Oracle Real Application Clusters（RAC）のアプリケーション開発のベスト・プラクティス

開発者用チェックリスト

Oracle ホワイト・ペーパー | 2016 年 7 月

目次

はじめに

このホワイト・ペーパーは、Oracle Real Application Clusters（RAC）データベース環境で使用するスケーラブルで高可用性を備えた高パフォーマンス・アプリケーションを開発するアプリケーション開発者用のチェックリストとして使用できます。このホワイト・ペーパーでは、関連する一般的なベスト・プラクティスに従って Oracle RAC データベース環境がセットアップされていることを前提としています。このホワイト・ペーパーの対象読者は、アプリケーション開発者コミュニティとデータベース管理者です。

用語

略語	説明
AC	アプリケーション・コンティニュイティ
API	アプリケーション・プログラミング・インタフェース
AQ	アドバンスト・キューイング
キャッシュ・フュージョン	サーバー上の複数のローカル・キャッシュを、グローバルで、調整され、同期されたキャッシュに'融合する'機能
FAN	高速アプリケーション通知
FCF	高速接続フェイルオーバー
FTS	全表スキャン
GCS	グローバル・キャッシュ・サービス
HA	高可用性
JDBC	Java Database Connectivity
OCI	Oracle Call Interface
Oracle XA	Oracle による X/Open 分散トランザクション処理インタフェースの実装
Oracle RAC	Real Application Clusters
SGA	システム・グローバル領域
TAF	透過的アプリケーション・フェイルオーバー
TG	トランザクション・ガード
UCP	Universal Connection Pool

RAC概要

Oracle Real Application Cluster は、オプションながらよく知られた Oracle Database の機能の 1 つで、複数のデータベース・インスタンスをクラスタ化し、共有する共通の物理データベースに同時にアクセスすることができます。このようなクラスタリングにより、容易にサーバーを追加して容量を増やしつつデータベースで N-1（N = クラスタ内のノード数）個のノード障害を許容できるため、データベースの高可用性とスケーラビリティを実現できます。Oracle RAC では、基盤となるグルーとして Oracle Clusterware を使用して同じクラスタ内のサーバーを結合するため、アプリケーションとエンドユーザーからは単一のシステムのように見えます。クラスタのサーバーを同期された状態に保つため、専用の高速かつ低レイテンシなプライベート・ネットワークを使用します。

Oracle RAC データベース内の各インスタンスは別々のクラスタ・メンバー（ノード）上で稼働し、独自の REDO スレッドと UNDO 表領域があります。各インスタンスには、管理対象となる独自の SGA があります。ただし、専用のプライベート・インターコネクトを介して、すべてのインスタンスがそれぞれのキャッシュをクラスタ内の他のインスタンスとの間で共有します。インスタンスのキャッシュ内に他のインスタンスが必要とするデータがある場合、そのデータは全体的なデータの整合性と一貫性を維持しつつ高速インターコネクトを介して送信されます。データベースのデータ・ファイル、制御ファイル、および REDO ログ・ファイルは高可用性共有ストレージに保管され、通信を管理する Oracle Clusterware を使用してすべてのクラスタ・ノードからアクセスできます。図 1 に、2 ノード RAC クラスタの基本的なアプリケーション実装アーキテクチャを示します。

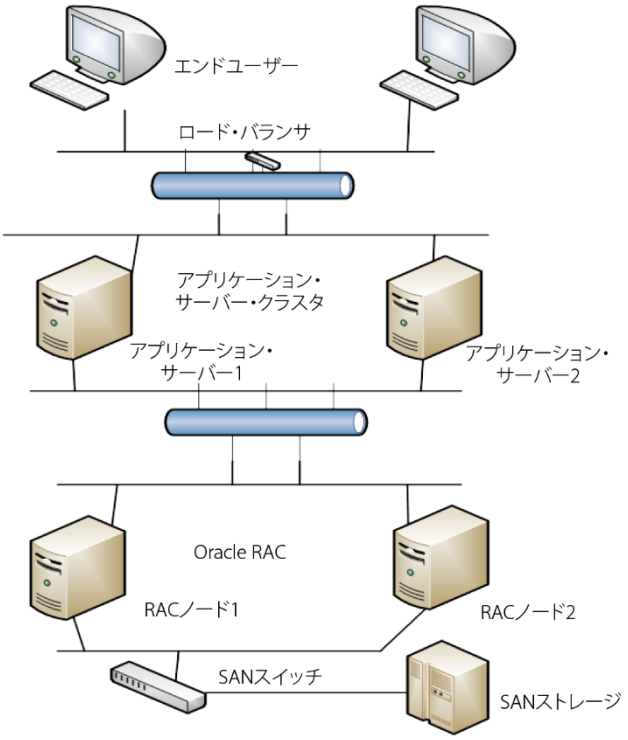

図1：Oracle RACを使用したアプリケーション・デプロイメント・アーキテクチャ（2ノード・クラスタの例）

アプリケーション開発者は、Oracle RAC のメリットを十分に活用可能なアプリケーションを設計およびビルドする場合、ベスト・プラクティスを適用するために、基盤となるデータベース・アーキテクチャを理解しておくと役立ちます。アーキテクチャに精通していれば、一般的ではあっても予防可能な問題を防止することもできます。これには、クラスタの機能を利用する効率的なアプリケーションを記述することが含まれます。このホワイト・ペーパーでは、アプリケーションの設計者および開発者の観点からこれらの側面について概要を述べます。

スケーラビリティの考慮事項

スケーラビリティとは、ワークロードの要求が増えたときに追加のリソースを活用することによってその容量を増やすことができる、アーキテクチャの能力のことです。Oracle RAC データベース環境の場合、これは 1 つ以上のデータベース・インスタンスを追加するだけで追加のワークロードに対処できるようになることを意味する場合があります。

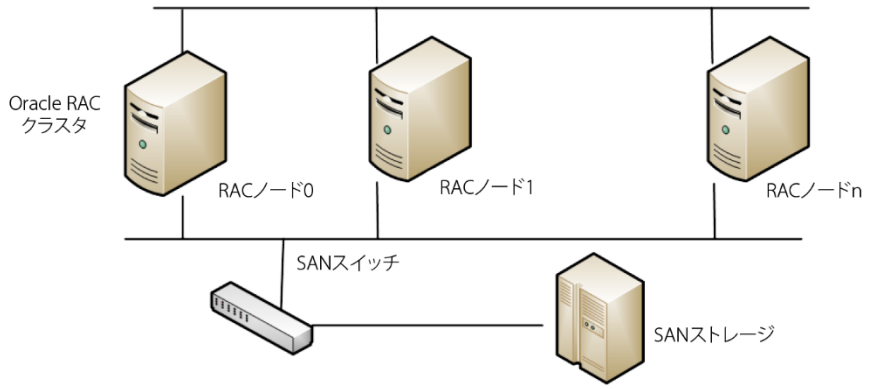

図2：RACのスケーラビリティ（クラスタへのノードの追加）

クラスタ・アーキテクチャがスケーラビリティを持つためには、インスタンス・レベルでの追加（増分）の CPU およびメモリ（キャッシュ）リソースの可用性と、スケーラブル・ストレージ・ソリューションが主要な役割を果たします。

スケーラビリティを念頭に置いてアプリケーションを設計する

優れたアプリケーションは、実行中にホットスポットや競合ポイント、シリアライズが一切発生しないように作成されており、本質的にスケーラブルです。アプリケーションがそのように作成されている場合、クラスタ環境と非クラスタ環境の両方で良好なパフォーマンスを示します。ただし、アプリケーションがいつもスケーラビリティを念頭に記述されているとは限りません。多くの場合、開発者は機能を高めることに重点を置きます。通常、パフォーマンスとスケーラビリティは後回しにされます。既存アプリケーションのパフォーマンスは、クラスタ環境ではいっそう際立つため、アプリケーションをクラスタ環境に移植する前にチューニングが必要な場合があります。オラクルでは、RAC 環境でのアプリケーションのスケーラビリティを高めるには、非 RAC 環境でのスケーラビリティも高めなければならないと常に考えています。

アプリケーションのワークロード処理がリソースのレベルに正比例して増加する場合は、線形的な拡張と言われます。既存のアプリケーションをシングル・インスタンスのデータベース環境から RAC データベース環境に移動する場合は、まずアプリケーションのスケーラビリティをテストし、必要なチューニングを施すことが重要です。Oracle Real Application Testing はデータベース層のイ

ンフラストラクチャ変更テスト向けに設計および最適化されており、実際のアプリケーションの本番ワークロードを使用して、テスト環境でデータベース・パフォーマンスを検証できます。

アプリケーションを記述する際に開発者が時として見過ごすベスト・プラクティスがあります。以下の習慣は、スケーラビリティが不十分なアプリケーションの原因となります。

- コードを適切に再使用しない（例：バインド変数や解析コードを反復使用しない）
- 全表スキャンを使いすぎ、索引付けを有効に利用しない
- 接続および接続プールを効率よく最適に使用しない、など

上記の詳細については、Oracle のドキュメントを参照してください。また、アプリケーションにスケーラビリティを持たせるため、ベスト・プラクティスを念頭に置いてデータベースを作成する必要があります。データベース実装のための Oracle RAC のベスト・プラクティスには詳しい情報があり、データベース管理者やアーキテクトが参照できます。

以下の考慮事項はスケーラブルなアプリケーションの設計に適用されます。

ホットスポットを最小限に抑えるか排除するためのオブジェクトおよびスキーマを設計する

RAC バックエンド・データベース・アーキテクチャによってアプリケーションのスケーラビリティを最大限に高めるには、ホットスポットを最小限に抑えることが重要です。ホットスポットは、（特に、別々のインスタンスの）アクティブな複数のセッションが一連の共通ストレージ・ブロックに頻繁にアクセスした場合に発生します。更新と読取りが頻繁な表または表の一部、およびデータベース・ブロックあたりの行密度が高い場合には、シリアライズが原因で"グローバルにホット"になる可能性があります。これに関して、ホットスポットを最小限に抑えるための考慮事項をいくつか以下に示します。

- **適切な行密度を使用する。**すべてのデータベース IO はデータベース・ブロックの単位です。通常、データベース・ブロックが大きいほどブロックあたりの行数も多くなる可能性があります。それらの行への同時アクセスが行われて複数のインスタンスからロックされると、ホット・ブロックが発生する可能性があります。そのような状況では、サイズの小さなブロックを使用するか、ストレージ・ブロックの密度を下げることが有効なことがあります。Oracle データベースでは、同じデータベース内にブロック・サイズが異なる表領域を作成できます。
- **索引表領域でサイズの小さなブロックを使用することを検討する。**索引エントリのデータ要素の数は少ないため、索引の行密度は同じブロック・サイズの表の場合よりもさらに高くなる可能性があります。索引表領域のブロック・サイズを小さくすると、そのようなホットスポットを軽減できる場合があります。
- **ローカル・アクセスによってホットスポットを防止する。**多くの場合、アプリケーションによるグローバルなホット・ブロックへのアクセスをローカライズして、複数のインスタンスではなく 1 つのインスタンスからのアクセスにすることが有効で実際に役立ちます。これは、アプリケーション全体では RAC データベースへの通常アクセスを続けることができる一方で、特定の機能はローカル・インスタンスからのみ実行でき、必要に応じてフェイルオーバーできることを意味します。この方法は、ワークロード・パーティションと呼ばれることもあります。

- **ハッシュ・パーティション化された表および索引を利用する。**ハッシュ・パーティション化を使用することにより、IO をある程度ランダムに分散させ、ホットスポットを最小限に抑えるか排除できる場合があります。ハッシュ・パーティション化されたグローバル索引を使用すると、索引エントリがパーティション化キーに基づいて複数のパーティションにハッシングされるため、競合を複数のパーティションに拡散させてホットスポットを軽減することができます。ハッシュ・パーティション化されたグローバル索引は、パーティション化されていない表でも利用できます。詳しくは、Oracle Support Note 220970.1 - RAC：よくある質問とこの OTN Note を参照してください。
- **逆キー索引を利用する。**シーケンス（順次増分される数字）に対応する索引には、複数のセッションが同時に行われて同じブロックに複数の値が挿入されるため、ホットスポットが常に 1 つ存在している可能性があります。多くの状況において、そのような順次生成される数字は必要ないと考えられます。それ以外の状況では、（通常の索引ではなく）逆キー索引によってホットスポットを排除することが可能です。
- **ローカル索引を使用して索引メンテナンスを最適化する。**パーティション・レベルでローカル索引を使用することにより、索引メンテナンス中のオーバーヘッドを削減できます。ローカルでパーティション化された索引には、関連付けられた表のパーティションとの間に 1 対 1 の関係があります。ローカル索引パーティション化では、各索引パーティションが相互に独立しているため、索引メンテナンスが容易になり、効率もよくなります。したがって、ローカル索引を使用すると索引メンテナンス操作中の全体的なオーバーヘッドが削減されます。詳しくは、Oracle のドキュメントを参照してください。

全表スキャンを最適に使用する。全表スキャン（FTS）は必要な場合にのみ使用し、可能かつ適切であれば、索引ベースのデータへのアクセスを最適化してください。RAC では、完全な順次読取りがリモート・インスタンスからのデータ取得よりも有利かどうかがチェックされません。そのため、他のインスタンスのアクティブ・ブロックが必要な見込みという結果の場合、広範なキャッシュ調整と同期が行われることになります。したがって FTS は、シングル・インスタンスのデータベースの場合と同様、クラスタ化されたデータベース環境では慎重に使用する必要があります。

アプリケーションのスケーラビリティをテストする

本番環境での使用を開始する前にスケーラビリティをテストすることにより、ホットスポットを発見でき、事前にアプリケーションをチューニングする機会が得られます。スケーラビリティ・テストは、本番さながらの環境において、本番さながらの代表的なワークロードで実施するのが最善です。Oracle Load Testing 12c は、アプリケーションのスケーラビリティ・テストを実施するための便利なツールです。Oracle Load Testing について詳しくは、データ・シートを参照してください。Oracle Real Application Testing（RAT）を使用すると、スタンドアロン・データベースからのワークロードを記録し、RAC データベースで再実行できます。Oracle RAT について詳しくは、Oracle 12c RAT Overview ホワイト・ペーパーを参照してください。また、Oracle Public Cloud で RAC データベースをセットアップすることについても検討してください。これにより、限られた時間内に迅速かつ費用対効果に優れた方法で、RAC データベース環境を構築できます。Oracle Database Appliance などの Oracle エンジニアド・システムでは、CPU とメモリ・リソースを動的に構成し、シングル・インスタンスまたは RAC インスタンス構成を使用してアプリケーションの垂直および水平方向のスケーラビリティをテストすることができます。

XA アプリケーションで分散トランザクション処理（DTP）サービスを使用する

グローバル・トランザクションの複数のブランチがデータベースにアクセスする XA トランザクションを使用するアプリケーションにおいて、同じ RAC データベース・インスタンスを利用してトランザクションを処理することによりメリットを得られる場合は、分散トランザクション処理を使用します。RAC XA アフィニティを提供する接続プールでは、DTP サービスを使用する必要はありません。たとえば、Oracle WebLogic のマルチプール・データ・ソースと Grid Link では、XA アフィニティの最適化を行うことができます。詳しくは、ホワイト・ペーパーXA and Oracle controlled Distributed Transactions を参照してください。

RAC データベースで自動セグメント領域管理（ASSM）を使用する

自動セグメント領域管理によりセグメント領域の管理が大幅に簡素化され、新しいインスタンスがオンラインになるときに、PCTUSED、FREELISTS、および FREELIST GROUPS などのストレージ・パラメータを手動で管理する必要がなくなります。『Oracle Support Note 180608-1 - Automatic Space Segment Management in RAC』を参照してください。自動セグメント領域管理により、データベースのデータ・ブロック内部でさらに効率よく領域管理を行うことができます。この機能は、クラスタ化されたデータベース環境で使用する必要があります。

データベース・シーケンスを最適に使用する

シーケンスはデータベースに属するスキーマ・オブジェクトの 1 つで、そのデータベースに属しているすべてのインスタンスによって共有されます。シーケンス・オブジェクトを拡張するためのアプローチの 1 つでは、クラスタ環境において、順序番号のキャッシュをインスタンス・レベルでローカルに使用します。これにより、順序番号の生成で隔たりが発生する可能性がありますが、大半のアプリケーションで問題となることはなく、シーケンスの競合を軽減するための実行可能なソリューションとすることができます。キャッシングでも順序付けされていない順序番号をさまざまなインスタンスで使用する結果となる可能性がありますが、ほとんどの場合、より高いスケーラビリティを達成するためのシナリオでリジリエンスを持つようにアプリケーションを設計できます。『My Oracle Support（MOS）Note 62002.1 - Caching Oracle Sequences』を参照してください。順序番号のキャッシングにより、クラスタ化されたデータベース環境で実行されているアプリケーションのパフォーマンスを著しく向上させることができます。たとえば、以下に示す 2 つのシナリオの経過時間（パフォーマンス）を参照してください。

```
Create sequence mycached_seq start with 1 increment by 1 cache 1000000;

set timing on;

declare

    xyz number;

    i number;

    begin

    for i in 1..100000 loop
```

```
    select mycached_seq.nextval into xyz from dual;
    end loop;
    end;
/
PL/SQL procedure successfully completed.
Elapsed:0:00:10.38
```

今度は、NOCACHE シーケンスで同じ匿名コードを試します。

```
Create sequence mynocached_seq start with 1 increment by 1 NOCACHE;
Set timing on;
declare
    xyz number;
    i number;
    begin
    for i in 1..100000
    loop
    select mynocached_seq.nextval into xyz from dual;
    end loop;
    end;
/
PL/SQL procedure successfully completed.
Elapsed:0:03:57.44
```

リスト 1：順序番号キャッシング

上記の例から明らかなように、シーケンスをインスタンス・レベルでローカルにキャッシングすると、クラスタ環境でのシリアライズのボトルネックが軽減されます。

アドバンスト・キューイングを最適に使用する

Oracle RAC を使用して、Oracle Advanced Queuing（AQ）に高可用性とスケーラビリティを持たせることができます。JMS ドライバを介して排他的にキューへのアクセスが行われる場合は、JMS Sharded Queue によって Oracle RAC でのパフォーマンスが最適化され、管理性が改善されます。AQ PL/SQL API を介してキューへのアクセスが行われる場合は、別々の RAC インスタンスによってそれぞれのキューが管理されるようにして AQ のパフォーマンスを改善できます。キュー表の場合は、さまざまなインスタンス・アフィニティまたはプリファレンスを指定し、キュー操作が別々のキューでパラレル化されるようにすることができます。オラクルでは、キュー表にインスタンス・アフィニティを設定することをお勧めします。インスタンス・アフィニティを設定することにより、

キュー・モニターのスケジューリングおよび伝播でバックグラウンド処理が分散されるようにすることができます。インスタンス・アフィニティを設定しない場合は、利用可能なインスタンスへとキュー表アフィニティが任意に割り当てられるため、キュー表にアクセスするアプリケーションと高負荷のキューを監視するキュー・モニター・プロセスの間で ping が実行される可能性があります。以降の章「機能性の考慮事項」にある推奨事項「DBMS_PIPE の代わりに Advanced Queuing（AQ）または Java Messaging Service（JMS）を使用する」も参照してください。

高可用性の考慮事項

アプリケーションとデータベースのアーキテクトが Oracle RAC を使用する重要な理由は、アプリケーションの高可用性（HA）です。複数のインスタンスを介して実現されるデータベース・サービスの可用性により、停止に対するリジリエンスをアプリケーションに持たせ、計画停止と計画外停止から透過的に保護されます。クラスタ環境でインスタンスまたはノードの障害が発生しても、残りのインスタンスからデータベース・サービスが提供されると期待されます。

トランザクション・ガードおよびアプリケーション・コンティニュイティ（AC）

Oracle Database 12.2 以降、トランザクション・ガード機能とアプリケーション・コンティニュイティ（AC）機能を使用できるようになりました。

トランザクション・ガードは、計画停止および計画外停止の場合に、最大 1 回の実行でアプリケーションが使用する汎用ツールです。アプリケーションは、論理トランザクション ID を使用して、停止に続くデータベース・セッションでの最後のオープン・トランザクションの結果を判定します。トランザクション・ガードを使用しないと、停止後に操作の再実行を試みるアプリケーションにより、トランザクションが重複してコミットされて論理的破損が発生する可能性があります。

アプリケーション・コンティニュイティは、リカバリ可能なエラーによってデータベース・セッションが使用不可となった後に、データベースに対するリクエストの再実行を中断なしで迅速に有効にする機能です。リクエストには、トランザクション処理と非トランザクション処理が含まれます。リクエストが正常に再実行された後、データベース・セッションが停止した時点からアプリケーションの再開が可能です。振替や航空券予約などで何が起きているのか分からないためにユーザーが疑念を持ったままにされることはなく、中間層のマシンを再起動してログオン・ストームからリカバリする必要もありません。アプリケーション・コンティニュイティにより、アプリケーション開発者がリクエストのリカバリを試みる必要なしに、多くの計画停止や計画外停止をエンドユーザーから隠すことによってエンドユーザー・エクスペリエンスが向上します。アプリケーション・コンティニュイティについて詳しくは、Oracle ホワイト・ペーパーOracle Database 12c によるアプリケーション・コンティニュイティを参照してください。アプリケーション・コンティニュイティのデモを、http://www.oracle.com/technetwork/products/clustering/ac-overview-1967264.html でご覧いただけます。

トランザクション・ガードはアプリケーション・コンティニュイティによって使用され、この機能によって自動的に有効にされますが、個別に有効にすることもできます。トランザクション・ガードによって、アプリケーション・コンティニュイティがトランザクションに複数回適用されないようにします。アプリケーションでアプリケーション・レベルの再実行が実装されている場合は、そのアプリケーションをトランザクション・ガードと統合する必要があります。トランザクション・ガードについて詳しくは、Oracle ホワイト・ペーパーOracle Database 12c のトランザクション・ガードを参照してください。

注：トランザクション・ガードを使用しない場合は、インスタンス障害の発生時、実行中のトランザクションが失われないように RAC で保護されることはありません。ただし、そのようなデータベース・サービスはクラスタ内の残りのノード上に永続的に残されます。

RAC データベース・アーキテクチャを意識していれば、一時的な障害を適切で透過的な方法で処理するようにアプリケーションを設計できます。

動的データベース・サービスを使用する

高可用性を透過的に保持するため、サービス名を使用してアプリケーションをデータベースに接続する必要があります。Oracle RAC データベースのすべての高度な機能はサービスに基づいています。

a. データベース管理者は、指定されたアプリケーションに固有のサービスを作成する必要があります。

b. データベース接続文字列または URL（Java アプリケーションの JDBC 接続文字列など）では常に、インスタンス ID（SID）ではなくサービスを使用する必要があります。

標準 Oracle 接続プーリングを使用する

Oracle RAC の高速アプリケーション通知（FAN）機能により、アプリケーションではデータベース・インスタンスの停止イベント時に通知を受け取ることができます。これによりアプリケーションでは、バックエンド停止中に適切に対応でき、アプリケーション開発者はそのようなイベント発生中のアプリケーションの動作をより適切に制御できます。FAN 機能は Oracle のほとんどの接続プールに統合されており、アプリケーションで Oracle RAC の HA 機能を活用できます。アプリケーション・アーキテクトは、自社開発の接続プールを使用せず、代わりに Oracle が提供する接続プールを使用する必要があります。次に例を示します。

- Oracle Universal Connection Pool (UCP) for Java（J2SE ベース・アプリケーション）
- WebLogic ベース・アプリケーションの GridLink データ・ソース
- TAF に統合された OCI 接続の FAN/OCI プール

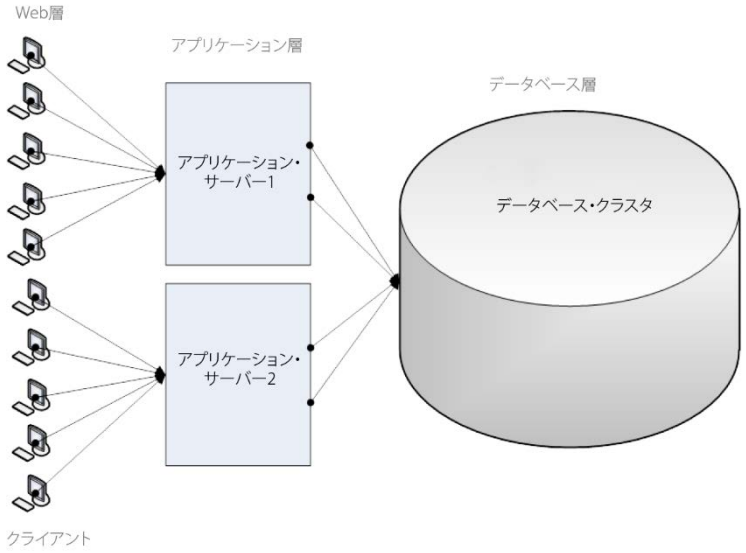

図3：基本接続プーリングの図

図 3 に示すように、接続プールにより、通常はクライアント・システムから Web サーバーへの接続として開始される大量のユーザー・リクエストを、アプリケーション層とデータベース層の間の、より少ない共有接続に効率よく統合して、データ・リクエストを処理します。これにより通常は効率が高まり、データベース層側のユーザー接続にかかる運用および管理のオーバーヘッドが軽減されます。

接続時および実行時ロード・バランシング

通常は、RAC バックエンド・データベースに接続するアプリケーションが、要求されたサービスを提供するすべてのインスタンスへの接続を確立します。Oracle 接続時ロード・バランシングにより、その時点で最小負荷のデータベース・インスタンスを使用してデータベースへの新しいクライアント接続が確実に確立されるようにします。

Oracle 接続プール（UCP および GridLink）では通常、データベースへの永続的な共有接続セットが確立され、一時的なアプリケーション接続リクエストが処理されます。Oracle 接続プールには実行時ロード・バランシング機能があり、この機能により、アプリケーション処理が続行されるのに従ってバックエンド・インスタンスに負荷が均等にかかるようになります。アプリケーションがデータベースへの接続を要求した場合は、通常、事前に確立されている接続がプールから渡されます。実行時ロード・バランシング機能が有効にされた Oracle 接続プールでは、RAC インスタンスからの定期通知として送信されるロード・メトリック情報を使用して、新しいアプリケーション接続リクエストを受け入れる事前に確立されている接続の適性を判別します。これにより、RAC インスタンス間および高効率のクラスタ化されたデータベース環境において、データベース・ワークロードの分散処理で確実にリアルタイム・ロード・バランシングが実行されます。

またアプリケーションは、必要とされる時間だけ接続を開いたままにし、できるだけ早く開放する（閉じる）（または、処理を完了する）ように設計する必要があります。これにより、接続操作と管理に関連した不要なオーバーヘッドを解消します。

透過的接続再試行のためのアプリケーションをビルドする

上記のとおり、接続プール、AC、および他の機能によって透過的な接続の再試行が支援されます。このセクションで説明されるアプローチは主に、以前に名前を挙げて説明した機能を使用できない場合に必要となります。RAC バックエンド・データベースで実行されているアプリケーションは、接続障害を透過的に処理するように設計する必要があります。これには、再試行して接続を確立し、アプリケーション・トランザクションを続行する必要があります。たとえば、JDBC 接続を使用する場合、アプリケーションは SQL_RECOVERABLE_EXCEPTION 例外を捕捉し、残された稼働データベース・インスタンスと新しい接続が確立されると処理中のトランザクションを再試行します。

アプリケーションは、接続を確立すると、その接続を接続プールに戻すまでに 1 つ以上のトランザクションで使用する場合があります。インスタンス停止が発生すると、アプリケーションは例外を受け取ります。例外が発生するとアプリケーションは、アクティブなトランザクションの途中や、トランザクションとトランザクションの間に、短時間のアイドル状態になる場合があります。アプリケーションがトランザクションの途中でない場合、再試行は簡単です。トランザクションの途中で例外を受け取った場合は、再試行に多くのことが関係することになり、アプリケーション・コードでこの状況を適切に処理する必要があります。たとえば、以下に示す、再試行機能を使用して透過的な接続を維持する一般的なアプリケーションの典型的なアプリケーション・ロジックを参照してください。

```
事前トランザクション処理を行う（存在する場合）
WHILE 指定されたエントリの数 > 0
```

```
TRY
  指定したエントリの数を減らす
  メイン・トランザクション処理を行う（おそらく、別のメソッドにバッチ処理）
     IF トランザクションが正常に完了 THEN
        トランザクションの後処理を行う
     ENDIF
  CATCH SQLException
     IF 接続が引き続き有効で使用可能 THEN
        例外の原因は db 停止ではないため、適切に処理するか、コール元に例外を戻す
     ELSE
        接続を閉じる
        新しい正常な接続を取得する
     IF 正常な接続を受け取った THEN
        再試行して while ループに戻りトランザクションを再開する
     ELSE
        例外をコール元に返す
  ENDIF
ENDTRY
ENDWHILE
```

リスト 2：接続再試行のためのアプリケーション疑似ロジック

FAN API を使用してアプリケーションを統合する

Oracle では、API を発行してアプリケーションで FAN 通知を受け取ります。これらの API アプリケーションを使用して、Oracle RAC データベース環境での停止に対処し、アプリケーション接続を管理し、より高度な動作を自動的にアプリケーションに組込むことができます。アプリケーションでの FAN API の統合について詳しくは、Oracle Real Application Clusters のドキュメントを参照してください。

機能性の考慮事項

機能の観点から見ると、ほとんどのアプリケーションは RAC データベースにデプロイされるときにシームレスに動作します。アプリケーションがシングル・インスタンスまたは Oracle RAC 環境のどちらで動作する場合でも、アプリケーションの機能に変更を加える必要がないようにする必要があります。それでも、Oracle RAC などの複数ノード・データベース・アーキテクチャでは、アプリケーションの設計および開発の際に特定の点について考慮する必要があります。

DBMS_PIPE の代わりに Advanced Queuing（AQ）または Java Messaging Service（JMS）を使用する

アプリケーションで Oracle データベースの DBMS_PIPE 機能を使用する場合は、メッセージのプロデューサとコンシューマの両方が同じインスタンスに接続されるようにする必要があります。これは、シングル・インスタンスのデータベース・アーキテクチャで問題となることはありませんが、RAC データベース環境では重要です。さらにアプリケーションでは、DBMS_PIPE の代わりに Advanced Queuing や Java Messaging Service などのメッセージング・メカニズムを使用することが推奨されています。Oracle Database 12c では、JMS Sharded Queue が導入され、データベース・キューイングの実装が最適化されています。Sharded Queue は、複数の物理キューに透過的にパーティション化され、システムによって自動的に保守される単一の論理キューです。JMS Sharded

Queue について詳しくは、『Oracle Advanced Queuing ユーザーズ・ガイド』と Oracle ホワイト・ペーパー（http://www.oracle.com/technetwork/database/jms-wp-2533807.pdf ）を参照してください。

データベース内からの外部ファイルへのアクセスで共有ストレージを使用する

アプリケーションがデータベースのストアド・プロシージャ（RAC データベースの任意のインスタンスから実行される可能性があります）を使用してデータベース内から外部ファイルにアクセスする場合に重要なのは、データベース・サービスを実行可能なすべての RAC インスタンスからそのファイルへのアクセスが可能になっている必要があることです。たとえば、そのような外部ファイルは、RAC データベース・クラスタのすべてのノードからアクセス可能な共有ストレージでホストされている可能性があります。共有ストレージでは、アプリケーションでの必要に応じて、適切なレベルでファイル・アクセスの並行性制御が行われる必要があります。多くの状況において、Oracle ASM Clustered File System（ACFS）は、そのような共有ストレージを実装する場合の最適な選択肢です。

外部表を使用するアプリケーションで共有ストレージを使用する

外部ファイルの場合と同様、外部表を使用するアプリケーションの場合は、外部表ファイルが、アプリケーションを実行することができる RAC データベース・クラスタのすべてのノードからアクセス可能になっている必要があります。

システム・ビューを使用するアプリケーションとスクリプトを GV$ビューに切り替える

v$データベース・ビューを使用するアプリケーションとデータベース管理スクリプトでは、RAC データベース環境の対応するビューはすべてのインスタンスからの情報をすべて表示する GV$ビューであることを認識して記述する必要があります。たとえば、以下に示すシングル・インスタンス環境のクエリの結果と対応するクエリ、および RAC データベース環境からの結果を参照してください。

シングル・インスタンス		Real Application Clusters	
SQL> select instance_name, host_name from v$instance;		SQL> select instance_name, host_name from gv$instance;	
INSTANCE_NAME	**HOST_NAME**	**INSTANCE_NAME**	**HOST_NAME**
rsdb1	rwsoda404c1n1	rsdb1	rwsoda404c1n1
		rsdb2	rwsoda404c1n2

表1 シングル・インスタンス環境とOracle RACデータベース環境でのクエリ

DBMS_JOB パッケージを使用するアプリケーション

アプリケーションでは、スケジュールされた時刻に使用可能なインスタンスへとジョブが確実に割り当てられるようにするため、DBMS_JOB パッケージの代わりに DBMS_SCHEDULER パッケージを使用することをお勧めします。DBMS_JOB パッケージを使用し続ける必要があるアプリケーションの場合は、ジョブに必要なリソースが確実に使用可能であり、データベースのすべてのインスタンス（サービスが実行されていない可能性があるインスタンスを含む）から確実にアクセス可能であることが必要です。ジョブを特定のインスタンスから実行する必要がある場合は、インスタンス・アフィニティ機能を使用してジョブを特定のインスタンスにバインドすることもできます。

Oracle RACデータベース・アプリケーション開発者用チェックリスト

表 2 に、このホワイト・ペーパーで提供される情報をユーザーが使用しやすいようにチェックリスト形式でまとめたものを示します。

Oracle RAC データベース・アプリケーション開発者用チェックリスト	
✓	スケーラビリティを念頭に置いてアプリケーションを設計する
✓	ホットスポットを最小限に抑えるか排除するためのスキーマを設計する
✓	アプリケーションのスケーラビリティをテストする
✓	XA アプリケーションで分散トランザクション処理（DTP）サービスを使用する
✓	RAC データベースで自動セグメント領域管理（ASSM）を使用する
✓	データベース・シーケンスを最適に使用する
✓	アドバンスト・キューイングを最適に使用する
✓	動的データベース・サービスを使用する
✓	標準 Oracle 接続プーリングを使用する
✓	接続時および実行時ロード・バランシング
✓	透過的接続再試行のためのアプリケーションをビルドする
✓	FAN API を使用してアプリケーションを統合する
✓	DBMS_PIPE の代わりに Advanced Queuing（AQ）または Java Messaging Service（JMS）を使用する
✓	データベース内からの外部ファイルへのアクセスで共有ストレージを使用する
✓	外部表機能を使用するアプリケーションで共有ストレージを使用する
✓	システム・ビューを使用するアプリケーションとスクリプトの GV$ビューに切り替える
✓	DBMS_JOB パッケージの代わりに DBMS_SCHEDULER パッケージを使用する

表2：Oracle RACデータベース・アプリケーション開発者用チェックリスト

アプリケーションのチューニング用ツール

スケーラビリティの問題をすばやく識別し、アプリケーションのスケーラビリティをチューニングするための主要なツールは、Oracle Enterprise Manager Cloud Control、Oracle Tuning Pack、および Oracle Load Testing（Oracle Application Testing Suite（ATS）のコンポーネント）です。これらのツールについて詳しくは、関連する Oracle のドキュメントを参照してください。

まとめ

RAC は、アプリケーションの高可用でスケーラブルなデータベース基盤として使用できます。非 RAC データベース環境でスケーラビリティに優れるアプリケーションは、一般に RAC 環境でもスケーラビリティを持ちます。しかし、RAC の高可用性機能を十分に活用してその恩恵を受け、障害に対してアプリケーションがリジリエンスを持つようにするには、アプリケーション・アーキテクチャの設計で RAC を意識する必要があります。さらに、複数インスタンス RAC データベース・アーキテクチャでは、アプリケーション・アーキテクト、システム管理者、およびアプリケーション開発者が意識する必要があります。障害発生時にアプリケーションの動作が透過的に行われるようにするには、データベースの外部のリソースに継続的にアクセスできるようにしておく必要があります。このホワイト・ペーパーには、アプリケーションを設計開発する場合の重要な考慮事項がまとめられており、Oracle RAC データベース・アーキテクチャの利点を最大限に活用するために役立てることができます。

参考資料

1. Oracle Enterprise Manager 13c Cloud Control データ・シート
http://www.oracle.com/technetwork/jp/database/manageability/ds-tuning-pack-db12c-1964661-ja.pdf

2. Oracle Real Application Testing データ・シート
http://www.oracle.com/technetwork/jp/database/manageability/real-application-testing-ds-12c-1964674-ja.pdf

3. MOS Note:『RAC and Oracle Clusterware Best Practices and Starter Kit (Platform Independent)（文書 ID 810394.1）』

4. MOS Note:『RAC: よくある質問（文書 ID 220970.1）』

5. OTN：『Using Basic Database Functionality for Data Warehousing』
http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/10g/r2/prod/bidw/bdf/bdf_otn.htm#t5a

6. Oracle Database VLDB およびパーティショニング・ガイド 12c リリース 1（12.1）、E41057–11

http://www.oracle.com/technetwork/jp/oem/app-test/index.html

7. Oracle Load Testing データ・シート

http://www.oracle.com/technetwork/jp/oem/app-quality-mgmt/index.html

8. Oracle Database 12c Real Application Testing Overview ホワイト・ペーパー
http://www.oracle.com/technetwork/database/manageability/real-application-testing-wp-12c-1896131.pdf

9. Using Oracle Database Cloud - Database as a Service
https://docs.oracle.com/cloud/latest/dbcs_dbaas/CSDBI/GUID-EB0BB703-B7D0-4C3C-B40B-B77D3F08127A.htm#CSDBI-GUID-EB0BB703-B7D0-4C3C-B40B-B77D3F08127A

10. XA および Oracle で制御する分散トランザクション ホワイト・ペーパー
http://www.oracle.com/technetwork/jp/products/clustering/overview/distributed-transactions-

and-xa-163941-ja.pdf

11. Oracle Database 12c によるアプリケーション・コンティニュイティホワイト・ペーパー
http://www.oracle.com/technetwork/jp/database/options/clustering/application-continuity-wp-12c-1966213-ja.pdf

12. Transaction Guard with Oracle Database 12c Hiding Unplanned Outages ホワイト・ペーパー
http://www.oracle.com/technetwork/jp/database/database-cloud/private/transaction-guard-wp-12c-1966209-ja.pdf

13. Database JDBC Developer's Guide, Oracle RAC Fast Application Notification
http://docs.oracle.com/cd/E57425_01/121/JJDBC/apxracfan.htm#CDCBIDHF

14. Oracle Database 12c: JMS Sharded Queues ホワイト・ペーパー
http://www.oracle.com/technetwork/database/jms-wp-2533807.pdf

15. Oracle Enterprise Manager Cloud Control Documentation 12c
http://docs.oracle.com/cd/E24628_01/index.htm

16. Oracle Application Testing Suite（OTN ページ）
http://www.oracle.com/technetwork/jp/oem/app-test/index.html

17. MOS Note: Automatic Space Segment Management in RAC Environments（文書 ID 180608.1）

18. MOS Note: Caching Oracle Sequences（文書 ID 62002.1）

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

海外からのお問い合わせ窓口
電話：+1.650.506.7000
ファクシミリ：+1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Oracle Real Application Clusters（RAC）のアプリケーション開発のベスト・プラクティス、開発者用チェックリスト
2016 年 7 月
著者：Hagen Herbst（RAC パック）、Ravi Sharma（RAC パック）（Pradeep Bhat による旧版を改作）
共著者：Sanjay Singh（RAC パック）、Markus Michalewicz（RAC 製品管理）