データベース設定

Enterprise applications

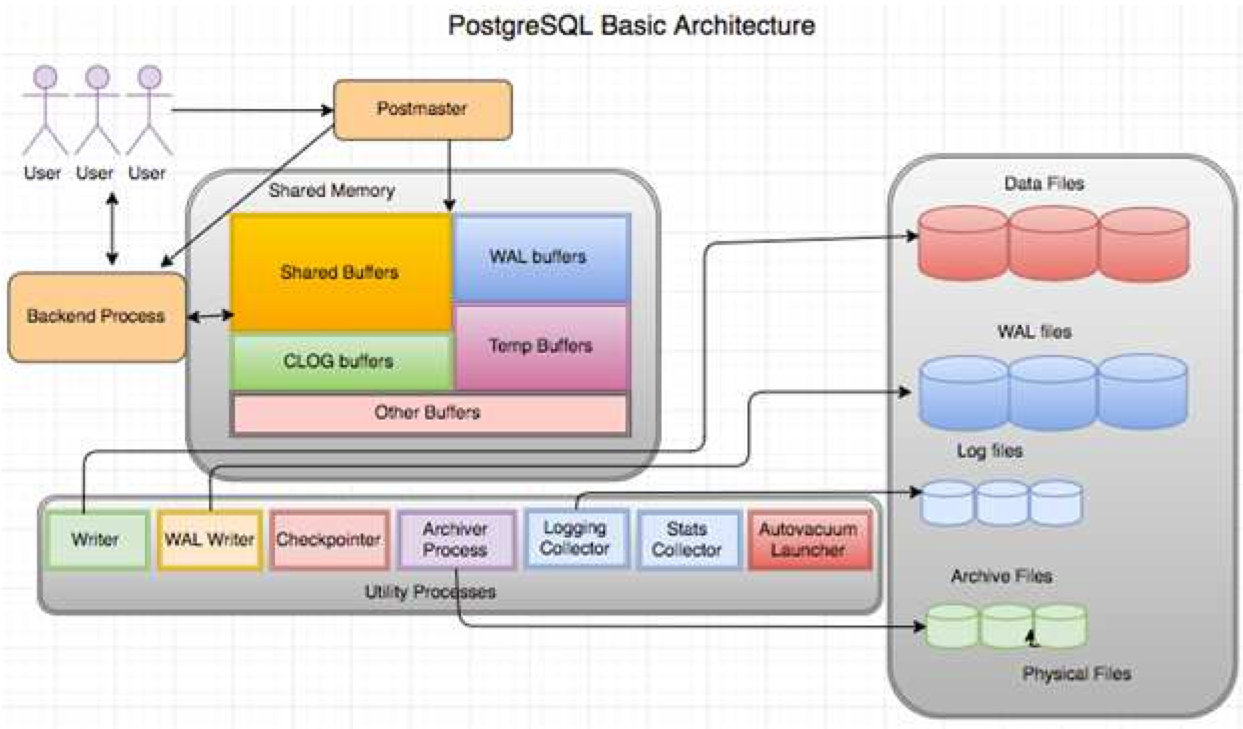
NetApp
February 10, 2026

目次

データベース設定

アーキテクチャ

PostgreSQLは、クライアントとサーバのアーキテクチャに基づいたRDBMSです。PostgreSQLインスタンスはデータベースクラスタと呼ばれ、サーバの集合ではなくデータベースの集合です。



PostgreSQLデータベースには、postmaster、フロントエンド(クライアント)、バックエンドの3つの要素があります。クライアントは、IPプロトコルや接続先データベースなどの情報を含む要求をポストマスターに送信します。postmasterは接続を認証し、さらに通信するためにバックエンドプロセスに渡します。バックエンドプロセスはクエリを実行し、結果を直接フロントエンド（クライアント）に送信します。

PostgreSQLインスタンスは、マルチスレッドモデルではなく、マルチプロセスモデルに基づいています。ジョブごとに複数のプロセスが生成され、各プロセスには独自の機能があります。主なプロセスには、クライアントプロセス、WALライタプロセス、バックグラウンドライタプロセス、およびcheckpointerプロセスが含まれます。

- クライアント（フォアグラウンド）プロセスがPostgreSQLインスタンスに読み取りまたは書き込み要求を送信しても、データを直接ディスクに読み書きすることはありません。最初に、共有バッファとWAL(Write-Ahead Logging)バッファにデータをバッファします。
- WALライタプロセスは、共有バッファとWALバッファの内容を操作してWALログに書き込みます。WALログは通常PostgreSQLのトランザクションログであり、シーケンシャルに書き込まれます。したがって、データベースからの応答時間を短縮するために、PostgreSQLはまずトランザクションログに書き込み、クライアントに確認応答します。
- データベースを整合性のある状態にするために、バックグラウンドライタープロセスは共有バッファにダーティページがないか定期的にチェックします。次に、NetAppボリュームまたはLUNに格納されているデータファイルにデータをフラッシュします。

- checkpointerプロセスも定期的に（バックグラウンドプロセスよりも少ない頻度で）実行され、バッファへの変更を防ぎます。WALライタプロセスに、NetAppディスクに保存されているWALログの末尾にチェックポイントレコードを書き込み、フラッシュするように指示します。また、すべてのダーティページをディスクに書き込み、フラッシュするようにバックグラウンドライタープロセスに通知します。

初期化パラメータ

新しいデータベースクラスタを作成するには、 `initdb` プログラム。A `initdb` スクリプトは、クラスタを定義するデータファイル、システムテーブル、およびテンプレートデータベース（template0およびtemplate1）を作成します。

テンプレートデータベースはストックデータベースを表します。システムテーブル、標準ビュー、関数、およびデータ型の定義が含まれています。 `pgdata` の引数として機能します。 `initdb` データベースクラスタの場所を指定するスクリプト。

PostgreSQLのすべてのデータベースオブジェクトは、それぞれのOIDによって内部的に管理されます。テーブルとインデックスは、個 々 のOIDによっても管理されます。データベースオブジェクトとそれぞれのOIDとの関係は、オブジェクトのタイプに応じて適切なシステムカタログテーブルに格納されます。たとえば、データベースとヒープテーブルのOIDは、 `pg_database` それぞれ`pg_classと`pg_classです。OIDを確認するには、PostgreSQLクライアントでクエリを発行します。

各データベースには、1GBに制限された個別のテーブルとインデックスファイルがあります。各テーブルには、それぞれサフィックス付きの2つのファイルが関連付けられています。 `_fsm` および `_vm`。これらは、フリースペースマップおよび可視性マップと呼ばれます。これらのファイルには空きスペース容量に関する情報が格納され、テーブルファイルの各ページに表示されます。インデックスには個 々 の空き領域マップのみがあり、可視性マップはありません。

。 `pg_xlog/pg_wal` ディレクトリには、先行書き込みログが格納されます。先行書き込みログは、データベースの信頼性とパフォーマンスを向上させるために使用されます。テーブル内の行を更新するたびに、PostgreSQLは先読みログに変更内容を書き込み、その後実際のデータページに変更内容をディスクに書き込みます。。 `pg_xlog` ディレクトリには通常複数のファイルが含まれていますが、initdbは最初のファイルだけを作成します。必要に応じて追加のファイルが追加されます。各xlogファイルの長さは16MBです。

設定

パフォーマンスを向上させるPostgreSQLのチューニング設定がいくつかあります。

最も一般的に使用されるパラメータは次のとおりです。

- `max_connections = <num>`:一度に持つデータベース接続の最大数。このパラメータを使用して、ディスクへのスワップを制限し、パフォーマンスを強制終了します。アプリケーションの要件に応じて、このパラメータを接続プールの設定に合わせて調整することもできます。
- `shared_buffers = <num>`：データベース・サーバのパフォーマンスを向上させる最も簡単な方法最新のほとんどのハードウェアでは、デフォルトはlowです。導入時に、システム上の使用可能なRAMの約25%に設定されます。このパラメータ設定は、特定のデータベースインスタンスでの動作によって異なります。試行錯誤して値を増減しなければならない場合があります。ただし、この値をHighに設定すると、パフォーマンスが低下する可能性があります。
- `effective_cache_size = <num>`:この値は、PostgreSQLのオプティマイザに、PostgreSQLがデータをキャッシュするために使用できるメモリ量を伝え、インデックスを使用するかどうかを判断するのに役立ちます。値を大きくすると、インデックスを使用する可能性が高くなります。このパラメータは、に割

り当てられているメモリの量に設定する必要があります。 `shared_buffers` さらに、使用可能なOSキャッシュの容量も表示されます。多くの場合、この値はシステムメモリ全体の50%を超えています。

- `work_mem = <num>`:このパラメータは、ソート操作およびハッシュテーブルで使用するメモリの量を制御します。アプリケーションで大量のソートを行う場合は、メモリの量を増やす必要があるかもしれませんが、注意が必要です。これはシステム全体のパラメータではなく、操作ごとのパラメータです。複雑なクエリに複数のソート操作が含まれている場合、複数のwork_mem単位のメモリを使用し、複数のバックエンドが同時にこれを実行する可能性があります。このクエリを実行すると'値が大きすぎると'データベース・サーバがスワップされることがよくありますこのオプションは、以前のバージョンのPostgreSQLではsort_memと呼ばれていました。

- `fsync = <boolean> (on or off)`:このパラメータは、トランザクションがコミットされる前にfsync()を使用して、すべてのWALページをディスクに同期するかどうかを決定します。電源をオフにすると、書き込みパフォーマンスが向上し、オンにすると、システムクラッシュ時の破損のリスクからの保護が強化されます。

- `checkpoint_timeout`:チェックポイント・プロセスは'コミットされたデータをディスクにフラッシュしますこれには、ディスク上で多くの読み取り/書き込み処理が含まれます。値は秒単位で設定され、値を小さくするとクラッシュリカバリ時間が短縮されます。値を大きくすると、チェックポイントコールが削減されるため、システムリソースの負荷が軽減されます。アプリケーションの重要度、使用状況、データベースの可用性に応じて、checkpoint_timeoutの値を設定します。

- `commit_delay = <num>` および `commit_siblings = <num>`:これらのオプションを組み合わせて使用すると、一度にコミットする複数のトランザクションを書き出すことで、パフォーマンスを向上させることができます。トランザクションがコミットされた瞬間に複数のcommit_siblingsオブジェクトがアクティブになっている場合、サーバはcommit_delayマイクロ秒を待って、一度に複数のトランザクションをコミットしようとします。

- `max_worker_processes / max_parallel_workers`:プロセスに最適なワーカー数を設定します。max_parallel_workersは、使用可能なCPUの数に対応します。アプリケーションの設計によっては、クエリで並列処理に必要なワーカーの数が少なくて済みます。両方のパラメータの値は同じにし、テスト後に値を調整することをお勧めします。

- `random_page_cost = <num>`:この値は、PostgreSQLが非シーケンシャルディスク読み取りを表示する方法を制御します。値を大きくすると、PostgreSQLはインデックススキャンではなくシーケンシャルスキャンを使用する可能性が高くなります。これは、サーバーに高速ディスクがあることを示します。計画ベースの最適化、真空化、クエリやスキーマの変更に対するインデックス付けなど、他のオプションを評価した後で、この設定を変更してください。

- `effective_io_concurrency = <num>`:このパラメータは、PostgreSQLが同時に実行を試みる同時ディスクI/O処理の数を設定します。この値を大きくすると、個 々 のPostgreSQLセッションが並行して開始しようとするI/O処理の数が増加します。指定できる範囲は1～1、000です。非同期I/O要求の発行を無効にする場合は0にします。現在、この設定はビットマップヒープスキャンにのみ影響します。ソリッドステートドライブ（SSD）やその他のメモリベースストレージ（NVMe）は、多数の同時要求を処理できることが多いため、数百の数を推奨します。

PostgreSQL設定パラメータの完全なリストについては、PostgreSQLのドキュメントを参照してください。

トースト

TOASTは、特大属性ストレージテクニックを表しています。PostgreSQLは固定のページサイズ（通常は8KB）を使用しており、タプルを複数のページにまたがることはできません。したがって、大きなフィールド値を直接保存することはできません。このサイズを超える行を格納しようとすると、トーストは大きな列のデータを小さな「ピース」に分割してトーストテーブルに格納します。

トーストされた属性の大きな値は'結果セットがクライアントに送信されるときにのみ(選択されている場合)プルアウトされますテーブル自体は非常に小さく、アウトオブラインストレージ（TOAST）を使用しない場合

よりも多くの行を共有バッファキャッシュに格納できます。

バキューム

通常のPostgreSQL操作では、更新によって削除または廃止されたタプルはテーブルから物理的に削除されず、VACUUMが実行されるまで存在したままになります。したがって、特に頻繁に更新されるテーブルでは、VACUMを定期的に実行する必要があります。ディスクスペースが使用されているスペースは、ディスクスペースが停止しないように、新しい行で再利用できるように再利用する必要があります。ただし、スペースはオペレーティングシステムに返されません。

ページ内の空き領域は断片化されません。VACUUMはブロック全体を書き換え、残りの行を効率的にパッキングし、1つの連続した空き領域ブロックをページに残します。

一方、VACUUM FULLは、デッドスペースのないまったく新しいバージョンのテーブルファイルを作成することで、テーブルを積極的に圧縮します。この操作により、テーブルのサイズは最小限に抑えられますが、時間がかかることがあります。また、処理が完了するまで、テーブルの新しいコピー用に追加のディスクスペースが必要になります。ルーチンバキュームの目的は、バキュームフルアクティビティを回避することです。このプロセスでは、テーブルが最小サイズに維持されるだけでなく、ディスクスペースの安定した使用量も維持されます。

表領域

データベース・クラスタが初期化されると、2つの表領域が自動的に作成されます。

。`pg_global` 表領域は共有システムカタログに使用されます。。`pg_default` tablespaceは、template1およびtemplate0データベースのデフォルトのテーブルスペースです。クラスタが初期化されたパーティションまたはボリュームの容量が不足し、拡張できない場合は、別のパーティションに表領域を作成して、システムを再構成できるようになるまで使用できます。

頻繁に使用されるインデックスは、ソリッドステートデバイスのような高速で可用性の高いディスクに配置できます。また、ほとんど使用されない、またはパフォーマンスが重要でないアーカイブデータを格納するテーブルは、SASドライブやSATAドライブなどの低コストで低速なディスクシステムに格納できます。

表領域はデータベースクラスタの一部であり、データファイルの自律的なコレクションとして扱うことはできません。これらは、メインデータディレクトリに含まれるメタデータに依存するため、別のデータベースクラスタに接続したり、個別にバックアップしたりすることはできません。同様に、（ファイル削除やディスク障害などによって）テーブルスペースが失われると、データベースクラスタが読み取り不能になったり、起動できなくなったりすることがあります。RAMディスクのような一時ファイルシステムに表領域を配置すると、クラスタ全体の信頼性が低下します。

作成後、要求元ユーザに十分な権限があれば、任意のデータベースから表領域を使用できます。PostgreSQLは、テーブルスペースの実装を簡素化するためにシンボリックリンクを使用します。PostgreSQLは、`pg_tablespace` Table（クラスタ全体のテーブル）を作成し、その行に新しいオブジェクト識別子（OID）を割り当てます。最後に、サーバはOIDを使用して、クラスタと指定されたディレクトリの間にシンボリックリンクを作成します。ディレクトリ `$PGDATA/pg_tblspc` クラスタで定義されている組み込み以外の各表領域を参照するシンボリックリンクが含まれます。