PGECons
PostgreSQL Enterprise Consortium

PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#2

チューニング編

製作者 担当企業名
NECソリューションイノベーティブ株式会社
株式会社富士通ソーシアルサイエンスラボラトリ
富士通株式会社

改訂履歴

版	改訂日	変更内容 ※表の列はスタイル「94.表内」を選択セルの背景は網掛け 25%灰色
第 1 版	2014/02/20	新規作成

はじめに

■本資料の概要と目的

本資料では、DBMS の変更による性能特性の差異を確認し、システムの性能要件を達成する方法に関する参考資料として、チューニングの作業や具体的な作業内容を紹介します。

本資料は以下のような構成となっています。

1. 目標性能の設定
   移行元のシステムにおける性能要件を確認し、
2. PostgreSQL に固有な性能上の留意点
3. 設計（パラメータ設計）
   性能関連のパラメータ設定やビルド時の指定、システム構成を紹介します。
4. 実装、構築
   SQL 文の最適化作業について紹介します。

資料の構成は、チューニング作業の手順に沿っています。
DB 移行に先立ち、達成すべき性能目標を設定したあと、パラメータ設定など性能の確保に必要なシステム設計上の検討を行い、移行したアプリケーションが実行する SQL 文の性能を確保するためチューニングを行います。

本資料が紹介する PostgreSQL のチューニングは、PostgreSQL の標準機能で実現可能な範囲に限定しています。
以下に挙げたような外部の機能を利用した性能情報収集や追加コンポーネントを使用したチューニングについては、一部にに前を紹介していますが、本資料の中では詳細な説明は行いません。

1. 性能情報収集
   チューニングには継続的かつ詳細な性能情報収集が必要となる場合があります。性能情報の定期的な収集やアラートには専用のツールを使用しますが、性能情報収集は 2013 年度 WG3 の活動の中で調査を行うため、本資料では取り扱いません。
2. 追加機能
   ヒント情報や統計情報のセーブ・ロードなど標準では提供されていない機能については、本資料では取り扱いません。

■本資料で扱う用語の定義

本資料では、曖昧な意味としてとらえることができる用語を用いますが、次のような意味で記載しています。

表 1: 用語定義

No.	用語	意味
1	DBMS （でーびーえむえす）	データベース管理システムを指します。ここでは、PostgreSQL および異種 DBMS の総称として利用します。

■本資料で扱うソフトウェア

本資料では、次のソフトウェアを用いてコマンド例証等を挙げています。以下ソフトウェアバージョンと異なるソフトウェアを用いる場合や、特別な設定等を入れて実行する場合は、実行結果を記載する章で利用するソフトウェアや設定について紹介します。

表 2: 動作環境

No.	環境名	実装	バージョン
1	移行先 DBMS	PostgreSQL	9.3
2	移行元 DBMS	Oracle Database	任意
3	〃	Microsoft SQL Server	任意
4	〃	DB2	任意

表 3: コマンド・ツール一覧

No.	コマンド・ツール	バージョン	ライセンス	入手元(URI)	概要
	pg_hint_plan	1.1.0	BSD	http://en.sourceforge.jp/projects/pghintplan/	ヒント情報による検索プランの指定を行うツール。
	pg_dbms_stats	1.3.1	BSD	http://sourceforge.jp/projects/pgdbmsstats/	PostgreSQL の実行計画で利用する統計情報を固定化して実行計画を制御するツール。

目次

1. 目標性能の設定

移行対象のシステムに関する性能要件や稼働中のシステムの性能情報を確認し、PostgreSQL 移行後の性能目標を設定します。主に、以下の情報を収集します。

1. システムの性能要件
2. 現行システムから収集した性能情報

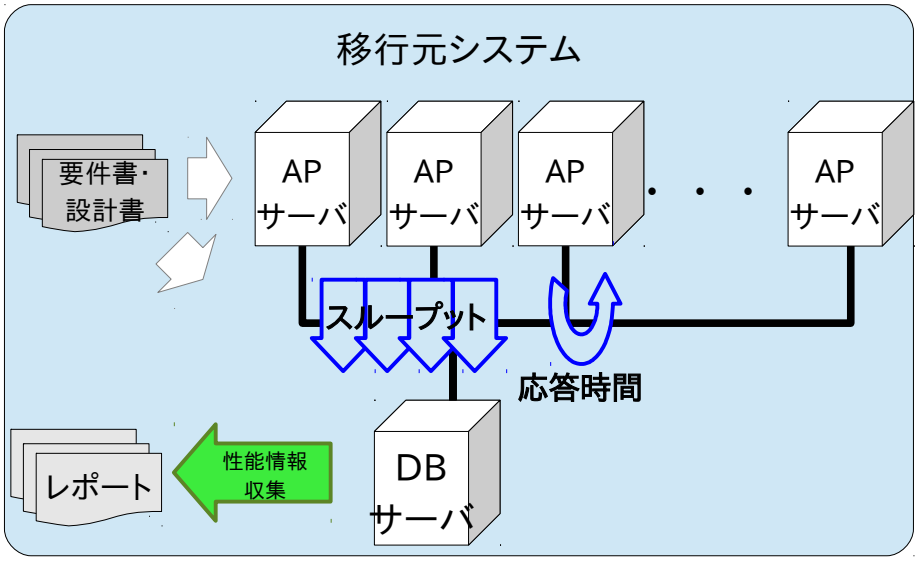


図 1.1: 既存システムの情報収集

1.1. システムの性能要件

PostgreSQL 移行後の性能目標（ベースライン）として、移行元システムにおける性能要件を確認します。システムに対て求められる応答時間やスループットデータ件数、消費するシステムリソース等に関する要件を要件書や設計書から抽出し、移行後のシステムで実現すべき性能目標とします。

DB に求める明示的な性能要件が存在しない場合、端末からの要求に対する応答時間やアプリケーションの処理時間に関する要件または性能データ等から、間接的に DB が達成するべき性能要件が確認可能な場合もあります。

1.1.1. 応答時間／処理件数

現行システムを構成するサブシステムや処理ごとに設定されている次のような性能要件を確認します。

- クエリの目標応答時間／応答時間の上限
- トランザクションの目標応答時間／応答時間の上限
- 単位時間当たりのトランザクション処理件数
- その他

応答時間には、要件書に指定されている性能要件以外に、アプリケーションサーバや API、監視システム、クラスタシステム等に設定されているタイムアウト時間など、実質的な応答速度の制限が存在する場合があるため、注意が必要です。

1.1.2. 同時接続数

PostgreSQL は設定およびサーバリソース次第で最大 8,388,607 接続が可能ですが、接続数分のプロセスが起動しワークロードの制御等を行わずに OS のリソースを要求することから、あまりに多くの DB 接続を行うと性能が劣化する可能性があります。

現行システムに関する以下のような要件や実際の設定値を確認してください。

- DB の設定における接続数の上限値
- アプリケーションサーバの台数／コネクションプールの設定
- アプリケーションサーバ以外が使用するコネクションの有無（バックアップサーバ、ジョブコントロールサーバ、監視システム、クラスタシステム、メンテナンス用のコンソール等）

1.1.3. 消費するハードウェア／OS リソース

現行システムにおいて、DBMS が利用する各種ハードウェアリソースの上限を設定しているかどうかを確認します。

- CPU
- メモリ
- ディスク領域
- ネットワーク
- その他

ハードウェアリソースは、DB インスタンスのほかに、業務ごとの利用制限が存在する場合があるため注意が必要です。ただし、PostgreSQL 自身は設定可能なハードウェアや OS リソースの上限に関する設定が限定的であるため、要件として求められているリソース制限ができない場合もあります。

1.1.4. 現行システムのチューニングに関する特記事項

既存システム開発時に発生した問題に対しては、DBMS 変更後のシステムにおいても同様の対策が必要となる可

能性があります。DB サーバや API、ネットワーク等の設定を変更したり、アプリケーションが実行するクエリの作成上問題となった点、ヒント情報、でプランを固定している個所など、移行元のシステム開発時に発生したボトルネックやチューニングの記録が存在する場合、事前に確認することを推奨します。

1.2. 現行システムの性能情報採取

文書化された性能要件と合わせて、現行の DBMS の設定や、以下のような運用中の DBMS の性能情報を採取し、目標性能の参考とします。

- 同時接続数
- スループット
- クエリの平均・最長応答時間
- データ容量
- CPU、I/O の負荷状況
- etc.

要件書や設計書において特に明示的な性能要件が存在しない場合、採取した性能情報を元に目標性能を設定します。以下、収集する主な情報および手段を例示します。

- システムレベルでの応答時間や処理件数
  Web アプリケーションの同時接続数や応答時間など、各システムに合わせて、応答時間や処理件数に関する情報を収集してください。
- Oracle における性能情報収集手段
  STATSPACK
  EnterpriseManager など
- Microsoft SQL Server における性能情報収集手段
  パフォーマンス データ コレクション
  パフォーマンスモニタなど
- DB2 における性能情報収集手段
  Optim Performance Manager
  DB2 管理ビューなど
- OS の性能情報収集、Windows）
  iostat（UNIX/Linux）
  vmstat（UNIX/Linux）
  パフォーマンスモニタなど

2. PostgreSQL に固有な性能上の留意点

PostgreSQL のチューニングを行う際、以下のような PostgreSQL に固有の問題点が存在します。本章では、チューニング作業において留意すべき PostgreSQL に固有な機能や動作、他 RDBMS との違いについて紹介します。データベース性能に影響する PostgreSQL に固有な検討事項としては、主に以下が挙げられます。

1. VACUUM 運用
2. COMMIT 処理による参照性能劣化
3. ロックの動作
4. 運用中のパラメータ変更
5. チューニング機能の差異

2.1. VACUUM 運用

PostgreSQL は追記型のデータベースであり、不要領域の回収やレコードの可視性を確保[1]するために VACUUM を実行する必要があります。VACUUM によりテーブルの肥大化を抑制したり、運用中の I/O 負荷を増大させる可能性があるため、性能の観点からも運用を検討する必要があります。

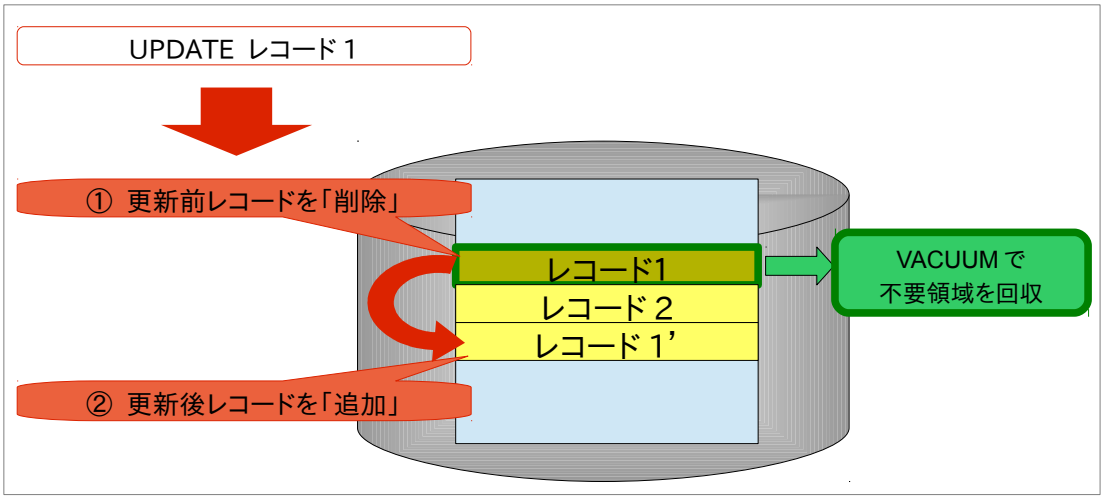


図 2.1: 追記型の仕組みと VACUUM

VACUUM 処理はバージョンを追うごとに改善されています。

- PostgreSQL8.3：HOT 機能の追加により、適切な物理設計により VACUUM の必要性が減少
- PostgreSQL8.4：Visibility Map により、VACUUM の処理対象を更新されたブロックに限定
- PostgreSQL9.0：VACUUM FULL の処理変更（テーブル内の再編成ではなく別領域へのデータの再作成）
- PostgreSQL9.3：VACUUM FREEZE による初期ロードデータへの VACUUM 運用回避

また PostgreSQL8.2 以降は自動 VACUUM が既定値で有効になっており、運用上の負担も軽減されています。

これらの改善を前提としてもなお、VACUUM 処理は依然として性能に影響を与える PostgreSQL に固有な運用として意識しておく必要があります。

- テーブルの分割／パーティショニング

  インデックスの VACUUM はインデックスファイル全体を読み込むため、あまり更新されないテーブルであっても高負荷になる場合があります。また、ログのように一定期間保持した後で廃棄するデータは、1 テーブルに格納して DELETE するのではなく、テーブルを分けてテーブルごとに削除することにより、VACUUM 運用自体を回避することができます。
- 自動 VACUUM のチューニング

  また、自動 VACUUM も既定の設定値では一律でテーブルデータの 20%が更新されると VACUUM 処理を行うため、レコード件数の増大により VACUUM 実行間隔が広がり、VACUUM 処理によるシステムへ

1 PostgreSQL では、レコードが更新されてから約 20 億トランザクションが経過する前に VACUUM をかけなければ存在するレコードが不可視になるため、最後に VACUUM が実行されてから一定数のトランザクションが実行されると強制的に VACUUM を実行します。

の負荷が増加する可能性があります。極端にデータ件数の異なるテーブルについては、自動 VACUUM を実行する閾値を個別に設定したり、自動 VACUUM の対象から外し個別に VACUUM を実行するなど、運用側での対処が必要となります。
テーブルごとの設定変更については、CREATE /ALTER TABLE の WITH 句をご覧ください。

2.2. COMMIT 処理による参照性能劣化

PostgreSQL はトランザクションが実行中か COMMIT または ROLLBACK したことを示す情報を clog ファイルに持っており、更新されたレコードが最初に参照され、当該データベースブロックがファイルに反映されたときに初めて COMMIT または ROLLBACK されたことがデータベースファイルに反映されます。

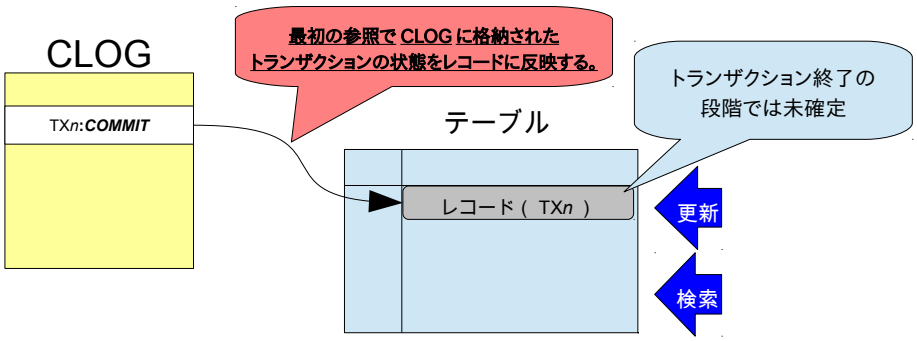


図 2.2: 参照処理における書き出し処理

このため、最初に行った参照処理ではファイルへの書き出し処理が発生し、想定した性能が得られない可能性があります。ログのように大量にデータを追記したあとで分析のために参照処理を実行すると、想定外のファイル書き出しが行われ、システム全体の処理性能を劣化させる可能性があります。

意図しないファイル書き出しを回避するには、更新トランザクション実行後に意図的な参照を行うか、初期データロードであれば COPY FREEZE（9.3～）を使用するなど、運用による回避を検討する必要があります。

2.3. ロックの動作

RDBMS には、大量更新によるロックリストの増加による性能劣化を回避するため、ロックの粒度をレコードからブロック、テーブルへと切り替える（ロックエスカレーション）製品があります。

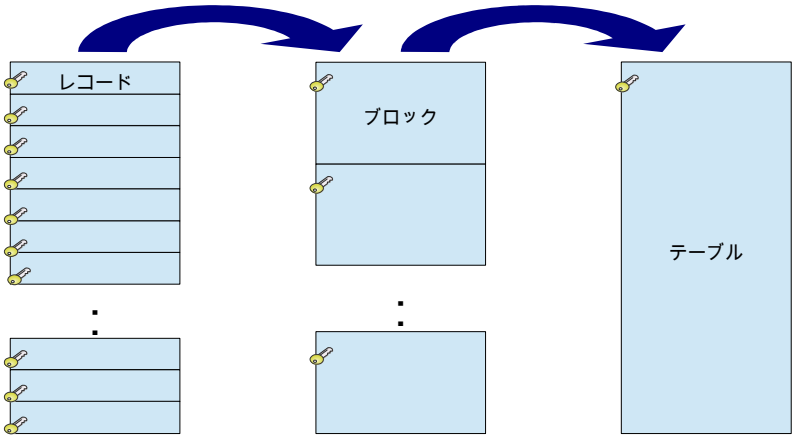


図 2.3: ロックエスカレーション

既定値ではロックエスカレーションを行うRDBMSの例としては、Microsoft SQL Server、DB2が挙げられます。これらのRDBMSでは更新処理件数や同一テーブルに対する更新処理の同時実行に応じて、インデックスや検索プランの変更、ロックリストのサイズや設定変更を行う必要があります。

PostgreSQLはロックエスカレーションを行いませんので、これらを検討する必要はありません。ただし、デッドロックやロックウェイトが発生する可能性を考慮したデッドロック検出の間隔（3.1.8 ロック管理）や、1トランザクションで多数のリレーションを更新する場合の最大ロック数（max_locks_per_transaction パラメータ）の設定値は検討する必要があります。

2.4. パラメータ設定変更の反映

PostgreSQL パラメータは、運用中に設定変更が可能なパラメータや DB 接続を行ったセッションごとに変更可能なパラメータが存在します[2]。設定を反映するタイミングには、以下のパターンがあります。

- DB の再起動が必用
  - ビルドもしくは DB 初期化時に設定
  - DB サーバの再起動時に反映可能
- DB を再起動せずに反映可能
  - リロード（pg_ctl reload）で反映
  - セッション開設時に反映
  - superuser が DB セッション内で変更可能
  - ユーザが DB セッション内で変更可能

2.5. サポートするチューニング機能の違い

商用 DBMS は様々なチューニング機能を提供しています。PostgreSQLも基本的な機能は提供していますが、機能存在しなかったり、実現するには追加ソフトウェアの導入が必要な機能があります。以下、商用 DBMS が提供するチューニング機能のいくつかをピックアップし、PostgreSQL の対応状況を紹介します。

1. 自動パラメータチューニング
   DB の運用中にデータやリクエストの傾向に関する情報を収集し、パラメータの設定変更やインデックス定義、SQL 文の変更のアドバイスを行う機能。

2. Hint 情報
   安定した性能を実現するため、特定のプランで検索処理を実行するよう予め指定する機能です。
   通常、RDBMS のオプティマイザははコスト計算により最適と判断したプランを選択しますが、データの傾向や統計情報のメンテナンス状況などの影響で不正なプランを選択する可能性があります。これを回避するため、クエリに対して特定の検索プランを選択するように指定します。

3. 統計情報のセーブロード
   ヒント情報と同様に、安定した性能を実現するために良好なプランを選択するオプティマイザ統計情報をセーブ・リストアし本番環境におけるプラン選択に利用する機能です。

4. バッファ固定
   使用頻度の高いインデックスなど、DB バッファへのプリロードやバッファの固定を行います。

2　パラメータごとの反映のタイミングについては、別紙1をご覧ください。

項番	チューニング機能	PostgreSQL	Oracle	SQL Server	DB2
1	自動パラメータチューニング	△ （pgtune）	○	○	○
2	Hint 情報	△ （pg_hint_plan）	○	○	○ （最適化プロファイル）
3	統計情報のセーブ・ロード	△ （pg_dbms_stats ）	○	○	○ （db2look -m）
4	バッファ固定	×	○	×	○

*表 2.1: **チューニング機能差異　（○…標準機能、△…追加機能により対応、×…非対応）***

3. 設計（パラメータ設計）

データベースでは、テーブル定義等を含む論理設計と DBMS の運用環境を設計する運用設計を行います。本章では、データベースの論理設計は既存システムで完了していることを前提とし、運用設計の1つである PostgreSQL および OS のパラメータ設計についての方針を記述します。

3.1. PostgreSQL パラメータ設計

PostgreSQL のパラメータ設定は、$PGDATA/postgresql.conf にて行います（$PGDATA はデータベースクラスタのパス）。本節では、postgresql.conf において設定可能なパラメータのうち、PostgreSQL のチューニングという観点で検討するべきパラメータを挙げ、可能な限り算出方針について提示します。

3.1.1. postgresql.conf の記載形式について

postgresql.conf において各設定値は「パラメータ名=設定値」の形式で記載されています。initdb コマンド（データベースクラスタ初期化）を行った後、$PGDATA に postgresql.conf が設置されます。PostgreSQL はこの段階でほぼ全ての設定値がファイル内に記載されています（全てコメントアウトされ、初期値で動作しています）。記載されていない設定値は、開発者向けのオプションや、追加モジュール向けの設定値となり、これらの多くは PostgreSQL の動作状態の管理（統計情報の詳細出力など）を行うために設定するもののため、本節では紹介しません。

なお、postgresql.conf の設定値は上から順に読まれますので、同一のパラメータ名に対する設定を複数回記述した場合は、最後の値が設定されます（上書き）。

以下に postgresql.conf における設定パラメータの一つである「listen_addresses（サービス IP アドレス待ち受け設定）[3]」を例として挙げ、記載内容について説明します。

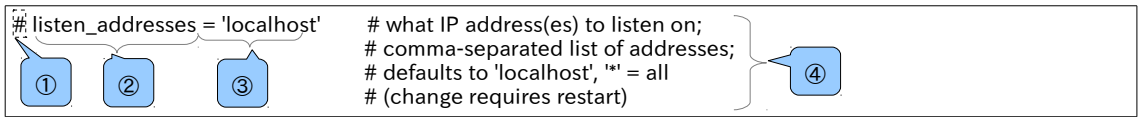


図 3.1: パラメータ参考例（listen_addresses）

3 PostgreSQL9.3 文書 18.3. 接続と認証:
http://www.postgresql.jp/document/9.3/html/runtime-config-connection.html

表 3.1: 設定値の記載構成

No.	対象	説明
①	#	“#” 以降に記述された同行の文章は全てコメントとして扱われます。 postgresql.conf の初期状態は全ての設定値がコメントアウトされています。コメントアウトされたパラメータは初期値が採用されるため、設定を行っていない場合は、コメントで記載されいる値が初期値として利用されています。 ※SET コマンドを利用してトランザクション毎などで一時的に設定可能なパラメータもあります。注意してください。
②	listen_addresses	パラメータ名です。
③	'localhost'	パラメータに対する設定値です。設定値は、論理値（bool）、整数、浮動小数点、文字列、列挙型のいずれかの形式です。本例では文字列が設定されています。 多くの場合は、許容する形式がコメントとして付記されていますが、詳細を知りたい場合は、PostgreSQL のドキュメントに記載されていますので、そちらを参照[4]してください。
④	# what IP address(es) to listen on; # comma-separated list of addresses; # defaults to 'localhost', '*' = all # (change requires restart)	パラメータに対するコメントです。設定可能範囲や、設定方法が記載されています。 ※本例の最後にある (change requires restart)という記載があるパラメータは設定の反映に再起動が必要です。記載がない場合は、pg_ctl reload による再読み込みまたは SET コマンドによる一時的な設定が可能です。

パラメータ設定方針を挙げるに当たり、本節では postgresql.conf のカテゴリ分けに従い列挙を行っています。postgresql.conf のカテゴリ分けは、以下のように postgresql.conf においてコメント文で区切られた範囲を指します。インターネット上にアップロードされた PostgreSQL 文書[5]の記載もこのカテゴリ分けに従って記載されており、日本 PostgreSQL ユーザ会による和訳[6]もそれに準じています。本節では PostgreSQL 文書の和訳のカテゴリ名を利用して記載を進めます。

```
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------
```

図 3.2: postgresql.conf のカテゴリ CONNECTIONS AND AUTHENTICATION の例（PostgreSQL 文書では「接続と認証」）

4 PostgreSQL 9.3 文書 第 18 章サーバの設定: http://www.postgresql.jp/document/9.3/html/runtime-config.html
5 PostgreSQL 9.3 文書: http://www.postgresql.org/docs/9.3/interactive/index.html
6 PostgreSQL 9.3 文書（和訳）:http://www.postgresql.jp/document/9.3/html/index.html

3.1.2. 接続と認証

接続受付ポートや、認証方式についての設定を行います。チューニングを検討するパラメータについて以下に記載します。

表 3.2: パラメータ(接続と認証)

No.	パラメータ名	初期値	説明
1	max_connections (まっくす・こねくしょんず)	100	PostgreSQL への接続を許可するクライアント接続数を設定します。通常、アプリケーションの利用者数や接続数の要件に従い設定を行いますが、次節「資源の消費」で記載しているように、PostgreSQL が利用するメモリ量に影響します。ハードウェアリソースが限定的な場合は、アプリケーション側で接続数を制御し、本値を抑制するなどの考慮が必要です。
2	superuser_reserved_connection (すーぱゆーざ・りざーぶど・こねくしょん)	3	PostgreSQL のスーパユーザ(DBMS 管理者)が優先的に利用できるクライアント接続数を指定します。本値は、max_connections の値を最大値として設定でき、アプリケーションが通常利用するクライアント接続数を減らして利用するパラメータとなります。 運用中に緊急で利用するような DBMS 管理者の数を設計し値を検討し、本値を踏まえた上で、max_connections を設定してください。

3.1.3. 資源の消費

PostgreSQL の処理(ソートやキャッシュ)で利用するメモリ量や、VACUUM 処理等のメンテナンス処理で利用するメモリ量を主に設定を行います。設計すべき、パラメータのチューニング方針を以下に記載します。

PostgreSQL では以下の計算式で算出されるメモリを利用します(max_connections 以外のパラメータは後述の節で解説します)。

```
利用メモリ量 = shared_buffers + wal_buffers
             + { ( work_mem + temp_buffers ) * max_connections }
             + { maintenance_work_mem * ( autovacuum_max_workers + メンテナンスユーザーセッション[7])}
```

図 3.3: PostgreSQL の利用メモリ計算式

7 メンテナンスユーザーセッション: VACUUM や REINDEX を実行するような DBMS 管理者のセッションを指す

表 3.3: パラメータ（資源の消費）

No.	パラメータ名	初期値	説明
1	shared_buffers （しぇあーど・ばっふぁーず）	128MB	データベース全体で利用する共有バッファを設定します。本値を大きくすることにより、パフォーマンスは向上しますが「チェックポイント処理」（共有バッファとストレージの同期処理）の処理負荷が向上する可能性があります。このため、むやみに値を大きくすることは推奨できません。 PostgreSQL 文書では 1GB のメモリを搭載したコンピュータを例に取り、25%としていますが、現在のコンピュータスペックでは本例は適応され難いスペックとなっています。OS や PostgreSQL 以外のサービスが消費するメモリを考慮して設定してください。
2	temp_buffers （てんぷ・ばっふぁーず）	8MB	セッション中で作成する一時テーブル用のメモリ量を指定します。本値で設定したメモリは直ちに確保されずに一時テーブル利用時にのみ、設定値を最大として必要な分だけ確保されます。 一時テーブルを利用する場合は、アプリケーション仕様における最大値から検討を始め、ハードウェアのメモリ総量を踏まえて決定してください。 全てのセッションで一時テーブルを作成すると、最大で max_connections * temp_buffers 分のメモリを確保しますので注意してください。
3	work_mem （わーく・めむ）	1MB	セッション中に利用する、ストレージに書き込む前の並び替え（ORDER By 等）とハッシュテーブル操作（副問い合わせ等）において利用するメモリ量を指定します。本値も temp_buffers と同様にメモリは直ちに確保されず、上述の操作が発生した際に設定値を最大として、必要な分だけ確保されます。 本値の指定は、アプリケーションで想定される最も問い合わせ件数が多い処理について、explain コマンドを用いた走行試験を行い、必要とされる最大の値の検討を行います。詳細は「4.2.4 ワークロード全体のスループットを向上させる」を参照してください。
4	maintenance_work_mem （めいんてなんす・わーく・めむ）	16MB	VACUUM や REINDEX、ALTER TABLE といったメンテナンス操作に利用するメモリ量を指定します。 以下のようなログが VACUUM コマンド実行時等に出力される場合は、本値が不足しているので増加を検討してください。 ログ例: WARNING: exceeded maintenance_work_mem while vacuuming relation "public.hoge" HINT: Consider increasing maintenance_work_mem VACUUM
5	vacuum_cost_delay （ばきゅーむ・こすと・でぃれい）	0	VACUUM 実行時において、VACUUM 処理を行う際のコスト（処理サイズや件数）において、一定の閾値（vacuum_cost_limit）を超えた場合に処理を遅延させる時間（ミリ秒）を設定します。自動 VACUUM 処理において、ストレージ装置への負荷が極地的に増大し、アプリケーションへの応答が遅延する場合は、本値の設定を検討してください。
6	bgwriter_delay （びーじーらいたー・でぃれい）	200	PostgreSQL が共有バッファから、実際にストレージ装置へデータベース情報（直近の更新情報）を書き込む際に動作する、「バックグラウンドライタ」の制御パラメータです。バックグラウンドライタは、幾つかの書き込み処理開始閾値に基づき、動作を断続的に行います。本値は書き込み処理終了後に、次の書き込み処理を行うまでの感覚を設定します。 チェックポイント処理のストレージ装置への負荷が高い場合は、本値の減少させ、バックグラウンドライタに、より多くの処理をさせることを検討してください。

7	effective_io_concurrency （えふぇくていぶ・あいおー・こんかれんしー）	1	PostgreSQL が利用しているストレージ装置の数と一致することで、パフォーマンスが向上する可能性があるパラメータです。 Bitmap Heap Scan（詳細は「4.2.3 SQL 文のレスポンスを向上させる」参照）が処理性能上のボトルネックとなっている場合は、本値の設定を検討してください。

3.1.4. ログ先行書き込み（WAL）

WAL に関する設定を行います。チューニングを検討するパラメータについて以下に記載します。

表 3.4: パラメータ（ログ先行書き込み WAL）

No.	パラメータ名	初期値	説明
1	wal_level （わる・れべる）	minimal	PostgreSQL の WAL 記録レベルを設定します。WAL の記録レベルは、PostgreSQL の停止するイベントによって利用できるレベルが異なります。初期値である”minimal”はスタンドアローンの PostgreSQL がクラッシュした場合等に、データベースを直近の状態に回復できる WAL ファイルを残すレベルとなります。アーカイブログ運用を行う場合は”archive”、ストリーミングレプリケーションを用いて、スタンバイサーバを利用する場合は”hot_stanby”を設定してください。
2	fsync （えふしんく）	on	メモリ上のデータベース情報がストレージに確実に書き込まれたか否か判定するオペレーティングシステム同期関数（fsync）利用是非の設定値になります。 off にした場合、メモリ上のデータベース情報を PostgreSQL はオペレーティングシステムに対して書き込み要求を行いますが、ストレージに書き込むタイミングはオペレーティングシステム任せ（非同期）となり、バックアップや緊急停止時のデータベースの回復是非が不明瞭になります。off にすることにより性能向上が見込めますが、データベースの用途が一時的な情報保持で特に情報揮発しても問題ないような場合以外では、基本的に off にするべきではありません。 また、本値を on としても、ストレージ装置に取り付けられたキャッシュ機構等により、データベース情報がストレージに書き込まれたか曖昧な場合もあります。取り扱うストレージ装置の仕様を確認して情報書き込みのタイミングについて検討してください。
3	synchronous_commit （しんくろなす・こみっと）	on	コミット命令の成功通知をクライアントに返却する際の「WAL 書き込みを待つか、待つ場合はどのように待つか」を設定します。 有効な値は、 ・on（マスタ、スタンバイのストレージ WAL 書き込み待ち） ・remote_write （マスタのストレージ WAL 書き込み待ち、スタンバイの WAL 転送待ち） ・local（マスタのストレージ WAL 書き込み待ち） ・off（待たない） の 4 つなります。ストリーミングレプリケーションで構成している場合に、”on”とすると、レプリカとの情報整合性が保たれますが、性能上問題が発生する場合があります。スタンバイの仕様（マスタの情報整合性の是非）を確認して本値を設計すると良いでしょう。
4	wal_sync_method （わる・しんく・めそっど）	fdatasync （linux）	fsync を on にした際の同期方法（関数）を設定します（off の際は利用されません）。本値で設定可能な値はオペレーティングシステムに依存します。設定するべき値はオペレーティングシステムに依存するため、オペレーティングシステムのシステム関数の見識を持たない場合、机上による設計で本値を選ぶことは困難です。 幸いなことに、PostgreSQL には本値を選択する際に参考となる診断ツール「pg_test_fsync[8]」が存在します。このモジュールの走行結果で最も高速な関数を選択することが一つの解になると考えられます。

8 PostgreSQL 9.3 文書 pg_test_fsync : http://www.postgresql.jp/document/9.3/html/pgtestfsync.html

5	full_page_writes （ふる・ぺーじ・らいつ）	on	PostgreSQL は「チェックポイント処理」と呼ばれるメモリの内容とストレージの内容を同期化する処理があります（チェックポイントは処理を実施した論理的位置を指します）。本値を”on”にするとチェックポイント処理の後に、まだチェックポイント処理前で同期化されていない情報を WAL バッファに追加してストレージに保存するようになります。これは、PostgreSQL の突然の停止時に備え、データベースの状態を保全する為です。 本値を”off”にするとパフォーマンスは向上しますが、データベース情報が損失する可能性があります。このため、データベースの利用に合わせて”off”にするか判断してください。通常の利用は”on”になります。
6	wal_buffers （わる・ばっふぁーず）	-1 （shared_buffers の 1/32）	ストレージ装置に書き込まれていないデータベース情報（WAL バッファ）を記憶するメモリ量を設定します。本値の下限は 32KB であり、下回る値を設定しても 32KB として扱われます。本値の最大値は 16MB です。初期値の「-1」を設定した場合も最大値は 16MB となります。本値は共有バッファ（shared_buffers）より確保する値です。 過去のコンピュータリソースにおいてメモリは高価なものでしたが、現状のコンピュータリソースではそれほど高価ではありません。本値は初期値の自動設定のまま（大抵の場合 16MB）で特に問題はありません。
7	commit_delay （こみっと・でぃれい）	0	トランザクション終了時に、WAL バッファをストレージ装置へ書き込む際に、同時に書き込めるコミット数（後述の”commit_siblings”）まで、待つ際にどの程度待つか設定します（マイクロ秒）。初期値は待ちません。 トランザクション数が多い PostgreSQL では本値とコミット数を適切に設計することで、ストレージ装置の書き込み回数を短縮できる可能性があります。
8	commit_siblings （こみっと・しーぶりんぐず）	5	前述の“commit_delay”と共に用いる設定値です。同時にストレージ装置へ書き込むコミット数を定義します。
9	checkpoint_segments （ちぇっくぽいんと・せぐめんと）	3	WAL が記載されたファイルを「セグメントファイル」と PostgreSQL では呼称します。本設定値はセグメントファイルが幾つ保存されたら、ストレージ装置へ書き込む（チェックポイント処理を行うか）を設定します。トランザクション数が多いシステムでは、ストレージ装置への書き込みが頻発し、ログ例の様な出力される場合があります。この場合は、ストレージ装置によっては、深刻な負荷となっている可能性がある為、注意が必要です（ストレージ装置のスペック上問題ない場合もあります）。本値を向上させることでストレージ装置の書き込み回数を短縮できる可能性があります。しかし、本値を大きくするとクラッシュした際のリカバリ時間が大きくなるというデメリットがあります。 ログ例: LOG: checkpoints are occurring too frequently (20 seconds apart) HINT: Consider increasing the configuration parameter "checkpoint_segments".
10	checkpoint_completion_target （ちぇっくぽいんと・こんぷりえーしょん・たーげーっと）	0.5	チェックポイント処理の完了目標時間を設定します。初期値は、次のチェックポイントが始まる時間の半分の時間を完了時間として設定しています。最大値は「1.0」となります。 チェックポイント処理によって、ストレージ装置書き込みの負荷が大きくなる場合は、本値を大きくすることで負荷を分散させることができますが、本値を大きくするとクラッシュした際のリカバリ時間が大きくなるというデメリットがあります。 本値の設計には、設定値”log_checkpoints”を「on」にすることで出力される次のようなログが役に立ちます。 ログ例: 2014-03-20 14:22:12 JST LOG: checkpoint starting: time 2014-03-20 14:25:25 JST LOG: checkpoint complete: wrote 1233 buffers (50.4%);

3.1.5. レプリケーション

PostgreSQL に組み込まれているレプリケーション機能（ストリーミングレプリケーション）に関する設定を行います。チューニングを検討するパラメータについて以下に記載します。

表 3.5: パラメータ（レプリケーション）

No.	パラメータ名	初期値	説明
1	synchronous_standby_names （しんくろなす・すたんばい・ねーむず）	-	同期レプリケーションそ行うスタンバイの名前の設定（設定値”application_name”の値 ）を行います。すべてのスタンバイで同期レプリケーションを行う場合は、「*」を設定します。本値を設定しない場合は、スタンバイサーバ非同期レプリケーションでマスタと通信します。 マスタの同期レプリケーションによる処理遅延を懸念する場合は、必要最低限の同期対象の名前のみ設定してください。

3.1.6. 問い合わせ計画

実行計画を立案するオプティマイザの振る舞いに関する設定を行います。本値の設定については、「4.1.1 EXPLAIN/EXPLAIN ANALYZE による SQL 文の実行計画の取得」を参照してください。

3.1.7. 自動 Vacuum 作業

自動 VACUUM に関する設定を行います。チューニングを検討するパラメータについて以下に記載します。

表 3.6: パラメータ（自動 Vacuum 作業）

No.	パラメータ名	初期値	説明
1	autovacuum_max_workers （おーとばきゅーむ・まっくす・わーかーず）	3	同時に実行する自動 Vacuum プロセスの数を指定します。 次のようなログが出力され、自動 Vacuum がプロセス数の制限により遅延するようなら、設定値を検討してください。 ログ例: LOG: maximum number of autovacuum workers reached HINT: Consider increasing autovacuum_max_workers (currently 3).

3.1.8. ロック管理

ロックに関する設定を行います。チューニングを検討するパラメータについて以下に記載します。

表 3.7: パラメータ（ロック管理）

No.	パラメータ名	初期値	説明
1	deadlock_timeout （でっどろっく・たいむあうと）	1s （1 秒）	SQL がデッドロックしているかどうかの検査を開始するまでの「ロック待ち時間」を指定します。この検査処理はサーバに負荷をかける可能性がありますので、トランザクションが必要とする最大（ロック）時間を設定することが理想です。

3.2. ハードウェア／OS設定

DB へのアクセスパターンや負荷状況を考慮し、OSおよびハードウェア構成を設計します。考慮すべきポイントは、リソースの許容範囲の拡大と負荷の分散です。

3.2.1. カーネルパラメータ

Linux カーネルのパラメータ設定により、PostgreSQL が使用するシステムのリソースの制限を変更することができます。システム全体でオープン可能なファイル数や接続数に応じたセマフォの全体数、共有メモリサイズなど、規模に応じたシステムリソース容量の設定を行います。カーネルパラメータの設定値については、マニュアル[9]を参考にしてください。

カーネルパラメータ	説明	関連する PostgreSQL のパラメータ
fs.file-max	システム全体で使用可能なファイルディスクリプタ数を指定する。PostgreSQL のバックエンドプロセスが同時にオープンするファイル数の設定と最大接続数から、PostgreSQL が要求するファイルディスクリプタ数を確認することができる。	**max_files_per_process** バックエンドプロセスが要求する最大ファイルディスクリプタ数。 **max_connections** DB への最大接続数。
kernel.sem	セマフォに関する設定を行う。 設定する項目には、セットごとのセマフォの最大数（SEMMSL）、システム全体のセマフォの最大数（SEMMNS）、セマフォ識別子の最大数（SEMMNI）、セマフォマップの中の項目の数（SEMMAP）がある。	**max_connections** DB への最大接続数。 **autovacuum_max_worker** 自動 VACUUM の最大同時実行数。
kernel.shmmax	共有メモリセグメントの最大サイズ（SHMMAX）、使用可能な共有メモリの総量（SHMALL）を指定する。 PostgreSQL9.3 移行は DB バッファ領域が mmap に変更され、必要な共有メモリサイズは大幅に削減されている。	**max_connections** DB への最大接続数。 **max_locks_per_transaction** トランザクションあたりに取得可能な最大ロック数。 **autovacuum_max_workers** 自動 VACUUM の最大ワーカープロセス数。 **max_prepared_transactions** 二相コミット用の最大トランザクション数 **shared_buffers** DB バッファサイズ **wal_buffers** WAL バッファサイズ **block_size** ブロックサイズ
net.ipv4.ip_local_port_range	TCP/IP の送信時に使用するポートの範囲を指定する。	**max_connections** DB への最大接続数。

表 3.8: カーネルパラメータ

3.2.2. I/O 分散

アプリケーションが複数セッションから DB サーバに対して複数セッションの同時アクセスを行ったり、参照処理と更新処理が混在する処理を行うと、I/O ウェイトによるボトルネックとなる場合があります。これを改善するため、データベースを構成するファイルやオブジェクトの格納策を物理的に分割し、I/O 負荷の分散を図ります。

① WAL 出力先の変更

更新情報の出力先である WAL ファイルをデータベースの格納ボリュームと分割することにより、更新処理における I/O ウェイトが改善する可能性があります。WAL ファイルの出力先、以下の何れかの

9 PostgreSQL 9.3 文書 第 17 章サーバの準備と運用：
http://www.postgresql.jp/document/9.3/html/kernel-resources.html

方法で設定することができます。

- DB が停止している状態で“$PGDATA/pg_xlog”を別ボリュームに移し、移動先のディレクトリに対するシンボリックリンク“$PGDATA/pg_xlog”を作成する
- DB の初期化（initdb[10]）時に”--xlogdir”パラメータを指定し、WAL の出力先を変更する。

checkpoint_segments の設定値が小さくチェックポイントが頻発している場合、ブロックイメージ出力による WAL 出力量が増大するため、チェックポイント関連のパラメータを合わせて検討することを推奨します（表 3.4 パラメータ（ログ先行書き込み WAL））。

② テーブルスペース

データベースやテーブル、インデックスのテーブルスペースを分散することにより、I/O ウェイトが改善する可能性があります。以下のように、異なるボリュームにテーブルスペースを作成し[11]、I/O を分散させることを検討するよう推奨します。

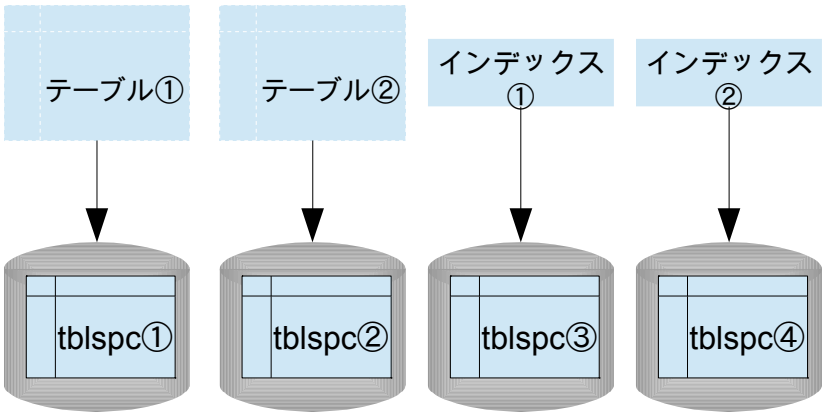


図 3.4: テーブルスペース

③ テンポラリ領域

ソート処理などの実行中に指定したメモリサイズの上限（work_mem）を超える領域を必要とする場合にはファイルを使用します。テンポラリファイルの出力先は、デフォルトテーブルスペースと同じディレクトリなので、既定の抽出レコード件数が多くなるとテンポラリファイルへの I/O がデータベースアクセスと競合する可能性があります。

テンポラリファイル用のテーブルスペースを確保し、temp_tablespaces を設定することで、データベースファイルとの I/O 競合が改善する可能性があります。当該パラメータは、データベースセッションごとに指定することができます。

3.2.3. I/O 性能改善

データサイズの拡大や大量のトランザクション実行は I/O の要求回数の増加につながる場合があり、I/O 性能がシステム全体の性能を決定することがあります。

I/O を分散するほか、高速なストレージの利用や SSD を利用し、I/O 性能そのものを向上させることにより、大幅な性能改善を達成する場合があります[12]。

3.2.4. CPU（Core）

PostgreSQL9.2 では参照系の処理が 64core までリニアにスケールする性能測定結果が公開されています。

適切なサーバサイジングにより、サーバのスケールアップによる参照系のスループット向上の可能性があります[13]。

10 PostgreSQL 9.3 文書：http://www.postgresql.jp/document/current/html/app-initdb.html
11 PostgreSQL 9.3 文書：http://www.postgresql.jp/document/9.3/html/sql-createtablespace.html
12 詳細は 2013 年度 PGECons WG1 の SSD 性能検証をご覧ください。
13 2012 年度 PGECons WG1 成果物：https://www.pgecons.org/wp-content/uploads/2013/06/WG1-2012.pdf

3.3. PostgreSQLビルド時の設定

PostgreSQL のブロックサイズや 1GB を超えるテーブルのファイル分割サイズ（テーブルセグメントサイズ）は、PostgreSQL のビルド（configure）実行時に決定します。

ブロックサイズやテーブルセグメントサイズの変更により、レコードサイズやカラム数の制限を拡大したり、レコードの格納効率改善、文字列の圧縮による CPU 負荷低減、TOAST への更新による VACUUM の負荷削減により、データベース性能が改善する場合があります。

パラメータ名	意味
--with-segsize	リレーションの実態であるファイルを分割するサイズを指定する。既定値では 1GB 毎のファイルに分割して格納されます。大容量データを格納する DB ではファイル数が膨大になる可能性がある。1GB 以上のサイズを指定することにより、同時にオープンするファイルディスクリプタ数を削減することができる。
--with-blocksize	データベースを構成するブロックサイズを指定する。既定値は 8KB であり、最大で 32KB を指定することができる。1 レコードの列数が多い場合や TOAST によるレコードの圧縮・分割の閾値を変更することで、性能特性を変更することができる。
--with-wal-segsize	WAL ファイルのサイズを指定する。既定値では 16MB であり、最大で 64MB まで拡大することができる。
--with-wal-blocksize	WAL への I/O 単位を指定する。規定値は 8KB でありm 64KB まで拡大することができる。

表 3.9: configure オプション

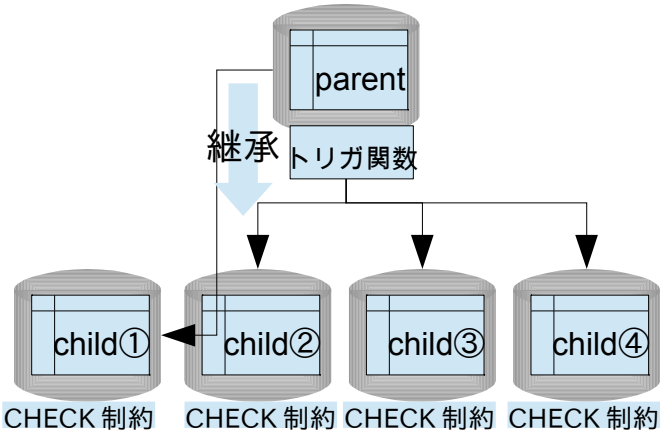
3.4. その他

3.4.1. パーティション

データ量の増大にともない、以下のような処理による負荷が増加することが予想されます。

- 検索処理
- 一定間隔で実行されるデータの削除処理
- VACUUM、ANALYZE

これに対し、テーブルをあらかじめ分割して格納するパーティショニングが性能の維持に有効な場合があります。PostgreSQL のパーティショニングは、テーブルの継承、トリガ関数、CHECK 制約を組み合わせて実現しています。適切な条件でデータを分散することにより検索処理やデータ削除、メンテナンスの対象を限定でき、処理性能が改善する可能性があります。



一方、パーティション表に対するデータ挿入はトリガ関数を使用するためオーバヘッドが大きく、応答時間が求められる更新処理や大量データロードには向きません。また、検索においてもパーティション数の増加にともない検索プラン作成時の CHECK 制約による対象パーティションの絞り込みによる性能劣化が大きく、PostgreSQL のマニュ

アルにおいても 100 パーティションが限度とされています。

3.4.2. インスタンスの分割

WAL など分散することができない PostgreSQL のリソースにおいて競合が発生し、ボトルネックとなる場合があります。大量更新かつ高可用性も求められるシステムでは、synchronous_commit や commit_delay など 可用性を犠牲にした WAL 出力設定（表 3.4 パラメータ（ログ先行書き込み WAL））による性能改善を行うことができないため、DB インスタンスの分割により WAL 出力の競合を回避したり、場合によっては PostgreSQL を運用するサーバの分割も含めて検討が必要となる場合があります。

3.4.3. レプリケーションによる参照負荷分散

検索リクエスト主体のシステムにおいては、検索処理の負荷を分散するため、レプリケーションによる検索用 DB ノードの追加が有効な場合があります。PostgreSQL のログシッピングレプリケーションには以下のような制約もありますが、特徴を理解した上でレプリケーションによる複数のノードの検索負荷分散が性能改善に有効な場合があります。

- PostgreSQL のログシッピングレプリケーションは、同期モードであってもトランザクションレベルでの同期を保証していません。
- レプリカのノードを複数台立てる場合、同期レプリケーションを構成するノードは 1 台のみであり、2 台目以降のノードは非同期でのレプリケーションとなります。

4. 実装、構築(テスト環境)

4.1. アプリケーションが利用する SQL の試行

本章では、アプリケーション内の SQL 文の最適化を実施する際の基礎となる内容を説明します。

4.1.1. EXPLAIN/EXPLAIN ANALYZE による SQL 文の実行計画の取得

SQL 文の実行計画を取得するためには、EXPLAIN 文を使用します（EXPLAIN ANALYZE 文を使用すると、実際に SQL 文を発行します）。

実行計画は、SQL 文の最適化の際の手がかりとなります。以下に psql を使用して実施例を示します。

```
$ psql -h serverName]  -p [portNum] testdb
psql (9.3.2)
"help" でヘルプを表示します.

testdb=# EXPLAIN UPDATE pgbench_accounts SET abalance = abalance + 1 WHERE aid = 12;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Update on pgbench_accounts  (cost=0.00..8.38 rows=1 width=103)
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts  (cost=0.00..8.38 rows=1 width=103)
         Index Cond: (aid = 12)
(3 行)
```

この例では、テーブル pgbench_accounts に定義したプライマリキー pgbench_accounts_pkey を使用して、インデックスを使用した問い合わせが実行されることを示しています。

SQL 文の最適化の手順、実行計画の詳細は、「4.2 SQL の最適化」および「PostgreSQL 9.3.2 文書 EXPLAIN」を参照してください。

4.1.2. 起動パラメータ(メモリ、ロギング項目など)の見直し

SQL 文のレスポンス向上を目的に、必要に応じて PostgreSQL の起動パラメータの見直しを実施します。

パラメータチューニングの詳細は、「3.1 PostgreSQL パラメータ設計」を参照してください。

4.2. SQL の最適化

本章では、SQL 文の最適化の作業の流れ、および具体的な手法について説明します。

4.2.1. PostgreSQL における SQL 処理について

PostgreSQL における SQL クエリ処理は、プランナ/オプティマイザで作成されるクエリの実行計画を元に最適化することができます。

プランナ/オプティマイザ

PostgreSQL では、SQL 文の問い合わせに対しプランナ/オプティマイザにより最適な実行計画を作成します。プランナ/オプティマイザは、問い合わせにより取り出すディスクのページ数を推定し、コストと呼ばれる指標を比較することで、最適な実行計画を選択して、SQL 文の実行基盤(エグゼキュータ)に処理を渡します。

プランナ/オプティマイザ、コストの詳細は、PostgreSQL 9.3 文書「第 14 章性能に関するヒント[14]」を参照してください。

アクセスパスの概要

実行計画を構成するアクセスパスには、主に以下のものが存在します。

表 4.1: アクセスパスを構成する要素

種類	名称	概要/関連する SQL 文要素
スキャン	Seq Scan	インデックスを使用せず、全件の検索処理を行います。
	Index Scan	インデックスを使用してスキャンを行います。
	Index Only Scan	問い合わせがインデックスに含まれるカラムのみで完結する場合、テーブルへのアクセスをせずに結果を返すスキャン（Index Only Scan）を行います。

14 PostgreSQL 9.3 文書：http://www.postgresql.jp/document/9.3/html/performance-tips.html

種類	名称	概要/関連する SQL 文要素
		(PostgreSQL 9.2 以降で使用されます)
	Bitmap Index Scan Bitmap Heap Scan	ビットマップを使用したスキャンを行います。 WHERE 句に AND/OR 条件を指定した場合、絞り込みを効率化します。
結合	Nested Loop Join	ネステッド・ループ結合を行います。
	Sort Merge Join	ソート・マージ結合を行います。
	Hash Join	ハッシュ結合を行います。
その他	Sort	ソート処理を行います(ORDER BY 句)
	Hash Aggregate	ハッシュ表を作成してから集約関数を実行します。
	Group Aggregate	ソートしてから集約関数を実行します。

4.2.2. チューニング目標の設定

チューニングによって達成したい目標を明確にします。例えば、以下のような目標です。

- SQL 文のレスポンスを xx 秒以内・1/xx に削減する
- ワークロード全体のスループットを xx 倍・xxMB/s に向上させる

4.2.3. SQL 文のレスポンスを向上させる

SQL 文のレスポンスを向上させるために、以下のポイントでボトルネックとなっている SQL 文を検出します。

- 時間のかかる SQL 文の検出
  postgresql.conf の log_min_duration パラメータを使用し、指定した時間よりも実行に時間のかかった SQL 文をログに記録します。
- 実行回数の多い SQL 文の検出
  pg_stat_statements を使用することで、サーバで実行された SQL 文の実行回数を記録することができます。実行回数の多い SQL 文の実行時間を改善することで、レスポンスの向上を狙います。
  実行回数の多い順(デフォルトでは上位 1,000 件、変更可能)が保持されているため、投入されている SQL 文全体の傾向を常に示しているわけではないことに注意が必要です。

EXPLAIN の読み方

前述のツールを使用して、チューニングを実施する SQL 文を特定できたら、EXPLAIN ANALYZE を使用して、想定したアクセスパスを使用してクエリが実行されたかどうかを確認します。(適切にインデックススキャンが使われているかどうか、など)

その後、アクセスパスの改善に向けて、以下を実行します。

- インデックスの定義
- SQL 文の見直し(問い合わせ条件、集計対象の精査など)
- 必要に応じて、問い合わせ時のバッファの使用状況(EXPLAIN 文の BUFFERS 句を使用)

その他の観点、アクセスパスの改善方法の詳細は、Let's PostgreSQL「スロークエリの改善」を参照してください。

オプティマイザの制御

想定したアクセスパスを使用する実行計画が得られない場合は、以下のような点の見直し、およびチューニングを実施し、オプティマイザの動作を制御します。

- 得られた統計情報の確認
  はじめに、得られた統計情報を以下の観点で確認します。
  - プランナが推定したレコード数と、実際のレコード数の確認
    両者に乖離がある場合は、プランナによる推定が正しくない場合がありますので、プランナの統計情報の更新(ANALYZE)を実施します。
- マルチカラムインデックス使用時の注意
  インデックス列に対する複数の条件を OR 句で結合した場合、そのインデックスが使用されません。例えば、インデックス(a,b)に対して WHERE a = 5 AND b = 6 や、WHERE a = 5 の条件において、インデックスが使用さ

れます。しかし、WHERE a = 5 OR b = 6 や、WHERE b = 6 のみの条件ではインデックスは使用されません。問い合わせ回数の多い条件、および当該テーブルに対する更新と検索の比率を考慮し、インデックスを作成します。詳細は「PostgreSQL 9.3 文書 11.5. 複数のインデックスの組み合わせ[15]」を参照してください。

- プランナメソッドの設定
  プランナメソッド設定によりオプティマイザを制御し使用できるアクセスパスを制限します。ただし、制限したアクセスパスが他のアクセスパスで代用できない場合には、制限したアクセスパスを使用する場合があります。以下に、プランナメソッドの設定パラメータを示します。

表 4.2: プランナメソッド設定の項目

パラメータ名	説明
enable_bitmapscan	スキャン時に Bitmap Index Scan/Bitmap Heap Scan を選択できるようにします。
enable_hashagg	集計時に Hash Aggregate を選択できるようにします。
enable_hashjoin	結合時に Hash Join を選択できるようにします。
enable_indexscan	スキャン時に Index Scan を選択できるようにします。
enable_indexonlyscan	スキャン時に Index Only Scan を選択できるようにします。 本パラメータは PostgreSQL 9.2 以降で存在します。
enable_material	問い合わせプランナの具体化の使用を有効にします。
enable_mergejoin	結合時に、Merge Join を選択できるようにします。
enable_nestloop	結合時に、Nested Loop を選択できるようにします。
enable_seqscan	スキャン時に Seq Scan を選択できるようにします。
enable_sort	Sort を選択できるようにします。
enable_tidscan	スキャン時に TID Scan を選択できるようにします。

- プランナコスト定数の設定
  プランナコスト定数をチューニングすることで、プランナが実際の実行効率を反映した実行計画をたてることができるようになります。詳細は「PostgreSQL9.3 文書 EXPLAIN の利用[16]」を参照してください。

表 4.3: プランナコスト定数

パラメータ名	説明
seq_page_cost	シーケンシャルアクセスでページを取り出す場合のコスト定数です。この値を 1.0 に設定し、他のプランナコスト定数を相対的に設定します。 データベースを配置するデバイスによっては、本パラメタと「3.3 パラメタチューニング」に記載の random_page_cost パラメタの値を調整することで、デバイスの特性に基づいた実行計画を選択することが可能になります。
cpu_tuple_cost	問合せが行を処理する場合のコスト定数です。 seq_page_cost との相対値を設定します。 デフォルトは 0.01 です。
cpu_index_tuple_cost	問い合わせがインデックス行を処理する場合のコスト定数です。seq_page_cost との相対値を設定します。 デフォルトは 0.005 です。
cpu_operator_cost	問い合わせの WHERE 句のそれぞれの演算子を処理する場合のコスト定数です。

15 PostgreSQL9.3 文書：http://www.postgresql.jp/document/9.3/html/indexes-bitmap-scans.html
16 PostgreSQL9.3 文書：http://www.postgresql.jp/document/9.3/html/using-explain.html

	seq_page_cost との相対値を設定します。 デフォルトは 0.0025 です。

- Hint 句による実行計画の固定化
  Hint 句を使用して実行計画を固定し、プランナの動作を制御します。
  これによりプランナに特定のプランナメソッドを強制することができます。
  SQL の最適化を行ったもののどうしても理想の実行計画にならない場合や、プランナに左右されずに実行計画を固定したい場合に利用します。
  Hint 句を使用できるようにするには、pg_hint_plan モジュールを使用します。pg_hint_plan は、以下からダウンロードして使用します。
  http://en.sourceforge.jp/projects/pghintplan/
  インストール方法、使用方法は、以下を参照してください。
  http://en.sourceforge.jp/projects/pghintplan/wiki/FrontPage
- 実行計画の保存、バックアップ
  pg_dbms_stats を利用すると、統計情報を固定化、バックアップすることができます。
  運用中のシステムで、統計情報が変わることによる実行計画の変化を止めたい場合などに使用します。
  pg_dbms_stats は、以下からダウンロードして使用します。
  http://sourceforge.jp/projects/pgdbmsstats/

パラメータチューニング

SQL 文の実行計画およびシステムカタログより、DB のパラメータチューニングが必要となる場合があります。
例えば、データアクセス時のバッファヒット率が低く、DB サーバの搭載メモリに余裕があり、共有バッファを拡張する場合などが挙げられます。パラメータチューニングの詳細は、「3.1 PostgreSQL パラメータ設計」を参照してください。

4.2.4. ワークロード全体のスループットを向上させる

ワークロードの全体のスループットを向上させるために、以下のポイントを確認します。

- ロック競合の確認
  システムカタログ pg_locks、pg_stat_activity を活用して、ロック待ちとなっている処理内容と対象のテーブルを確認します。
  以下のような SQL 文をシステムカタログに対して発行し、ロックの件数、ロックの種類を確認します。

```
SELECT l.locktype, c.relname, l.pid, l.mode, substring(a.current_query, 1, 30) AS query,
(current_timestamp - xact_start)::interval(3) AS duration FROM pg_locks l LEFT OUTER JOIN
pg_stat_activity a ON l.pid = a. procpid LEFT OUTER JOIN pg_class c ON l.relation = c.oid WHERE NOT
l.granted ORDER BY l.pid;
```

- 各リソースの使用率の確認
  OS のコマンドなどを利用し、リソースの使用状況を確認します。
  - データベースクラスタを配置したディスクのスループット、ディスクのビジー率
  - CPU 使用率
  - メモリの使用量、特にディスクスワップの発生有無

- 集計/結合処理における一時領域の使用状況
  EXPLAIN ANALYZE 文により、集計/結合処理が作業メモリ上で実施されるか、ディスク上にスワップするか確認することができます。以下に一例を示します。

```
GroupAggregate  (cost=19518857.85..19544437.27 rows=1023177 width=48) (actual
time=2564192.904..2774538.597 rows=34809 loops=1)
  ->  Sort  (cost=19518857.85..19521415.79 rows=1023177 width=48) (actual
time=2564177.929..2737325.302 rows=72050924 loops=1)
        Sort Key: "column1"."column2", "column3"."column4"
        Sort Method: external merge  Disk: 417974kB
…
```

上記の場合、ソート処理に 417974kB(約 400MB)の一時ファイルを作成したことが分かります。SQL 文を実行したセッションの work_mem の値の見直しを実施します。

- 不要領域の回収(VACUUM)の進行状況
  pg_stat_user_tables システムカタログを参照することで、不要領域の回収(vacuum)の進行状況を確認します。以下にシステムカタログの各カラムが示す内容を説明します。
  - n_live_tup
    有効レコードの件数を示します。
  - n_dead_tup
    無効レコードの件数を示します。全体(有効レコード+無効レコードの和)に対して無効レコードが増加すると、SQL 文のレスポンスが低下します。
  - last_vacuum
    最後に手動で VACUUM を実施した時刻を示します。
  - last_autovacuum
    最後に autovacuum が実施された時刻を示します。
    autovacuum の実施契機を変更するには、postgresq.conf の autovacuum_*パラメータをチューニングします。

- PreparedStatement 使用時のステートメントキャッシュについて
  JDBC ドライバの設定により、指定した回数以上の PreparedStatement を実行した場合に、サーバ側で解析済みの文をキャッシュし、以降はそれを使用することで解析のコストを削減しています。
  以下の方法で、準備文をキャッシュしているか確認します。
  データベース接続時の prepareThreshold パラメータにより、キャッシュされるまでの回数を指定することができます(デフォルト値は 5 です)
  - log_min_duration_statement = 0 を設定します。
  - pg_ctl reload を実行します。
  - 対象となるアプリケーションを実行します。
  - データベースのログを確認します。解析対象の文が<unnamed>から S_1 のように出力された場合には、準備文がキャッシュされています。

    [キャッシュされていない]

    ```
    LOG:  期間: 0.112 ミリ秒  解析 <unnamed> : UPDATE a SET c = 'xxx' WHERE a = $1 AND b =
    $2LOG:  期間: 0.122 ミリ秒  バインド <unnamed>: UPDATE a SET c = 'xxx' WHERE a = $1 AND b =
    $2
    ```

    [キャッシュされている]

    ```
    LOG:  期間: 0.152 ミリ秒  解析 S_1 : UPDATE a SET c = 'xxx' WHERE a = $1 AND b = $2LOG:  期間:
    0.122 ミリ秒  バインド S_1: UPDATE a SET c = 'xxx' WHERE a = $1 AND b = $2
    ```

5. まとめ

異種 DBMS からの移行においては、高速化している新しいハードウェアの恩恵を得られる場合も多くありますが、データ容量や処理すべきトランザクションが増加すると、導入したままの状態で目標性能を達成することは困難になります。
PostgreSQL の特徴を前提に、システム構成やパラメータの設定等の物理設計を行ってください。また、可能であれば事前に目標性能の実現性を検証することを推奨します。

SQL のチューニング方法に関しては、多くの DBMS と同様に PostgreSQL もコストベースのオプティマイザを搭載しており、検索プランの確認手段やプランの選択に関わるパラメータを PostgreSQL も提供しています。DBMS により収集可能な性能情報や支援ツールの差異はありますが、プランの選択が性能に大きな影響を与える点に差異はありませんので、効率的なプランが選択されるよう、パラメータの設定や SQL 文の記述の変更、統計情報のメンテナンスを行う必要があります。

また、本資料では性能チューニングと関わりの深い性能情報収集（監視）や統計情報のメンテナンスについてはあまり触れませんでした。運用関係の調査、検証を行う PostgreSQL エンタープライズコンソーシアム・WG3 の成果も合わせてご覧ください。

著者

版	所属企業・団体名	部署名	氏名
チューニング編 第1版 （2013年度 WG 2）	NECソリューションイノベーティブ株式会社	PFシステム事業部	岩浅 晃郎
	株式会社富士通ソーシアルサイエンスラボラトリ	公共ビジネス本部	青木 俊彦
			杉山 貴洋
			小山田 政紀
			高橋 勝平
	富士通株式会社	データマネジメント・ミドルウェア事業部	山本 明範
			山本 貢嗣