はじめに

■コース概要

PostgreSQL のチューニングや障害時のリカバリ手順、レプリケーションの概要など、より高度な運用管理のポイントを、実習を通して習得できます。

■コースのゴール

テキストを見ながら以下を行えるようになる。

・チューニングすべき SQL を洗い出し、実行計画を読める
・ボトルネックごとのチューニング方法を検討できる
・障害に対する事前準備と障害復旧の方法を説明できる
・レプリケーションの種類と概要を説明できる

■受講対象者

利用中の PostgreSQL のパフォーマンスや障害対応でお困りの方

■前提条件

以下 2 点を満たしている方

・PostgreSQL 管理の基礎理解
「PostgreSQL 管理基礎」コースを受講された方、もしくは、PostgreSQL のアーキテクチャ、PostgreSQL サーバーの起動停止やパラメータの変更、VACUUM などの PostgreSQL 管理の基礎を理解している方。

・Linux の基礎理解
Linux の基本操作として、ディレクトリ操作（ls、pwd、cd コマンド）やファイル操作（cat、vi コマンド）、Linux ユーザー操作（su、sudo コマンド）を理解している方。

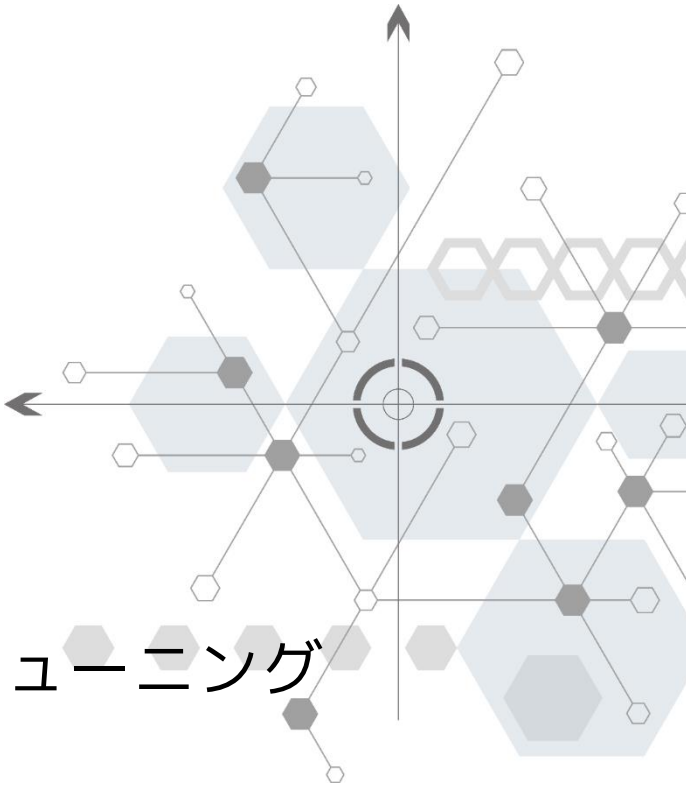
■テキスト内の記述について

▼構文

[ ]	省略可能
{ A | B }	A または B のどちらかを選択

▼マーク

	注意事項
	参考情報



第 3 章
データベースチューニング

この章では、PostgreSQL データベースのデータベースチューニングについて説明します。

01　データベースチューニング概要
02　パフォーマンスの測定
03　代表的なチューニングポイント

本章のゴール

・データベースチューニングの概要と手順を説明できる。
・代表的なチューニングポイントがわかる。

02 パフォーマンスの測定

PostgreSQL のパフォーマンスを測定する場合、累積統計情報ビューを参照します。
累積統計情報ビューでは、SQL の実行回数、アクセスブロック数、トランザクション数など、主にパフォーマンスに関する情報を確認できます。pg_stat_もしくは pg_statio_という接頭辞が付けられています。

(1) 情報収集に関する動作設定

累積統計情報は各 postgres バックエンドプロセスによって自身が実行した操作の記録を共有メモリ上に書き込み記録していきます。

パラメータ名	内容	デフォルト値
track_counts	表と索引に対するアクセス情報を収集するかどうかを制御します。	on
track_functions	ユーザー定義関数の使用状況を追跡するかどうかを制御します。	none
track_io_timing	ブロックの読み取り/書き込みに関する情報を収集するかどうかを制御します。	off
track_activities	実行中の SQL を監視するかどうかを制御します。	on

(2) 統計情報のリセット

累積統計情報はデータベースクラスタ作成時もしくはデータベースクラスタが異常終了して累積統計情報が破棄された時点からの累積値として表示されます。データベースクラスタを再起動しても継続して保持されます。効果的な測定のためには、定期的に累積統計情報の値を記録しておき、中長期での変化を確認することが推奨されます。もしくは、調査に際し、対象期間の開始時点で蓄積された値をリセットします。
リセットする場合は、pg_stat_reset()関数をスーパーユーザーで実行します。

構文

```
SELECT pg_stat_reset();
```

■主な累積統計情報ビュー

ビュー名	内容
pg_stat_activity	現在実行中の SQL に関する情報を表示します。
pg_stat_database	各データベースの使用状況を表示します。
pg_stat_all_tables	各表へのアクセスに関する統計情報を表示します。
pg_statio_all_tables	各表への I/O に関する統計情報を表示します。

累積統計は PostgreSQL 14 までは実行時統計や稼働統計と呼ばれていました。PostgreSQL 15 から統計情報の取得アーキテクチャと併せて名称も変更されました。

03 代表的なチューニングポイント

（1）共有バッファのチューニング

SQL 処理のためにディスクから読み込んだデータは共有バッファ上にキャッシュされます。
それを再利用することでディスク I/O を削減できます。共有バッファの再利用率（キャッシュ・ヒット率）を確認しながら適切なパラメータ設定を行います。

1）キャッシュ・ヒット率の調査

共有バッファのキャッシュ・ヒット率が低くないか確認します。

■表レベルのキャッシュ・ヒット率

```
/* pg_statio_user_tables 表からキャッシュ・ヒット率を確認 */
bench=# SELECT relname,
bench-#  ROUND(heap_blks_hit*100/(heap_blks_hit+heap_blks_read),2) AS cache_hit_ratio
bench-# FROM pg_statio_user_tables WHERE  heap_blks_read > 0 ORDER  BY cache_hit_ratio;
     relname      | cache_hit_ratio
------------------+-----------------
 dept             |            0.00
 pgbench_accounts |            9.00
 pgbench_branches |           76.00
       :
```

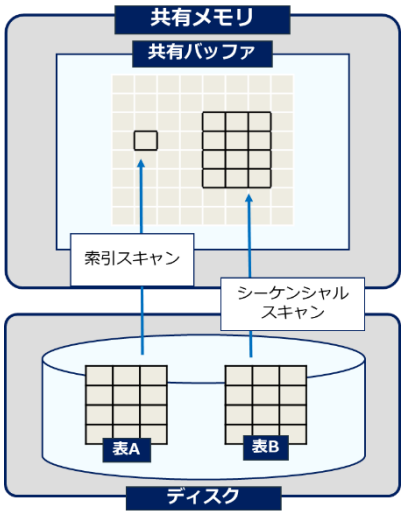
pg_statio_user_tables からは、表単位で収集されたヒット数（heap_blks_hit）とディスク読み込み数（heap_blks_read）が得られます。

MEMO

2）共有バッファのチューニング方法

通常、次のようなステップでチューニングを行います。

・SQL チューニング
本来期待する実行計画が選択されず不要なデータを読み込むことでキャッシュ・ヒット率が低下していることがあります。このような場合、シーケンシャルスキャンを索引スキャンに変更して、必要なブロックのみ読み込むようにするなど、メモリを効率的に使用できるように調整します。



・共有バッファのサイズを増加
SQL が適切に実行されていてもキャッシュ・ヒット率が低い場合、共有バッファのサイズが小さすぎる可能性があります。そのため、適切なサイズに調整します。
共有バッファのサイズは、shared_buffers パラメータで指定します。デフォルト値は 128MB です。
shared_buffers のサイズは一般的にデータベースサーバーに搭載されているメモリの 25%から 40%を設定すると良いと言われています。

・pg_prewarm
期待した表が共有バッファに乗らない場合は、pg_prewarm エクステンションを利用して明示的に共有バッファに読み込むこともできます。

■pg_prewarm で表を共有バッファに読み込む

```
/* pg_prewarm エクステンションを作成 */
bench=# CREATE EXTENSION pg_prewarm;
CREATE EXTENSION

/* pg_bench_accounts 表を明示的に共有バッファに読み込む */
bench=# SELECT pg_prewarm('pgbench_accounts');
 pg_prewarm
------------
       16394
```

huge page を使用するかどうかを制御するパラメータに huge_pages があります。huge page を使用すると操作するメモリサイズが大きくなり、メモリ管理のオーバーヘッドを減らせます。shared_buffers パラメータの値が大きい場合に有効です。

(2) WAL バッファのチューニング

頻繁に変更処理が行われるシステムでは、WAL バッファのサイズを調整することでパフォーマンスを改善します。

1) 変更処理に関するアーキテクチャ

データベースで行われたすべての変更履歴は、WAL ファイルに書き込むことで永続化されます。
変更履歴は WAL バッファに一時的に格納され、以下のタイミングで walwriter プロセスがバッファの内容をまとめて WAL ファイルに書き込みます。

・トランザクションがコミットされたとき
・WAL バッファが一杯になったとき
・wal_writer_delay パラメータに設定された時間が経過したとき
（デフォルト：200ms/書き込む内容が無い場合は 5s 休止）

2) WAL バッファのサイズに関する問題

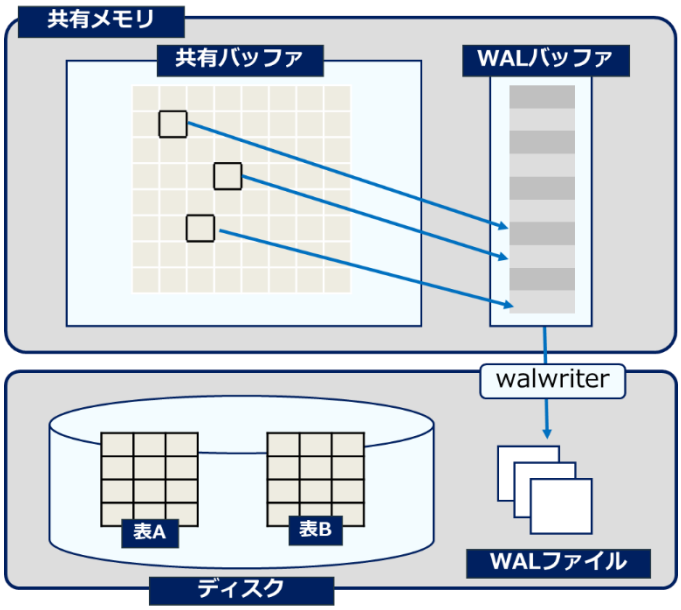
WAL バッファのサイズが小さすぎると、コミット以外のタイミングでも書き込みが発生します。
その結果、ディスク I/O が増加しパフォーマンスが低下する恐れがあります。
そのため、以下のようなケースでは、WAL バッファのサイズを適切に設定し、変更履歴の書き込み頻度を減らすようにします。

・1 トランザクションでの更新データ量が多い場合
・複数セッションで同時に更新処理を行っている場合

3) WAL バッファのサイズの調整

WAL バッファのサイズは、wal_buffers パラメータで指定します。
デフォルト値は「-1」で、shared_buffers パラメータの 1/32（約 3%）が確保されます。

■変更履歴の永続化



ディスクの性能が低くて WAL ファイルへの書き込みが遅い場合、WAL ファイルへの書き込み I/O 削減がパフォーマンスの改善に効果的なことがあります。この問題に対処する方法として、下記パラメータの調整が考えられます。

・wal_compression パラメータ

WAL のフルページ書き込みを圧縮するかどうかを制御するパラメータです。pglz、lz4、zstd から圧縮形式を選択できます。WAL のフルページ書き込みはチェックポイント後の最初の書き込みや pg_basebackup の実行などで発生します。WAL のフルページ書き込みはデータの変更量に関わらず 8KB で、サイズが大きくなる傾向があります。そのため、WAL を圧縮することで書き込み量の削減が期待できます。

デフォルトは off（圧縮しない）です。圧縮展開時に CPU リソースを使用する点を考慮して設定するかを検討します。

・full_page_writes パラメータ

WAL のフルページ書き込みを行うかを制御するパラメータです。チェックポイント後にそのページが最初に変更された過程でページすべての内容を WAL に書き込むかどうかを制御するパラメータです。off に指定した場合は更新分のデータのみが WAL に書き込まれるため、WAL 書き込みによる I/O が低減します。その結果、応答性能が向上する可能性があります。デフォルトは on です。ただし、off に設定すると OS クラッシュや電源障害時にデータ破損を引き起こす恐れがあるため、特別な要件が無い限り on にして運用します。