Enterprise Postgres のスケールアウト機能の技術を解説

システム構成の方式には、シェアード・エブリシング方式とシェアード・ナッシング方式があります。この記事では、スケールアウト（水平スケール）の基礎的なアーキテクチャーとして、シェアード・エブリシングとシェアード・ナッシングの 2 つの方式の特徴や違いを考察します。
また、本記事の最後では、「FUJITSU Software Enterprise Postgres 14 SP1（以降、 Enterprise Postgres 14 SP1）」がスケールアウト機能にシェアード・ナッシング方式を採用した狙いと、今後より多くの要件に対応できるよう独自に追加した機能について解説します。

なお、データベースにおけるスケールアウトの必要性、スケールアウト機能の概要や使い方については、以下の記事がお薦めです。

- 【スケールアウトとは】言葉の意味やデータベースでの必要性を解説
- Enterprise Postgres のスケールアウト機能を紹介 ～ Active-Active 構成を実現 ～
- Enterprise Postgres 14 SP1 のスケールアウト機能の動作を検証

技術解説の背景

近年、仮想化やクラウド活用に大きな進展があります。その根底には、CPU、メモリーなどの資源が「所有」から「利用」へ変化し、さらに、ビジネスの大きさに応じてフレキシブルに活用できるようになってきたことが挙げられます。経営においても、データドリブンの考え方が浸透してきており、それに応じて、開発手法やスケールアウト機能に求められる要件も変化してきました。ますます、データ自体の価値が高まり、データの整備と活用が進んできています。

このような背景におけるデータベースの選択は重要な位置づけとなります。そのため、改めて、スケールアウト方式のアーキテクチャーから考察し直す必要があります。つまり、データドリブン経営を想定すると、今あるデータから新しいサービスを追加、さらに追加したサービスで記録・収集できるデータから、また新しいサービスを追加していく改善のスパイラルを回す必要があります。この時、サービスからデータを最初に記録するデータベースシステムは、既存のサービスに影響を与えず安全にデータ業務を追加できる必要があります。それを実現するためには、マルチテナント型で多種データに対応できるようなスケールアウト基盤が必要になります。また、その実行基盤としても、多種多様な業務に対応できるよう、オンプレミス、クラウドを問わず利用できることも重要になります。

スケールアウトを実現する 2 つの方式について

ここでは、現在主流である「シェアード・エブリシング方式」と、今後主流になるであろう「シェアード・ナッシング方式」について、近年のアーキテクチャーの動向を踏まえて、それぞれの特徴について説明します。

シェアード・エブリシング方式とは

シェアード・エブリシング方式とは、システム構成の方式の 1 つで、複数のサーバーにまたがってディスクを共有する方式のことです。以降では、データベースを例としてシェアード・エブリシング方式の特徴を解説します。

特徴

シェアード・エブリシング方式を採用したデータベースでは、データベースのデータセットは共有ストレージ(シェアードディスク、共有ディスク)に保管され、複数のデータノードからアクセスされる構成です。各データノードでは独立して SQL を処理します。各データノードのデータベースキャッシュには、共有ストレージ内の全データを対象として、すべて同じデータを持ちます。そのため、アプリケーションはどのデータノードからでも、同じデータにアクセスできます。
ただし、1 つのデータノードでデータが更新されると、データベースキャッシュの状態は、一時的に他のデータノードとの不整合が発生した状態になります。そのため、更新されたデータを他のデータノードにも反映することで不整合を解決します（以降、キャッシュ整合性解決と表現します）。このキャッシュ整合性解決は、インターコネクトを利用して各データノード間での分散ロックとデータ転送が行われるため、性能上のボトルネックになる可能性があります。
また、この方式は、データノードを複数台構成することで、拡張性と可用性を同時に実現できます。

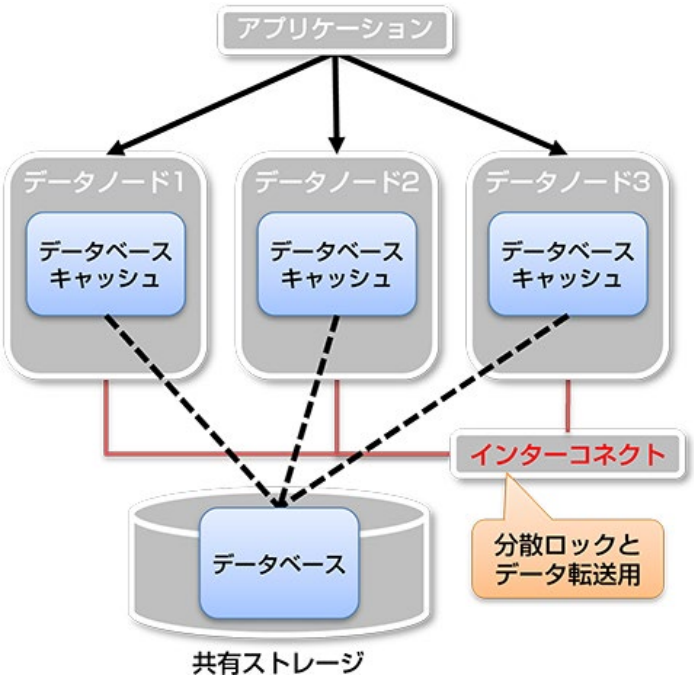

図 1：シェアード・エブリシング方式

メリット

シェアード・エブリシング方式のメリットとして、以下が挙げられます。

- 性能特性：データ配置の設計が必須ではないため簡単にシステム性能を向上できる
- 拡張：利便性としてのスケールのし易さ、負荷分散による資産の有効活用に優れている
- 可用性：データノード（サーバー）を複数台配置することで可用性を簡単に向上できる

デメリット

シェアード・エブリシング方式のデメリットとして、以下が挙げられます。

- 性能特性：キャッシュ整合性解決が実施された場合、システム性能が落ちる恐れがある
- 拡張：データノード数を増やしていくほど、チューニングが必要不可欠になる
- 可用性：ストレージには別途ハードウェアやソフトウェアで冗長化が必要である

シェアード・ナッシング方式とは

シェアード・ナッシング方式とは、システム構成の方式の 1 つで、サーバーごとにディスクを独立させる方式のことです。以降では、データベースを例としてシェアード・ナッシング方式の特徴を解説します。

特徴

シェアード・ナッシング方式を採用したデータベースでは、データベースのデータセットを期間や地域などの単位で分割し、各データノードのローカルストレージに配置する構成です。この方式には管理ノードがあり、データセット全体の管理や、複数のデータノードに効率的にアクセスできるよう制御します。各データノードでは独立して SQL を処理します。各データノードのデータベースキャッシュには、分割されたデータを持ちます。そのため、アプリケーションがデータにアクセスするためには、管理ノードを経由するか、あるいは、直接データノードに接続します。
また、この方式は、他のデータノードの影響をほとんど受けないため、各データノードで、業務（トランザクション）を並列実行させることに適しています。インターコネクトにおいては、テーブルデータ、SQL、実行計画などの通信に使用されるため、シェアード・エブリシング方式のインターコネクトよりも、通信量は少なく済みます。
また、この方式で可用性を実現するためには、管理ノードとデータノードをそれぞれ冗長化します。

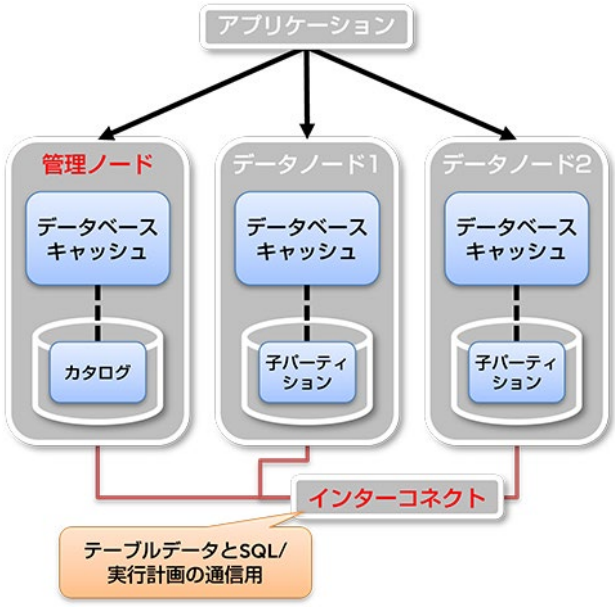

図 2：シェアード・ナッシング方式

メリット

シェアード・ナッシング方式のメリットとして、以下が挙げられます。

- 性能特性：データノード間の影響がほとんどなく、キャッシュ整合性解決の考慮が不要である
- 拡張：データを分散配置することで、データノード間の更新排他による影響が少なくできる
- 可用性：特別な共有ストレージや高速なインターコネクトなどを必要とせず、一般的なクラウド上でも採用できる

デメリット

シェアード・ナッシング方式のデメリットとして、以下が挙げられます。

- 性能特性：アプリケーションの処理とデータ分散が適切に行われるよう、データ配置の設計が必要である
- 拡張：データ配置の再検討が必要である
- 可用性：データノードを個別に冗長化する必要がある

スケールアウト方式の観点ごとの比較

スケールアウトに関する 2 つの方式を、スケールアウトの重要な観点である「性能特性」、「拡張性」、「可用性」について比較し、技術的側面での違いを明確にしていきます。

性能特性の比較

ここでは、それぞれのスケールアウト方式の構造を基に、スケールアウト時のスケール効果（性能特性）の違いについて述べ、そこから判断される適切なワークロードについても示します。

- **シェアード・エブリシング方式**

  この方式は、データ配置の設計を行わなくても、ある程度のデータノード数までのスケール効果が期待できます。ただし、データノード数が増えることでキャッシュ整合性解決の負荷が大きくなり、スケール効果が鈍化します。これは、図 3 左側のように、複数のデータノードで同じデータを更新する可能性が高くなることが原因であり、データノード数が増えるほど顕著になります。そのため、スケール効果を出すためには、データブロックを小さくしたり、パーティショニングテーブルを使ったりして、各データノードが別々のデータブロックにアクセスできるようなチューニングが必要になります。
  適切なワークロードとしては、主に参照系トランザクションが多いシステム、あるいは、高多重で短時間トランザクションの用途に適しています。また、各データノードで同じデータを参照するような用途（同時分析、機械学習など）にも適しています。

- **シェアード・ナッシング方式**

  この方式は、主に 1 つのトランザクションが 1 つのデータノードに閉じて処理できるよう、データが分散化されていることを前提とします。そのため、データノードを増やすほど、大きなスケール効果が期待できます。このようなデータセットは、例えば、期間ごと、地域ごとのように分割されているケースであり、既にデータベースで利用されるデータにおいても、そのようなケースは多いと思われます。
  また、この方式は、データノード間の影響がほとんどなく、キャッシュ整合性解決の考慮も不要です。その利点を活かし、データノード毎に異なる業務やデータを配置するような運用にも対応できます。
  適切なワークロードとしては、バッチのような比較的長時間の更新系トランザクションを、各データノードで並列実行させるような用途に適しています。また、銀行業務における支店追加のように、同様な業務を拡張していくような用途にも適しています。

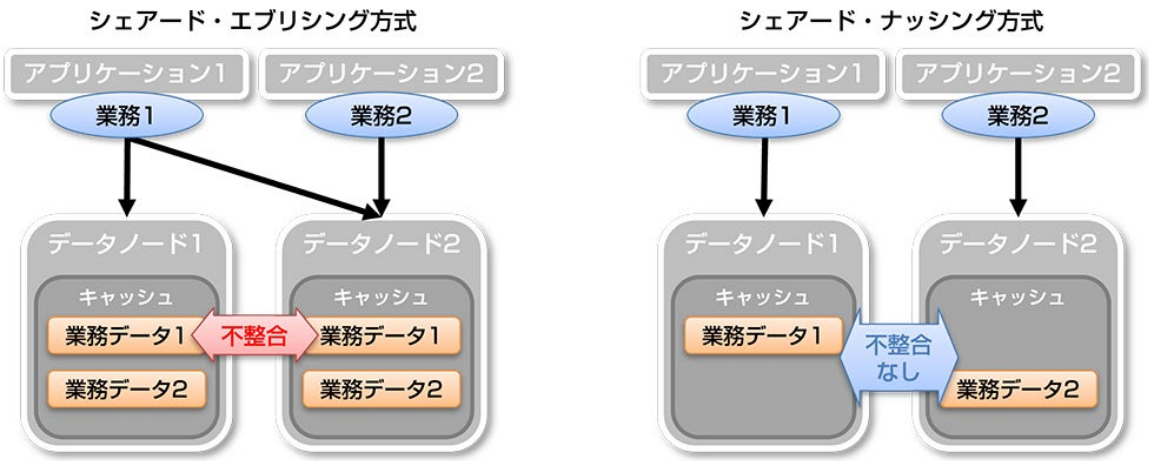

図 3：2 つの方式についてのデータ配置の違いと不整合発生の有無

さらに、通常のデータセットには、各データノードで常に参照されるようなマスターデータが含まれます。シェアード・ナッシング方式でスケールアウトを実現するためには、マスターデータそのものを各データノードへ配布する仕組みを併せ持つ必要があります。これにより、アプリケーションが 1 つのデータノードに閉じてトランザクションが実行できるようになります。

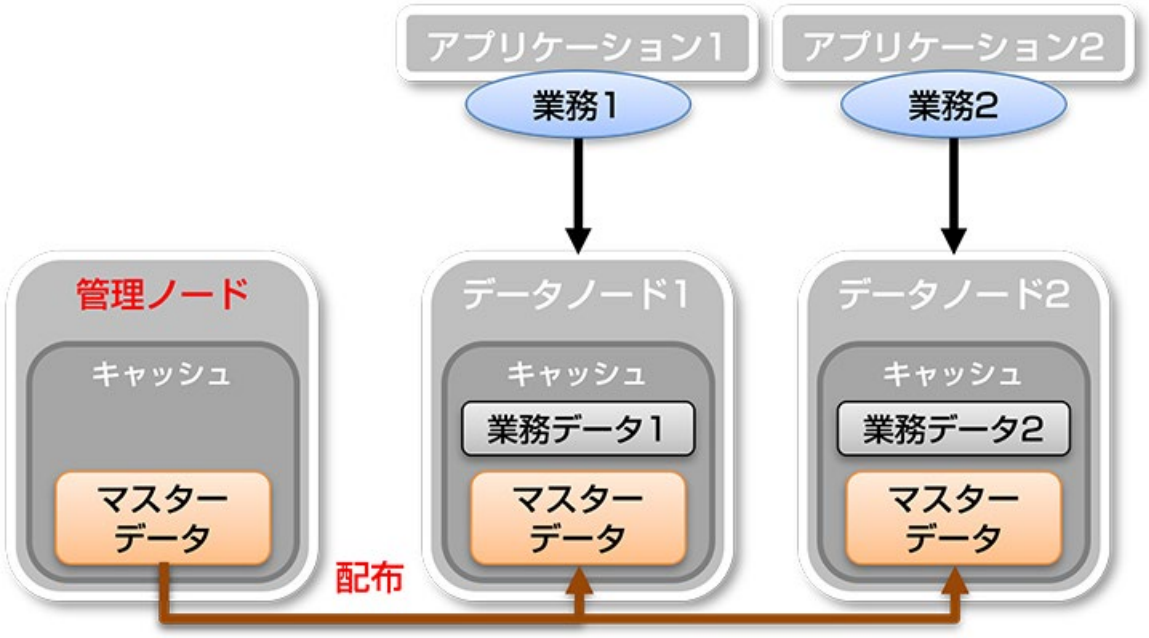

図 4：シェアード・ナッシング方式のマスターデータの配布

どちらの方式においても、スケールアウトの「性能特性」を高めるポイントは、データの分散化であることが分かりました。ワークロードについては、それぞれの方式ごとに適切なワークロードがあることも分かります。以下に、観点ごとの違いをまとめます。

比較項目	シェアード・エブリシング方式	シェアード・ナッシング方式
スケールアウトによる効果	キャッシュ整合性解決の負荷が高くならない範囲であればデータノード数分のスケール効果があります。キャッシュ整合性解決の負荷を軽減するためには、アプリケーションの処	アプリケーションの処理とデータ分散が適切に行われるよう、データ配置設計がなされて

比較項目	シェアード・エブリシング方式	シェアード・ナッシング方式
	理とデータ分散が適切に行われるようチューニングが必要です。	いることを前提に、データノードの台数に見合ったスケール効果があります。
スケールアウトを阻害する要因	キャッシュ整合性解決の負荷が増えるほどスケール効果が出ません。	複数データノードにまたがるアクセスがあるとスケール効果が出ません。
適切なワークロード	参照系トランザクション、あるいは、高多重で短時間トランザクション	バッチのような比較的長時間の更新系トランザクション、あるいは、複数ノードでのトランザクションの並列実行

参考

シェアード・エブリシング方式において性能を上げるには、以下のチューニングが必要になります。

- 複数のデータノードが同じデータをアクセスしないよう、データブロックを小さくしたり、データブロック内の空き領域の比率を大きくしたりする
- データセットをハッシュパーティショニングで構成し、格納場所を分散化する

また、データノードが複数存在することで、シーケンスの番号が競合しないよう対処する必要もあります。これは、シェアード・ナッシング方式においても同様です。なお、シェアード・エブリシング方式で行われた性能を上げるためのチューニングや設計については、シェアード・ナッシング方式へのデータベース移行の際に引き継ぐことができます。

拡張性の比較

ここでは、それぞれのスケールアウト方式の構造から、スケールアウトの拡張性についての違いを示します。
なお、比較を行う上で、データ処理の実態について考慮する必要があります。近年では、センサーデータを活用した健康情報、家電、スマートスピーカーなどの普及により IoT の浸透によるデータ爆発の時代を迎えておりデータの大規模化が進んでいます。企業内においては、数多くの業務システムの集約化と運用の簡素化が進んでいます。そして、それらの実行環境としてクラウド利用が進んでいることが挙げられます。これらの要因を考慮しながら比較します。

- **シェアード・エブリシング方式**

  利便性としてのスケールのし易さ、負荷分散による資産の有効活用に優れており、急な業務負荷に対応できます。なお、キャッシュ整合性解決の影響があるため、スケールには一定の限界があります。また、データセット全体がキャッシュ対象になることから、シングルテナント大容量データ型のスケールアウトに適しています。

- **シェアード・ナッシング方式**

  期間、地域、業務、支店などの単位でデータを分散配置することで、データノード間の更新排他による影響が少なく構成できます。その利点を活かし、最初はスモールスタートでいくつかの業務を集約し、徐々に新たな業務をデータノード追加により増やしていくような、マルチテナント多種データ型のスケールアウトに適しています。この方式では、多種データ利用やクラウド展開がし易いことから、現在では適用範囲が広がっています。

データの大規模化と業務集約に向けた「拡張性」について比較すると、マルチテナント多種データ型のスケールアウトに適しているシェアード・ナッシング方式が優位であることが分かります。適用先に一般的なクラウドや仮想環境を選択できることも利点になります。以下に、観点ごとの違いをまとめます。

比較項目	シェアード・エブリシング方式	シェアード・ナッシング方式
拡張の容易性（スモールスタートからの拡張）	業務中の急な処理能力不足に際しても、データはそのままで、データノードを追加することでスループットが向上できます（データノード数を増やしていくほど、チューニングの必要性が高まります）。	データノード追加で新たな業務追加に対応できます。その際、業務の大きさに応じて、柔軟に CPU やメモリーなどの資源量が決められます。
データノードのキャッシュの対象	全データノードが、同じデータセットをキャッシュ対象にします。	各データノードで保持するデータのみがキャッシュ対象になります。
既存業務の負荷が増えた時の対応	データノードの負荷を見ながら、データノードを必要分追加し、均等に処理を振り分けることができます（負荷分散機能）。	データノード単位に、業務に見合った CPU、メモリーなどの資源を割り当てられます（スケールアップ）。また、業務負荷の上がった特定のデータノードを対象に、参照系のデータノードを追加して参照業務を拡張できます。

可用性の比較

ここでは、それぞれのスケールアウト方式の構造から、スケールアウトの可用性についての違いを示します。

- **シェアード・エブリシング方式**

  データベース構成上にデータノード（サーバー）を複数台配置することで可用性を向上できる仕組みであり、拡張性と可用性の両方を併せ持つメリットの大きな構成です。しかし、システムアーキテクチャー上、ストレージには別途ハードウェアやソフトウェアで冗長化が必要になります。そのため、クラウド展開においては、クラウド上に共有ストレージや、高速なインターコネクトなどの特別な機構が必要になります。

- **シェアード・ナッシング方式**

  スケールアウトを構成するデータノードや管理ノードを個別に冗長化することで可用性を担保します。また、クラウド展開においては、特別な共有ストレージや高速なインターコネクトなどを必要とせず、一般的なクラウド上でも採用が可能な点は、大きなメリットになると言えます。

「可用性」についての比較では、シェアード・エブリシング方式が優位であることが分かります。しかし、クラウド展開においては、シェアード・ナッシング方式が優位であることが分かりました。以下に、観点ごとの違いをまとめます。

比較項目	シェアード・エブリシング方式	シェアード・ナッシング方式
サーバー（VM）の可用性	どのデータノードが故障しても、生存ノードですべてのデータにアクセスできます。	データが各データノードに分散しているため、各データノード、および、管理ノードをそれぞれ冗長化する必要があります。
ストレージの可用性	共有ストレージ障害に対しては、別途ハードウェアやソフトウェアでの冗長化が必要です。	データノード上のローカルストレージを利用するため、データノードを冗長化することで、ストレージも冗長化されます。
フェイルオーバー時の可用性	故障したノードの処理を他のノードが引き継いで、縮退運用で業務を継続できます。	データノードをホットスタンバイで冗長化しておくことにより、故障したノードを切換えて業務を継続することができます。
クラウド上での可用性	クラウド上でも、共有ストレージや、高速のインターコネクトのような特別な機構が必要になります。	特別なハードウェア、ソフトウェアを必要としないため、多くのタイプのクラウドで冗長化構成が作れます。

以上、「性能特性」、「拡張性」、「可用性」の観点で、2 つのスケールアウト方式について比較を行いました。どちらの方式にも固有の優位性があることが分かりました。しかし、今後のクラウド展開を考慮すると、特別な機構を必要とせず、一般的なクラウド上で実現が可能なシェアード・ナッシング方式に優位性があると言えます。また、データドリブン経営に向けて価値のあるデータを随時追加しながら活用していく上でも、マルチテナント多種データ型でデータベースをスケールアウトしていける、シェアード・ナッシング方式が優位と考えます。
これらの判断により、クラウド時代において、より多くの業務に対応し価値を提供できるよう、Enterprise Postgres 14 SP1 のスケールアウト機能では、シェアード・ナッシング方式を採用しました。

Enterprise Postgres のスケールアウト機能のメリット

Enterprise Postgres 14 SP1 のスケールアウト機能は、シェアード・ナッシング方式をベースとし、新規業務追加により拡張していくような用途に適した構成を採用しました。さらに、エンタープライズ利用を想定して、より多くの用途で利用できるよう以下の 4 つの独自機能も搭載しています。これらの機能により、より多くのワークロードに適用できるようになります。
概要については、「Enterprise Postgres のスケールアウト機能を紹介 ～ Active-Active 構成を実現 ～」を参照してください。ここでは詳細を説明します。

シャード名指定による 2 通りのアクセス方法

Enterprise Postgres 14 SP1 のスケールアウト機能では、データノード上に、複数のテーブル空間のグループとして「シャード」という論理的な箱を用意します。データベースのテーブルやパーティショニングで分割されたパーティションは、各データノードの「シャード」上に配置されます。
アプリケーションからテーブルやパーティションにアクセスするには、データベースサーバーの状態監視や透過的接続を支援するための機能である「Connection Manager」を介して行います。「Connection Manager」は、「中央管理ノード」と連携し、適切なデータノードにルーティングするための、2 通りのアクセス方法を提供します。

①　性能を重視したアクセス

アプリケーションがシャード名を指定して Connection Manager に接続すると、Connection Manager がコネクションを自動でルーティングし、対応するデータノードに直接的に接続されるため、高速なアクセスを可能にします。

②　利便性を重視したアクセス

アプリケーションがシャード名を指定しないで Connection Manager に接続すると、中央管理ノードが SQL 文を解析し、該当するデータノードを特定してコネクションを張り、適切なデータノードにアクセスすることができます。なお、複数のデータノードにアクセスするようなトランザクションを実行する場合も、この方法でアクセスします。その際、内部的に二相コミットによる同期アクセスが行われるため、データの安全性も確保されます。

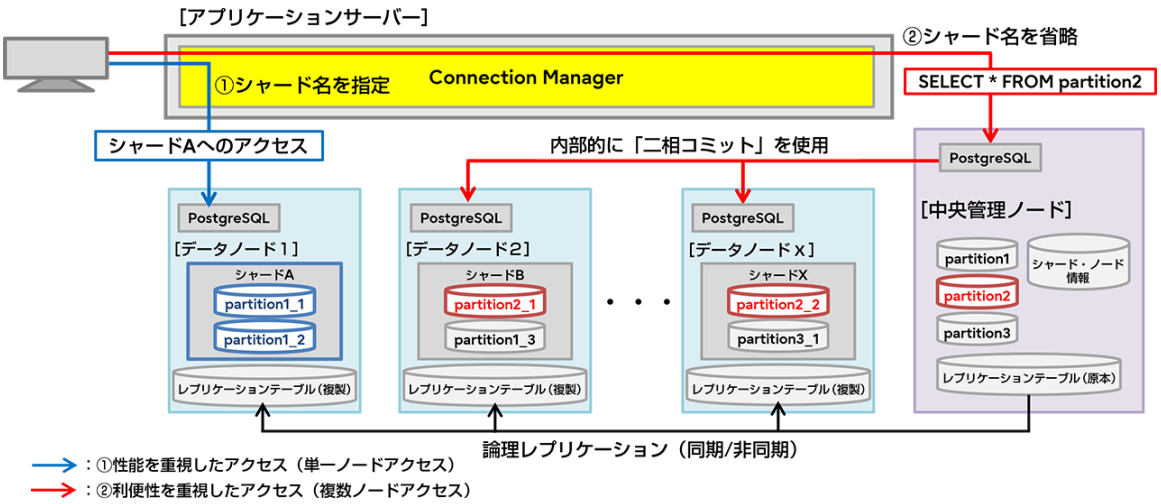

図 5：Enterprise Postgres 14 SP1 のスケールアウト機能のデータ配置とアクセス方法

レプリケーションテーブルによるマスターデータの配布

多くのデータベースには、常にアクセスされるマスターデータ（各データノードで共通となるデータ）があります。例えば、マルチテナントモデルにおいては、商品マスターやユーザーが入力したフォームデータ（主に参照データ）などがあります。このようなマスターデータは常にアクセスされるため、各データノード上に配置することが適切です。
Enterprise Postgres 14 SP1 のスケールアウト機能では、PostgreSQL の論理レプリケーションを利用して、このマスターデータの原本を中央管理ノードに配置し、そこから各データノードへ配布（複製）します（図 5 の「レプリケーションテーブル」の箇所）。なお、マスターデータの更新を行う際には、中央管理ノードに接続して行います。
また、このような構成を作成する際には、中央管理ノードと全データノードでデータオブジェクトを定義する必要があります。これは、データノード数が増えるほど手間が掛かる作業になります。そこで、Enterprise Postgres では、「DDL の一元実行」機能が提供されています。この機能は、中央管理ノードで、以下の例のようにロールやトリガーなどのオブジェクトを作成／変更／削除した際に、自動的にそれらの操作を各データノードにも伝搬させます。この機能を利用することで、定義時の利便性が向上します。

```
-- 例 1: 全データノード対象のオブジェクトの定義（ロールを作成する）
SET pgx_ddl_target_node = 'ALLNODES';
CREATE ROLE role_sc;

-- 例 2:全ノード対象のオブジェクトの変更
SET pgx_ddl_target_node = 'ALLNODES';
ALTER ROLE role_sc LOGIN; -- role_sc は全ノード対象のオブジェクトのため、全ノードに対して実行される

-- 例 3:全ノード対象のオブジェクトの削除
```

```
SET pgx_ddl_target_node = 'ALLNODES';
DROP ROLE role_sc; -- role_sc は全ノード対象のオブジェクトのため、全ノードに対して実行される
```

参照処理の拡張

実際の業務においては、処理能力を超えるアクセスの増加に対応しなければならないケースも想定されます。その多くの場合は参照系のアクセスです。Enterprise Postgres 14 SP1 のスケールアウト機能では、ストリーミングレプリケーションの機構を利用して、データノードの参照レプリカ（複製）を追加することで、個々のデータノードの参照処理を拡張することができます。

図 6 は、データノード 1 の参照処理を拡張した例になります。その際、アプリケーションからの接続文字列に、シャード名と共に「target_session_attrs=prefer-read」を指定することで、参照系の問い合わせは参照レプリカを優先して接続されるようになります。あらかじめ、アプリケーションで target_session_attrs パラメーターを設定しておけば、データノードの負荷が上昇した際に参照レプリカを追加することで、透過的にスループットを向上させることが可能になります。これにより、より多くのワークロードに対応することが可能になります。

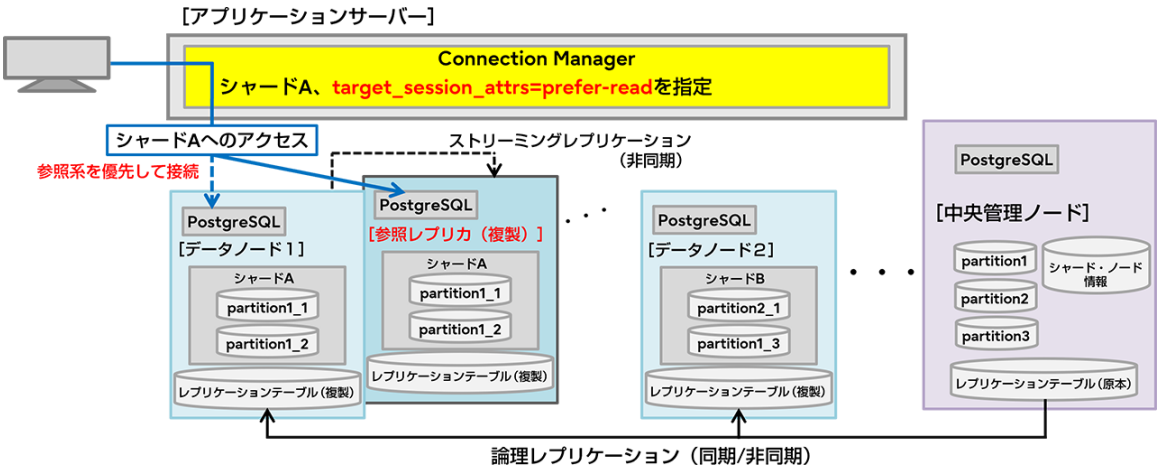

図 6：レプリカ追加による参照処理の拡張

多数のノードをコントロールする高可用性

Enterprise Postgres 14 SP1 のスケールアウト機能では、拡張性と可用性の両方を確保するために、必然的にインスタンス数が多くなり、複雑な構成になります。すると、運用監視や、異常時の切り替え処理などの実装についても複雑になってしまいます。そのため、これら運用に必要な機構についても、製品の機能だけで実現できるようワンストップで提供しています。それを実現する機構は以下のとおりです。

- **Connection Manager（アプリケーションサーバー内、中央管理ノード内、データノード内）**

  シャード名からデータノードへの接続をルーティングする機能（接続ルーティング）を提供します。また、中央管理ノードとデータノード間の接続に利用されます。サーバーやネットワークなどに異常が発生した場合に、中央管理ノードとデータノードをスタンバイに切り替えます。その際に不要になった接続の資源を回収します。

- **Mirroring Controller（中央管理ノード内、データノード内）**

  ノードのプライマリーサーバーとスタンバイサーバーを切り替えます。また、データベースサーバーと Connection Manager の生死監視をします。

- **Mirroring Controller（裁定サーバー内）**

  プライマリーサーバーとスタンバイサーバーが相互の状態を正確に把握できない場合、第三者として異常なデータベースを隔離します。また、データベースサーバーと Connection Manager の生死監視をします。

- **Scale out Controller（裁定サーバー内）**

  Connection Manager と Mirroring Controller の機能を利用して、ノードの生死監視、スプリットブレインの防止およびスタンバイへの自動切り替え機能を提供します。

これらの機構の配置イメージは、図 7 のとおりです。この図では、2 つの VM で構成した例を示しています。なお、裁定サーバーは、同一ネットワーク内において別の用途で利用している VM 上に置くこともできます。

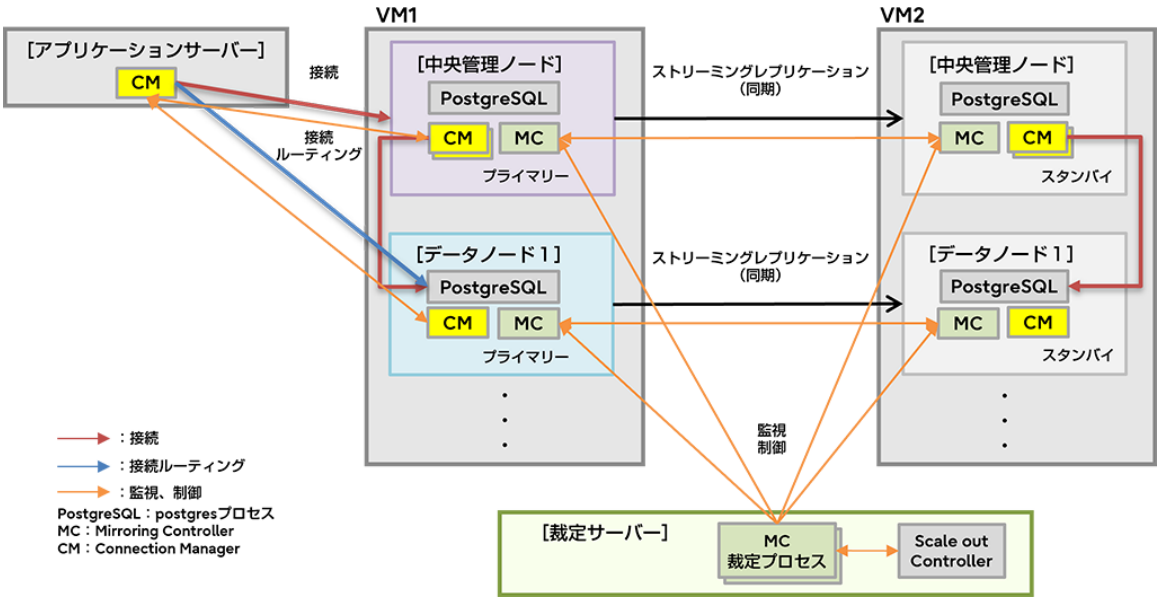

図 7：Enterprise Postgres 14 SP1 のスケールアウト機能の高可用性構成（2 つの VM ノードの例）

Enterprise Postgres では、仮想化やクラウド活用が大きく進む時代において、変化を見極め、これからのビジネスに適合できるよう、スケールアウト機能を実現しました。ぜひ、Enterprise Postgres の活用をご検討ください。お問い合わせは本ページに表示されている「お問い合わせ」ボタンから、または本ページ下部に記載の「本コンテンツに関するお問い合わせ」からご連絡ください。

2022 年 9 月 30 日