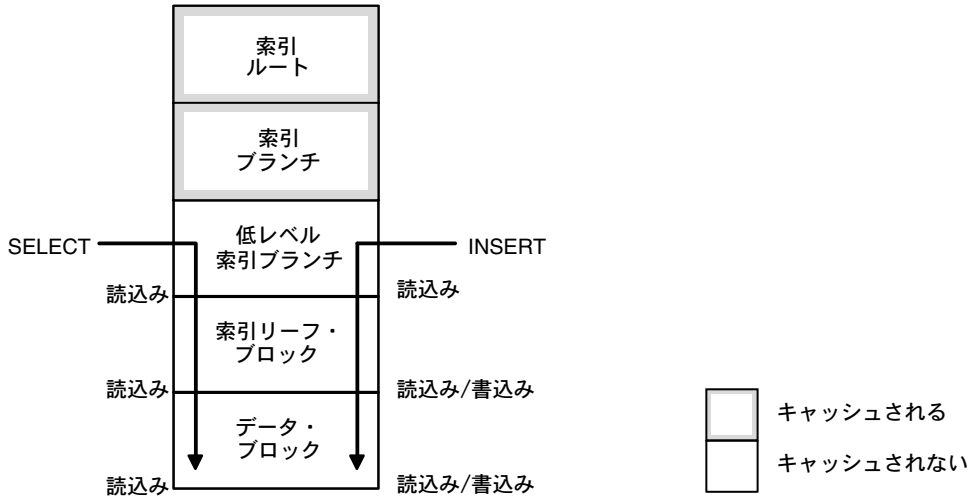
操作列内の各値に対して、最悪のシナリオで生成される I/O の数を計算します。

SELECT 操作には読込みアクセス、INSERT、DATE および DELETE 操作には読込み / 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・ブロックのみでなく、すべての関連する索引ブロックにアクセスします。

注意： 操作ごとに生成される I/O の数は、表のアクセス・パスおよび表サイズによって異なります。また、表が持つ索引の数によっても異なります。たとえば、小さい索引には、1 つの単一索引ブランチ・ブロックのみがあります。

例として、図 A-1 に、大規模な表のデータへの読込み / 書込みアクセスを示します。この表では、2 つのレベルの索引はバッファ・キャッシュになく、高いレベルの索引のみがシステム・グローバル領域（SGA）にキャッシュされます。

図 A-1 SELECT 操作または INSERT 操作ごとの I/O の数



この例の場合、主キーを使用してデータにアクセスすると想定すると、SELECT には次の 3 回の I/O が必要になります。

1. 第 1 下位レベルの索引ブロックを読み込む 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 I/O が 1 回
3. データ・ブロックを読み込む I/O

注意： すべてのルートおよびブランチ・ブロックが SGA にある場合、SELECT に伴う I/O は、リーフ索引ブロック読込みおよびデータ・ブロック読込みの 2 回のみです。

INSERT 文または DELETE 文には、少なくとも次の 5 回の I/O が必要です。

1. データ・ブロックを読み込む I/O が 1 回
2. データ・ブロックを書き込む I/O が 1 回
3. 索引ごとに 3 回の I/O（索引エントリの読込みに 2 回、および索引の書込みに 1 回）

この例の UPDATE には、次の 7 つの I/O が伴います。

1. 第 1 下位レベルの索引ブロックを読み込む 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 I/O が 1 回
3. データ・ブロックを読み込む I/O が 1 回
4. データ・ブロックを書き込む I/O が 1 回
5. 第 1 下位レベルの索引ブロックを再度読み込む I/O が 1 回
6. 第 2 下位レベルの索引ブロックを再度読み込む I/O が 1 回
7. 索引ブロックを書き込む I/O が 1 回

注意： INSERT または DELETE はすべての索引に影響しますが、UPDATE は 1 つの索引にしか影響しない場合があります。変更された索引キーの数をチェックしてください。

サンプル表に対する操作ごとの I/O

事例では、索引数が表ごとに異なるため、操作ごとの I/O 数は表ごとに異なります。

表 A-3 に、ORDER_HEADER 表に関する各タイプの操作によって生成される I/O の数を示します。ORDER_HEADER 表に索引が 4 つあると想定します。

表 A-3 操作ごとの I/O 数 : サンプル ORDER_HEADER 表

操作	選択	挿入	更新	削除
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 : これらの値は、ご使用のデータベース内の実際の索引数および各表に対するアクセス・パスによって調整する必要があります。

表 A-4 に、事例のその他の表に対する操作ごとに生成される I/O の数を示します。各表には、主キー索引のみがあると想定します。

表 A-4 操作ごとの I/O 数 : その他のサンプル表

操作	選択	挿入	更新	削除
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
アクセスのタイプ	3	5	7	5

これらの分析には、データに対して行われた変更もロールバック・セグメントを生成し、追加の I/O が伴うことを考慮する必要はありません。これらの I/O は、インスタンス・ベースです。したがって、これらの変更が Oracle Parallel Server アプリケーションの問題の原因になることはありません。

参照 : 索引の詳細は、『Oracle8*i* 概要』を参照してください。

事例 : 表アクセス分析

表 A-5 に、事例でアプリケーションを標準的に使用している場合のおおよその値を示します。

表 A-5 事例 : 表アクセス分析ワークシート

表名	1 日のアクセス・ボリューム							
	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
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。

- ORDER_HEADER 表および ORDER_ITEM 表のみに、大量の書込みアクセスが発生します。
- ORGANIZATIONS は主に読込み専用表です。ORGANIZATIONS は、INSERT、UPDATE および DELETE 操作のいくつかによっても保持されますが、通常は SELECT のみに使用されます。

ユーザーごとのトランザクション・ボリュームの分析

ご使用のデータベース内の表への、既存の（または予測される）アクセス・パターンを分析します。その後、表のパーティション化方法、およびアクセス・パターンによるグループ化方法を決定します。

トランザクション・ボリューム分析ワークシート

大量の書込みアクセスがある各表について、ユーザーのタイプごとに 1 日のトランザクション・ボリュームを分析します。

> **注意 :** 読込み専用表に対しては、ユーザーのタイプごとにトランザクション・ボリュームを分析する必要はありません。

表 A-6 に示すワークシートを使用します。

表 A-6 トランザクション・ボリューム分析ワークシート

表名 :									
		1 日のトランザクション・ボリューム							
		読込みアクセス		書込みアクセス					
		選択		挿入		更新		削除	
ユーザーのタイプ	ユーザーの数	操作	I/O	操作	I/O	操作	I/O	操作	I/O

まず、ユーザーのタイプごとにトランザクションのボリュームを見積もります。その後、必要な I/O の数を計算します。

事例 : トランザクション・ボリューム分析

次の表に、事例における、大量の書込みアクセスがある 3 つの表 ORDER_HEADER、ORDER_ITEMS および ACCOUNTS_PAYABLE のトランザクション・ボリュームの分析を示します。

ORDER_HEADER 表

表 A-7 に、事例における、ORDER_HEADER 表の値に対するおおよその見積りを示します。

表 A-7 事例 : トランザクション・ボリューム分析 ORDER_HEADER 表

表名 :ORDER_HEADER									
ユーザーのタイプ	ユーザーの数	1 日のトランザクション・ボリューム							
		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
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のみが、この表に挿入を実行します。
- 売掛処理オペレータおよび発送処理オペレータのみが、更新を実行します。
- セールス・マネージャおよび財務アナリストは、選択操作のみを実行します。
- 買掛処理オペレータは表を使用しません。

削除は、メンテナンス操作として実行されます。したがって、この分析で削除を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、現在の月のデータにアクセスし、財務アナリストは、過去のデータにアクセスする場合が多いことがわかります。

ORDER_ITEMS 表

表 A-8 に、事例における、ORDER_ITEMS 表の値に対するおおよその見積りを示します。

表 A-8 事例 : トランザクション・ボリューム分析 : ORDER_ITEMS 表

表名 :ORDER_ITEMS									
ユーザーのタイプ	ユーザーの数	1 日のトランザクション・ボリューム							
		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
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のみが、この表に挿入を実行します。
- 更新はほとんど実行されません。
- 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、大量の選択操作を表に対して実行します。
- 買掛処理オペレータは表を使用しません。

注文ヘッダーは、使用可能な製品リストより多くの変更（住所の変更など）を必要とする傾向があるため、ORDER_HEADER 表には ORDER_ITEM より多くの書込みが行われます。新しい項目は、ジャーナル・エントリとして表示されるため、ORDER_ITEM 表はほとんど更新されません。

ACCOUNTS_PAYABLE 表

表 A-9 に、事例における、ACCOUNTS_PAYABLE 表の値に対するおおよその見積りを示します。この表には、特に大量の書込みアクセスは示されていませんが、買掛処理オペレータが実行する主な操作を含むため、この表も分析しています。

表 A-9 事例 : トランザクション・ボリューム分析 : ACCOUNTS_PAYABLE 表

表名 :ACCOUNTS_PAYABLE									
ユーザーのタイプ	ユーザーの数	1 日のトランザクション・ボリューム							
		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
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。

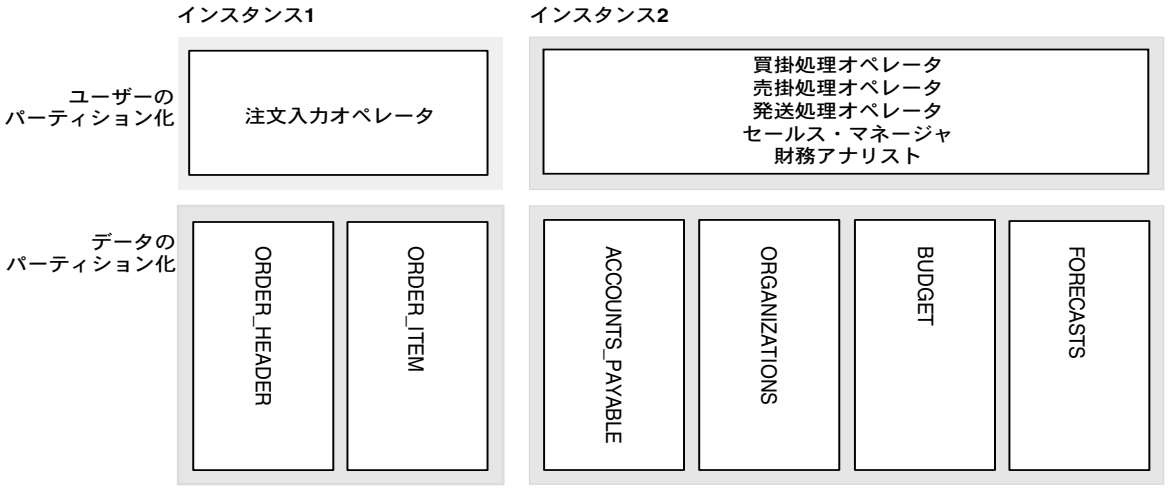
- 買掛処理オペレータは、1 日 50 の購入注文をサプライヤに送信します。この表のデータを変更するのは、このオペレータのみです。
- 財務アナリストは、必要に応じて情報を調査します。

削除は、メンテナンス操作として実行されます。したがって、この分析で削除を考慮する必要はありません。

事例 : 初期パーティション化計画

この事例では、ORDER_HEADER および ORDER_ITEM 表に対して大量の挿入アクティビティを行う多数の注文入力オペレータを、複数のマシンに分割しない例を示します。これらのユーザーは、最もよく使用する 2 つの表とともに 1 つのノードに集約します。したがって、図 A-2 に示すように、注文入力オペレータ用に 1 つのノードを、他のすべてのユーザー用に 1 つノードを設定することをお薦めします。

図 A-2 事例 : ユーザーおよびデータのパーティション化



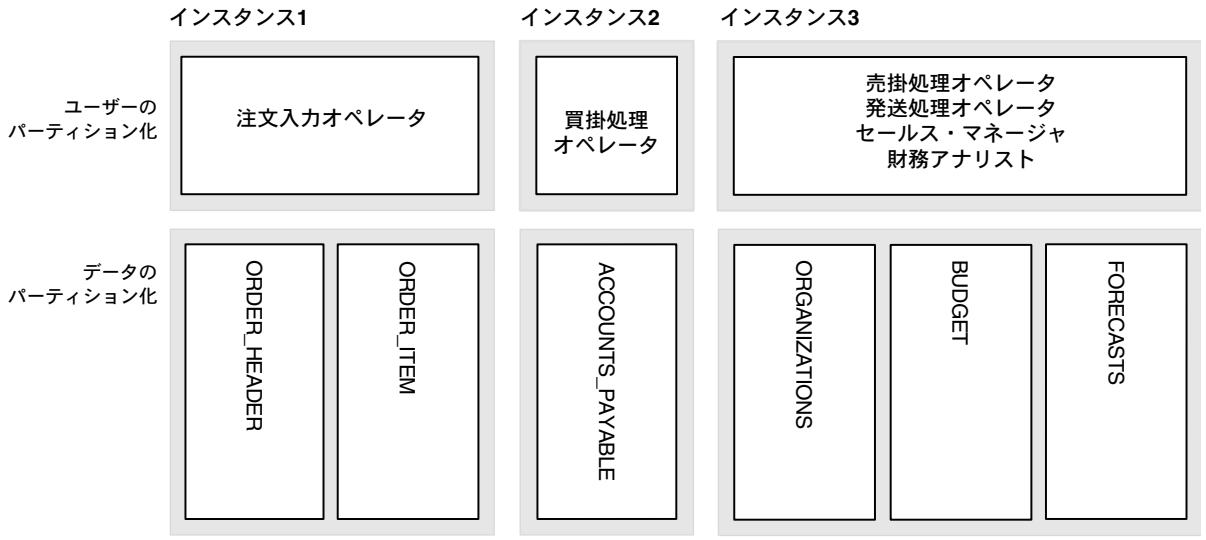
このシステムは、ノード間で適切に均衡化されています。財務アナリストによって行われるデータベース集中型のレポートは、大量のシステム・リソースを使用します。一方、他の注文入力オペレータによって実行されるトランザクションは、比較的単純です。

システムにおけるユーザー数を調整することによってロード・バランスを試みることは、一般的には有効ですが、必ずしも重要ではありません。チューニングに対するロード・バランスの優先順位は、競合の削減より低くなります。

事例 : さらなるパーティション化計画

事例では、買掛データが買掛処理オペレータによってのみ書込みされることは明白です。したがって、図 A-3 に示すように、このデータを異なるインスタンスに効率的にパーティション化できます。

図 A-3 事例 : ユーザーおよびデータのパーティション化 : 設計オプション 1



データの特定の部分に書込みアクセスを行う必要があるすべてのユーザーが 1 つのノードに集中すると、すべての PCI ロックがこのノードに常駐します。したがって、ロック・オーナーシップがインスタンス間で移動することはありません。

この分析に基づいて、次の項で説明する 2 つの設計オプションが使用可能です。

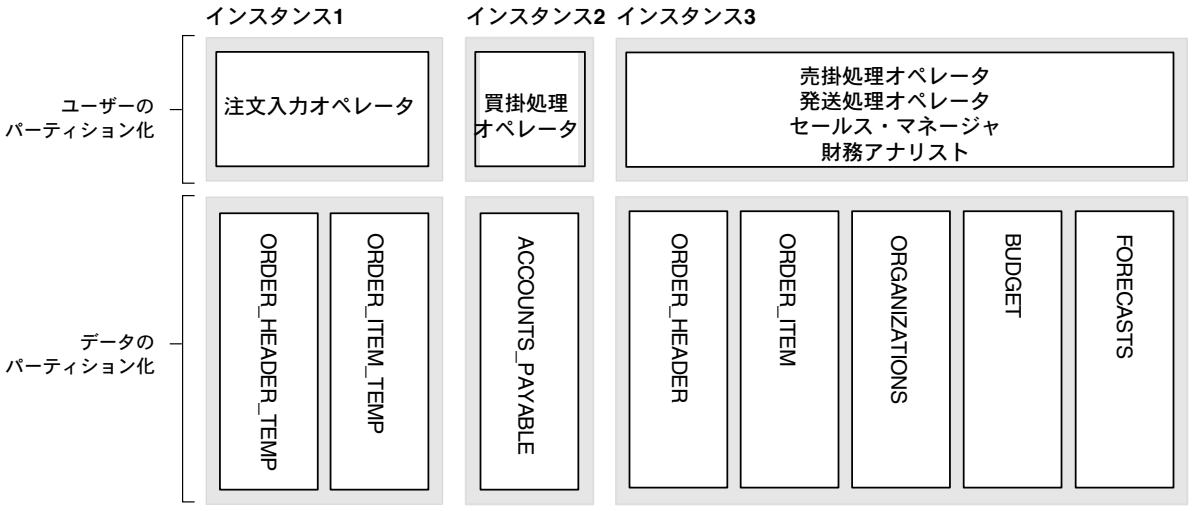
設計オプション 1

図 A-3 に示すように、1 つのインスタンス上のすべての注文入力オペレータとともにシステムをセットアップして、表の排他的 PCM ロックに対する競合を最小化できます。これによって、セールス・マネージャおよび財務アナリストは最新の情報を取得できます。セールス・マネージャおよび財務アナリストは、主に過去のデータを必要とするため、カレント・レコードに対する競合はそれほど多く発生しません。

設計オプション 2

設計オプション 1 のかわりに、ORDER_ITEM/ORDER_HEADER に対して、別々に一時表を実装できます。この表は、新しい注文情報の記録にのみ使用します。すべての問合せが実行される主な表に、夜間に変更を取り込むことができます。この方法は、財務分析に現行のデータが必要ない場合に有効です。これは、財務分析が主に、履歴データの検索を行う場合にのみ有効な方法です。財務分析に最新のデータが必要な場合、この方法は適切ではない場合があります。

図 A-4 事例 : ユーザーおよびデータのパーティション化 : 設計オプション 2



索引のパーティション化

ご使用のシステムにある複数のノードが同じ索引を挿入している場合は、索引のパーティション化を検討する必要があります。この場合、各インスタンスが、索引内の異なる位置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 の事例では、アプリケーションおよびデータの使用方法がパーティション化されているため、この問題は発生しません。

参照：

- 索引の競合を回避するために、空きリスト、空きリスト・グループおよび順序番号を使用する際のヒントについては、8-6 ページの「索引用の空きリストの作成」を参照してください。
- 空きリスト・グループの使用を回避するために、表およびインスタンスを物理的にパーティション化する場合の考慮点については、『Oracle8*i* 概要』を参照してください。

高粒度ロックまたは低粒度ロックの実装

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・ファイルに対する高粒度ロックまたは低粒度ロックを使用するかどうかを決定する必要があります。

高粒度ロックを使用する最悪のケースを想定して設計する必要があります。その後、設計または監視フェーズで、ロックが多すぎることが判明した場合、または false ping の疑いがある場合は、低粒度ロックを試みる必要があります。

分析は、データベース・レベルから開始します。表 A-10 に示すワークシートを使用できます。

表 A-10 ワークシート：高粒度ロックまたは低粒度ロックに対するデータベース分析

ブロック・クラス	関連パラメータ	高粒度ロックまたは低粒度ロックのいずれかの使用

次に、表 A-11 に示すワークシートにファイルおよびデータベース・オブジェクトを記入します。各ファイルに使用するロック・モードを決定します。

表 A-11 ワークシート : 高粒度ロックまたは低粒度ロックを使用する時期

ファイル名	含まれるオブジェクト	高粒度ロックまたは低粒度ロックのいずれかの使用

参照 : データ・ファイルに対するロックの適用の詳細は、第 9 章「インスタンス・ロックの設定」を参照してください。

設計の実装およびチューニング

この時点で、見積った値を使用した分析は終了しました。設計を完了するためには、アプリケーションをプロトタイプ化するか、実際に実装する必要があります。実システムを観察することによって、このシステムをさらにチューニングできます。

次の方法でチューニングを行います。

- ping されているブロックを識別し、競合が存在する場所を判断します。
- ping および false ping を削減するために、1 つのインスタンスから他のインスタンスにユーザーを移動することを検討します。
- 高レベルの false ping を検出した場合、各ファイルにさらに多くのロックを適用することによって粒度を増加させることを検討します。
- 挿入時に ping が発生する場合、空きリストを調整するか、複数のシーケンス・ジェネレータを使用して、挿入が索引の異なる部分で実行されるようにします。

参照 : 『Oracle8*i* パフォーマンスのための設計およびチューニング』を参照してください。

索引

A

B

C

D

E

F

G

H

N

O

P

R

S

T

U

V

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

の

は

ひ

ふ

へ

ほ

ま

み

め

も

ゆ

よ

ら

り

る

ろ

わ