PostgreSQL インサイド

チューニング
データベースチューニング
技術を知る

FUJITSU

- [ ] 導入／環境設定
- [ ] 移行
- [ ] 性能
- [x] チューニング
- [ ] バックアップ／リカバリー
- [ ] 冗長化／負荷分散
- [ ] 監視
- [ ] データ連携
- [ ] 災害対策
- [ ] 豆知識

データベースのチューニングとは、データベースの性能維持または向上を阻害するボトルネックを見つけ、その原因を調査し、解決していくことです。ここでは、チューニングの 1 つである「データベースチューニング」について解説します。

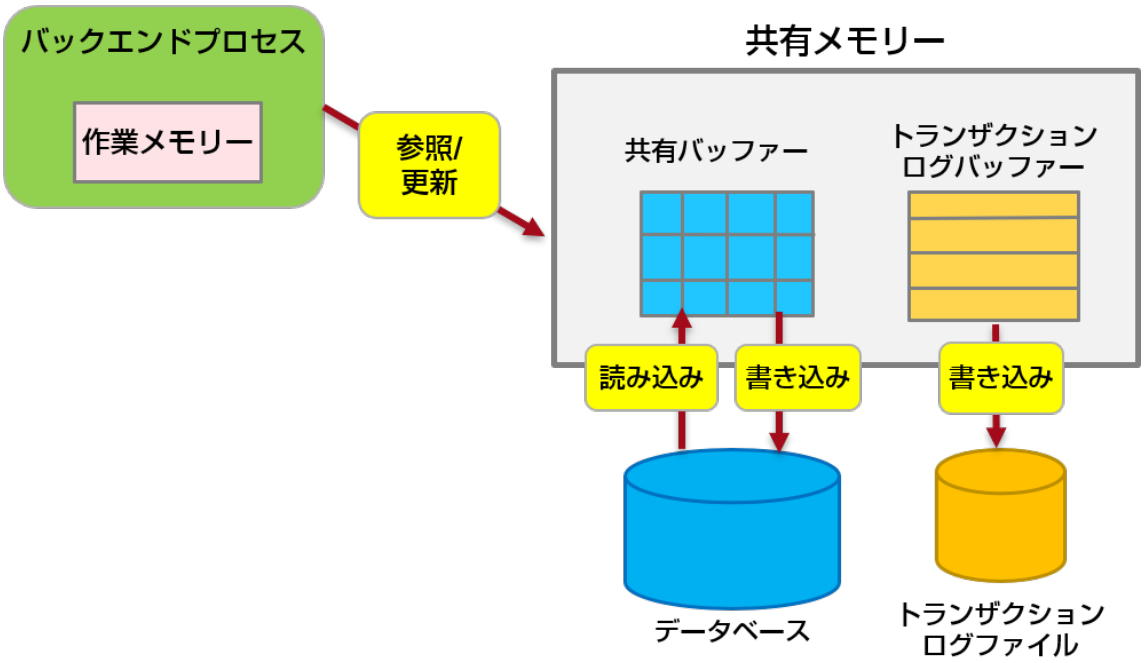
1. データベースチューニングとは

データベースチューニングは、サーバーの性能を最大限に利用できるようにデータベースシステムが使用するメモリー使用量を最適化し、ディスク I/O を減らすことを目的としています。システム構成や運用内容に応じて、セットアップ時の初期設定の段階で実施しておくことができます。
データベースチューニングの解説を始める前に、まず、データベースチューニングの前提となるメモリーとディスク I/O について簡単に説明した後、実際のチューニング方法を説明していきます。

1.1 メモリーとディスク I/O

PostgreSQL がデータベースにアクセスする場合、まずディスク上の必要なデータを共有メモリー上の共有バッファーに読み込みます。そして共有バッファー上のデータを読み書きすることで処理を行います。その後、また同じデータにアクセスする必要が生じた場合には、ディスクにアクセスするのではなく、すでに共有バッファー上に読み込まれているデータに対してアクセスすることで、ディスク I/O を低減させています。
トランザクションログに関しても同様で、データ更新の記録であるトランザクションログは、一度、共有メモリー上のトランザクションバッファーにためられ、その後、ディスク上のトランザクションログファイルに書き込まれます。トランザクションログファイルに書き込まれるタイミングは、トランザクションがコミットされた場合と、トランザクションログバッファーが満杯になった場合です。
また、作業メモリーは、バックエンドプロセス（注 1）ごとに確保されるメモリーです。ソート処理やテーブル結合などで使用されます。作業メモリーが不足すると、ディスク上に作業ファイルが作成され、作業ファイルに対するディスク I/O が発生してしまいます。

- 注 1　クライアントからの接続要求を受けたときに生成されるプロセス。SQL はこのプロセス内で実行される。



データベースチューニングでは、これらの仕組みをもとにメモリーを適切に割り当てて、ディスク I/O を減らすことで、データベースの処理性能を高めていきます。

1.2 データベースチューニングの方法

データベースチューニングの方法は、postgresql.conf ファイルのパラメーターチューニングです。postgresql.conf には多くのパラメーターがありますが、以降では、データベースチューニングで有効なパラメーターを以下のカテゴリーに分けて説明していきます。

カテゴリー	パラメーター名
接続	max_connections
メモリー	shared_buffers
	work_mem
	maintenance_work_mem
トランザクションログ（WAL）	wal_buffers
	max_wal_size
	checkpoint_timeout
自動バキューム処理	autovacuum_work_mem
	autovacuum_vacuum_threshold
	autovacuum_vacuum_scale_factor

カテゴリー	パラメーター名
ロック処理	deadlock_timeout

なお、ここでは、PostgreSQL 10 で設定可能なパラメーターを説明しています。また、各パラメーターのデフォルト値は、PostgreSQL 10 でのデフォルト値を記載しています。

参考

FUJITSU Software Enterprise Postgres では、運用管理ツール「WebAdmin」を使用してデータベースクラスタを作成した場合、FUJITSU Software Enterprise Postgres が運用に最適な値を各パラメーターに自動設定します。必要に応じて、WebAdmin でのパラメーターチューニングも可能です。

2. パラメーター

データベースチューニングで有効なパラメーターを説明します。

2.1 接続に関するパラメーター

max_connections

PostgreSQL に同時に接続できるクライアントの最大数です。デフォルト値は、100 です。値を大きくするとメモリー使用量が増え、性能に影響する可能性があります。これは、PostgreSQL に接続するたびに新しいサーバープロセスが起動されるためであり、共有メモリーやセマフォをチューニングする必要があります。業務にあわせて適切な接続数を設定することが重要です。

2.2 メモリーに関するパラメーター

shared_buffers

共有バッファーのサイズです。デフォルト値は、128 メガバイトです。デフォルト値が比較的小さく設定されているため、物理メモリーが 1 ギガバイト以上搭載されているマシンの場合、その 25%程度の値を設定することを推奨します。共有バッファーを利用することで処理効率の向上が見込まれますが、大きすぎるとバッファー検索に時間がかかる、また、メモリー領域を圧迫してディスクスワップが発生するなど、逆に性能劣化となる可能性があるため、設定値については、検証を実施しながらチューニングする必要があります。

参考

pg_stat_database ビューの blks_read（ディスクから読み込んだブロック数［ディスク I/O 発生］）と blks_hit（バッファーにヒットしたブロック数）から、キャッシュヒット率が確認できます。以下の SQL を実行します。pg_stat_database ビューについては、「PostgreSQL 文書」を参照してください。

【SQL の例】

```
SELECT datname,blks_read, blks_hit, round (blks_hit * 100 / (blks_hit + blks_read),2) AS hit_ratio FROM pg_stat_database WHERE blks_read > 0;
```

work_mem

問い合わせ実行時に、ソートやハッシュテーブル操作のために使用される作業メモリーのサイズです。デフォルト値は、4 メガバイトです。work_mem の値を大きくすることで、問い合わせ性能の向上が見込まれます。しかし、work_mem の値が大きいと、接

続数や問合せ内容に比例し、必要となるメモリーが大きくなり、メモリーが枯渇する可能性があります。最大でも「(物理メモリー - shared_buffers の値) ÷ max_connections の値」以下の値を設定します。複雑な問い合わせの場合、問い合わせの中でソートやハッシュが複数回実行されることがあります。この場合、設定された work_mem の値の数倍のメモリーが必要となり、メモリー不足から性能劣化を招く可能性があるため注意が必要です。

maintenance_work_mem

VACUUM、CREATE INDEX、および ALTER TABLE ADD FOREIGN KEY などの保守操作のために使われる作業メモリーのサイズです。デフォルト値は、64 メガバイトです。作業メモリーを大きくすることで、手動バキュームやインデックス作成などを高速に行うことが可能です。

2.3 トランザクションログ（WAL）に関するパラメーター

トランザクションログに関するパラメーターでは、まず「チェックポイント」について理解しておく必要がありますので、「チェックポイント」を簡単に説明します。PostgreSQL では、共有バッファー上の更新データを頻繁にディスクに書き込みません。しかし、何らかの障害が発生し、共有バッファーの更新データが消えてしまったという事態に備えて、ディスク上のトランザクションログファイルにトランザクションログを書き込むことで障害発生時のデータ信頼性を確保しています。ただし、トランザクションログも無限にためられるわけではなく、どこかのタイミングで制御する必要があります。このタイミングが「チェックポイント」です。
チェックポイントでは、共有バッファー中の更新データをディスクに反映するとともに、不要なトランザクションログを削除しています。チェックポイントはディスクへの書き出しを行う処理のため、頻発すると性能低下の原因となります。そのため、「チェックポイントをどの程度の間隔で発生させるか」という設定が重要となります。この設定は、max_wal_size と checkpoint_timeout で設定します。max_wal_size または checkpoint_timeout のどちらかの閾値に達すると、チェックポイントが発生し、共有バッファー上の更新データがすべてディスクに書き出されます。そのため設定値を小さくすると、頻繁にディスク I/O が発生してしまいます。一方、値を大きくしすぎると、リカバリーに必要な更新データをトランザクションログとして保持していることになり、リカバリー時間が長くなる可能性があります。

wal_buffers

トランザクションログバッファーのサイズです。デフォルト値は、shared_buffers の値の 32 分の 1 です。トランザクションログは、トランザクションがコミットするたびにディスク上のトランザクションログファイルに書き込まれるため必要以上に大きな値を設定する必要はありません。しかし、頻繁にデータ更新が発生するトランザクションを実行した場合や多数のトランザクションを同時に実行した場合など、未書き込みのトランザクションログでトランザクションログバッファーが満杯になってしまうことがあります。このような場合、トランザクションのコミットを待たずに、たまったトランザクションログをすべてディスクに書き出すといったディスクアクセスが発生してしまいますので、値を大きくします。

max_wal_size

チェックポイント処理を行う契機となるトランザクションログのサイズです。デフォルト値は、1 ギガバイトです。ここで指定するサイズ分のトランザクションログがトランザクションログファイルに書き込まれるとチェックポイント処理が行われます。ログメッセージに以下のメッセージが出力されている場合は、チェックポイントが頻発していることを示していますので値を大きくします。

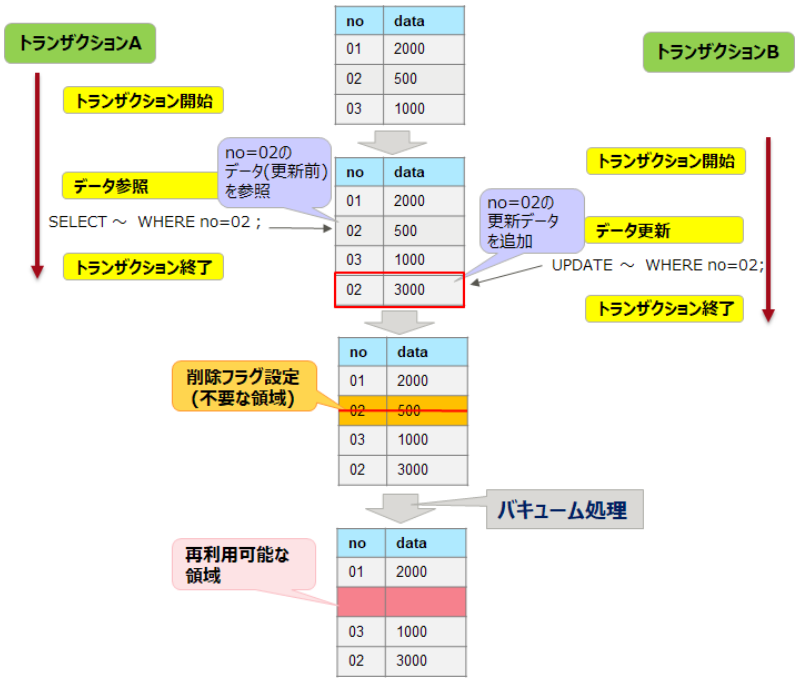
```
LOG:  checkpoints are occurring too frequently (19 seconds apart)
                                        HINT:  Consider increasing the configuration parameter "max_wal_size"
```

checkpoint_timeout

チェックポイント処理を行う間隔を時間で指定します。デフォルト値は、5 分です。デフォルト値は短めの設定となっているため、30 分を初期値として設定することを推奨します。値の設定には、前述したようにリカバリー時間の考慮が必要ですが、データの更新が少ない場合は、トランザクションログ量が少ないため、比較的長めの値を設定することができます。

2.4 自動バキューム処理に関するパラメーター

PostgreSQL では、ディスク内のテーブルのデータに対して更新や削除をしても、それらのデータには削除フラグが設定されているだけで、更新前のデータが元の場所に残っています。更新後のデータは、新たなデータとして末尾に追加されていきます。これは、「追記型アーキテクチャ」と呼ばれるしくみです。PostgreSQL では、追記型アーキテクチャーの採用により、同じデータに対する更新処理と参照処理が同時に実行できる MVCC（MultiVersion Concurrency Control：多版型同時実行制御）を可能にしています。削除フラグがつけられた更新前のデータを参照するトランザクションが存在しなくなると、このデータ領域は、不要な領域として扱われます。不要な領域が増えると、ファイルサイズが大きくなるとともに、共有メモリーにキャッシュがされにくくなり、ディスクアクセスが増加することで性能低下につながります。ファイルサイズが無制限に増加しないよう、この不要な領域を再利用可能な状態に変更する処理が「バキューム処理」です。



バキューム処理は、通常、自動バキューム機能により自動的に行われますが（postgresql.conf の autovacuum パラメーターのデフォルト値が有効）、より効果的に行われるように、自動バキュームに関するパラメーターを調整します。

autovacuum_work_mem

自動バキュームで使用する作業メモリーのサイズです。デフォルト値は、maintenance_work_mem の設定に従います。作業メモリーが不足すると、バキュームにかかる時間が増えるため性能が低下します。自動バキュームの実行ログを参照し、インデックススキャンが 2 以上であるような場合は、メモリーを増やすと効果的です。自動バキュームの実行ログ出力は、postgresql.conf の log_autovacuum_min_duration パラメーターで設定します。

【自動バキュームを実行した際のログ出力例】

```
Log: automatic vacuum of table "mydb.public.sales": index scans: 2
                                                    :
```

autovacuum_vacuum_threshold / autovacuum_vacuum_scale_factor

autovacuum_vacuum_threshold は、自動バキューム処理を行うかどうかの閾値となる更新行数です。デフォルト値は、50 行です。autovacuum_vacuum_scale_factor は、自動バキューム処理を行うかどうかの閾値となる更新データの割合です。デフォルト値は 20%であり、これは、テーブルの 20%が不要な領域となったことを示します。自動バキューム処理は、テーブル内で更新された行数が、「autovacuum_vacuum_threshold の値 + autovacuum_vacuum_scale_factor の値 × テーブルの行数」 の 閾値を超えたときに

実行されます。autovacuum_vacuum_scale_factor では割合を指定するため、バキューム処理の対象となる不要な領域の行数は、テーブル内の全体行数に依存し、テーブルの行数が多いほど、バキューム処理に時間がかかります。行数が多いテーブルの場合は、autovacuum_vacuum_scale_factor に小さい値を設定し、少ない頻度でバキューム処理を実行させると効率的です。ただし、postgresql.conf ファイル内でパラメーターを設定すると全テーブルが対象となってしまいます。バキューム処理の実行契機をテーブルごとに制御できるように、ALTER TABLE 文を使用してテーブル単位での設定を推奨します。

2.5 ロック処理に関するパラメーター

PostgreSQL では、複数のトランザクションが同じデータを更新する「ロック待ち」が発生すると、それがデッドロック（決して解除されることのないロック）かどうかを検出するための処理が実行されます。デッドロックの検出処理は、データベースに対して負荷がかかる処理であり性能低下の原因となり得ることから、デッドロック検出処理開始の猶予時間を設定します。

deadlock_timeout

デッドロック検出処理を待機する時間です。デフォルト値は、1 秒です。データ更新が頻繁に発生するような高負荷なシステムでは、ロック待ちの時間が長くなることが多く、デッドロック検出処理が頻繁に行われてしまう可能性があるため、「デフォルト値（1 秒） × 同時セッション数」の設定を推奨します。

ここでは、データベースチューニングを解説しました。性能劣化がおきないように、システム規模や運用要件にあわせて、適切なパラメーターチューニングを実施してください。

2019 年 3 月 26 日