LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Gold
技術解説無料セミナー

2017/9/30

NTTテクノクロス株式会社
クラウド＆セキュリティ事業部
勝俣 智成

Agenda

OSS-DB

- 基礎解説
- 運用管理
  - 運用管理用コマンド全般
  - ホット・スタンバイ運用

～ 休憩 ～

- 性能監視
  - アクセス統計情報
  - クエリ実行計画

自己紹介

OSS-DB

■勝俣　智成（かつまた　ともなり）
NTTテクノクロス株式会社

■経歴
■2002年NTTソフトウェア（現NTTテクノクロス)入社。
■数年間は全文検索に関する業務を担当。
■PostgreSQLとの出会いは2004年。
■PostgreSQLに全文検索機能やXML検索機能などを拡張する開発に従事。以降、開発・国内外のPostgreSQLカンファレンスへの参加、社内外でのPostgreSQL研修の講師などを行っている。

OSS-DB技術者認定が必要な理由

OSS-DB

- 使う前に設定が必要(インストールしただけでは利用できない)
  - ユーザ
  - アクセス権
  - テーブルの作成
  - プログラムの開発
- 重要な用途
  - 基幹業務での利用
  - バックアップ
  - セキュリティ
- 複雑な用途
  - 分散DB
  - パフォーマンスチューニング
  - トラブルシューティング
- 製品による違い
  - 一般論だけ学んでも、現場で活躍できない

OSS-DB技術者認定試験の概要

OSS-DB

- 認定の種類
  - Silver(ベーシックレベル)
    - OSS-DB Exam Silverに合格すれば認定される
  - Gold(アドバンストレベル)
    - OSS-DB Silverの認定を取得し、OSS-DB Exam Goldに合格すれば認定される
- Silver認定の基準
  - データベースの導入、DBアプリケーションの開発、DBの運用管理ができること
  - OSS-DBの各種機能やコマンドの目的、使い方を正しく理解していること
- Gold認定の基準
  - トラブルシューティング、パフォーマンスチューニングなどOSS-DBに関する高度な技術を有すること
  - コマンドの出力結果などから、必要な情報を読み取る知識やスキルがあること

OSS-DB Exam Goldの出題範囲

OSS-DB

- 運用管理(30%)
  - データベースサーバ構築
  - 運用管理コマンド全般
  - データベースの構築
  - ホット・スタンバイ運用
- 性能監視(30%)
  - アクセス統計情報
  - テーブル/カラム統計情報
  - クエリ実行計画
  - スロークエリの検出
  - 付属ツールによる解析
- パフォーマンスチューニング(20%)
  - 性能に関するパラメータ
  - チューニングの実施
- 障害対応(20%)
  - 起こりうる障害のパターン
  - 破損クラスタ復旧
  - ホット・スタンバイ復旧

出題範囲に関する注意事項

OSS-DB

- 最新の出題範囲
  http://www.oss-db.jp/outline/examarea.shtml

- 前提とするRDBMSはPostgreSQL 9.0以上
  Version 9.0と9.1の違い等、Versionに依存する問題は出題されない

- 出題範囲に関するFAQ
  http://www.oss-db.jp/faq/#n02

傾向と対策

OSS-DB

- 公式ドキュメントは基本的に最新Versionを読むべき
（最も情報量が多いため）

- GUCパラメータやシステムテーブル・ビューは、単純に意味を覚えるのではなく、影響まで理解しなければならない

  例 deadlock_timeout

  意味:　ロック状態になった時にデッドロック検出処理を開始するまでの待機時間

  影響:　値を小さくすると、デッドロックの検出は早くなるが、実際にはデッドロックが発生していないのに検出処理が 動くことが多くなるため、CPUに無駄な負荷がかかる　可能性も高くなる

- 実機での動作確認は極めて重要
（予想通りに動作しなければ、何かしら理解不足があるということ）

データ構造

OSS-DB

- データ構造としてPostgreSQLの以下の点についてまとめる
  - プロセス構造
  - 利用する主なメモリ
  - データベースクラスタの構造（PGDATA）
  - データの格納方法

- プロセス構造
  - PostgreSQL起動時に起動するプロセスと接続ごとに起動するプロセスに大別される

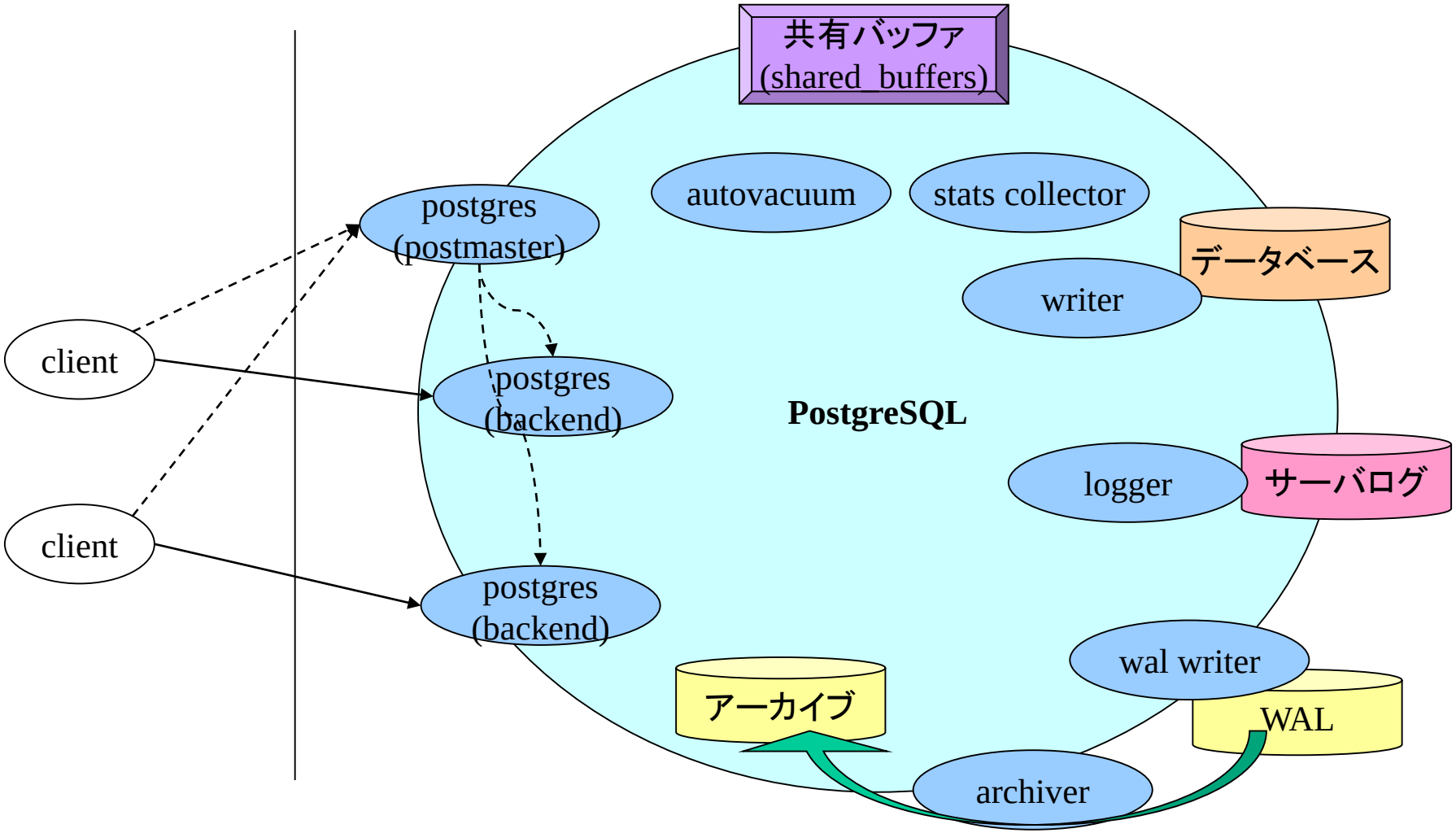
バックグラウンド・プロセス

/var/lib/pgsql-9.0/bin/postgres	・・・親玉（リスナ）
postgres: logger process	・・・サーバログ出力
postgres: writer process	・・・データ書き出し
postgres: wal writer process	・・・WAL書き出し
postgres: autovacuum launcher process	・・・VACUUM実行
postgres: archiver process	・・・WALアーカイブ
postgres: stats collector process	・・・統計情報収集

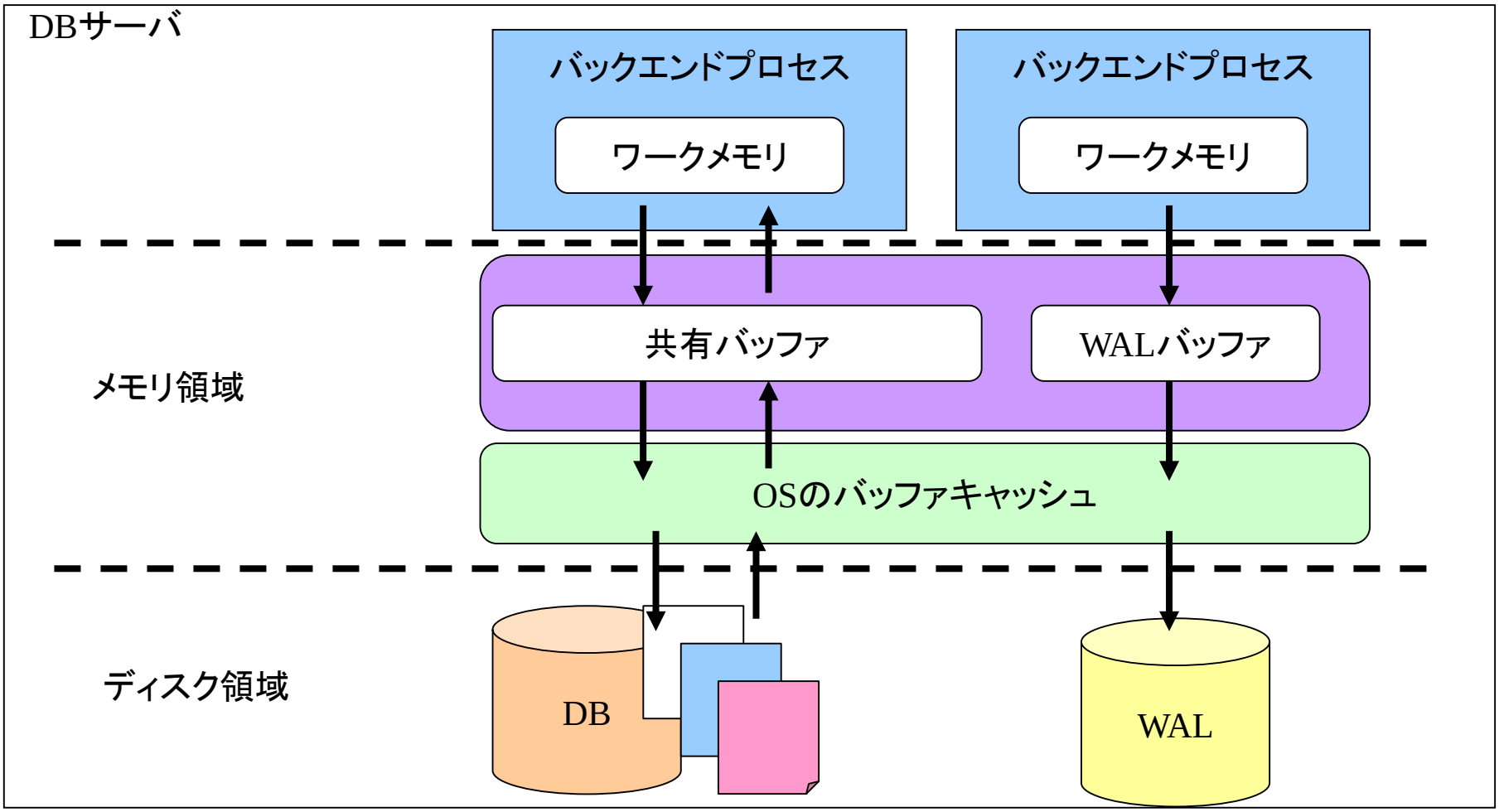
バックエンド・プロセス

postgres: katsumata a [local] idle	・・・クライアントから接続

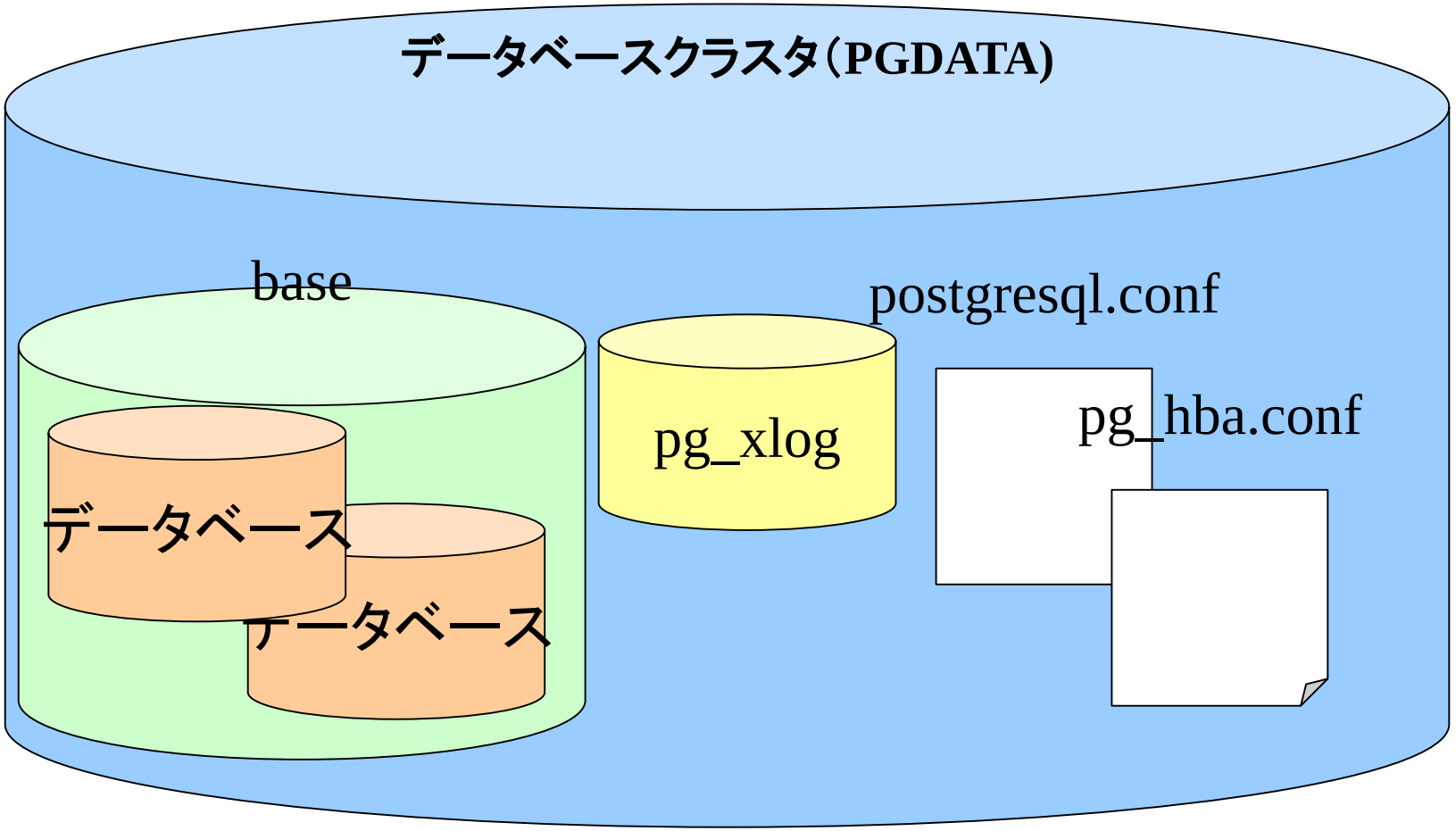
- PostgreSQLのプロセスイメージ図



- PostgreSQLが使用するメモリイメージ図



- データベースクラスタのイメージ図



データ構造

OSS-DB

- データベースのディレクトリ
  - 初期状態では「postgres」「template0」「template1」の3つのデータベースが存在する
  - それぞれのOID(ディレクトリ名)確認方法は後述
  - データベースを作成すると、「template1」のディレクトリがコピーされる

データ構造

OSS-DB

データ格納方法

- ユーザが作成したデータベースは、base配下のディレクトリとして管理される。OIDと呼ばれる、データベースクラスタで一意のIDがディレクトリ名となる
- ユーザが作成したテーブル/インデックスは、データベースのディレクトリ配下に1テーブル/1インデックスにつき1ファイルが割り当てられる。filenodeと呼ばれる、一意のIDがファイル名となる。
  - サイズが1GBを超える場合は、「XXX.1」のように分割される

テーブルファイル

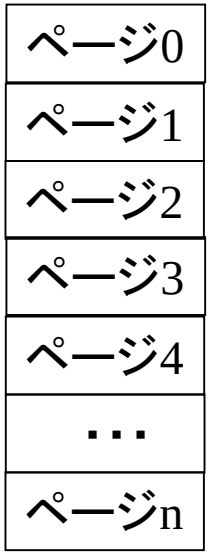
- 1ファイル最大1GB、データ量の増加にともないページ単位(8192byte)に増加していく
- メンテナンス処理によりVisibilityMapやFreeSpaceMapといったファイルも作成される
  - テーブルのファイル名が「0000」の場合の各ファイルの命名規則は以下のとおり
    - 0000_vm：VisibilityMapファイル
    - 0000_fsm：FSMファイル
- VisibilityMapファイルは、各ページの可視状態を管理し、VACUUM時に不要なページをスキャンしないようにするために利用される
- FreeSpaceMapファイルは空き領域を管理し、データ挿入時にどこに挿入するかを決める

データ構造

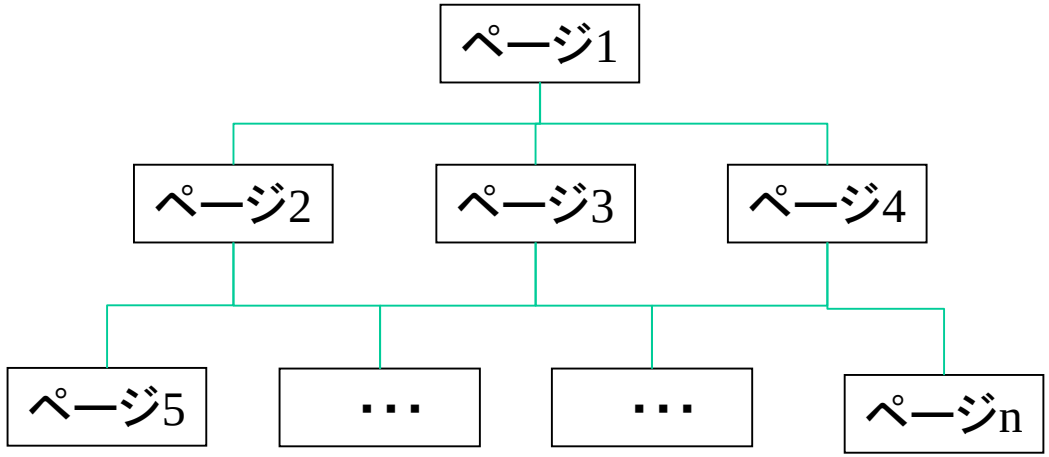
インデックスファイル

- 1ファイル最大1GB、データ量の増加にともないページ単位(8192byte)に増加していく
- 先頭の1ページはメタページとして固定。その他のページがルート、リーフページとインターナルページとして利用される

物理ファイル



内部的にはB-Tree構造



テーブル空間

- CREATE TABLESPACE文で${PGDATA}/base以外の領域(ディレクトリ)をデータ保存先にすることができる
  - CREATE TABLESPACE <テーブル空間名>
    LOCATION '<ディレクトリ>';
- 定義したテーブル空間にはOIDが割り当てられ、${PGDATA}/pg_tblspc配下に実際の格納先を指したシンボリックリンクとして配置される
- データを別のデバイスに配置しI/O効率をよくしたい場合などに有効
- オンラインバックアップ時にはすべてのテーブル空間のデータもバックアップ対象にすることを忘れないこと！

容量監視

OSS-DB

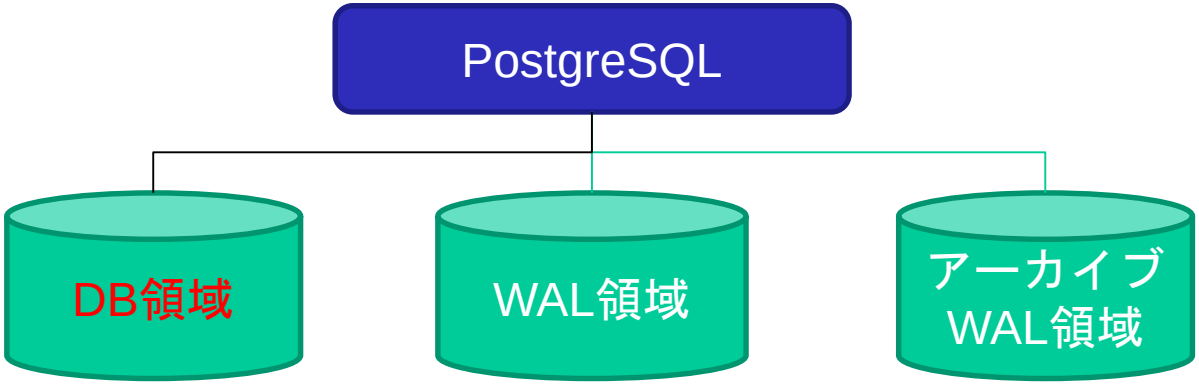
ディスク容量監視

- PostgreSQLが正常に動作していることを確認する観点のひとつとしてディスク容量の監視がある
- 監視すべき主な領域には、以下がある
  - データベース領域
  - WAL領域
  - アーカイブWAL領域
- それぞれの領域に、どのようなファイルがどの程度作成されるのかを見積もった値をベースとし、運用中の実測値を監視する

容量監視

データベース領域のディスク容量監視

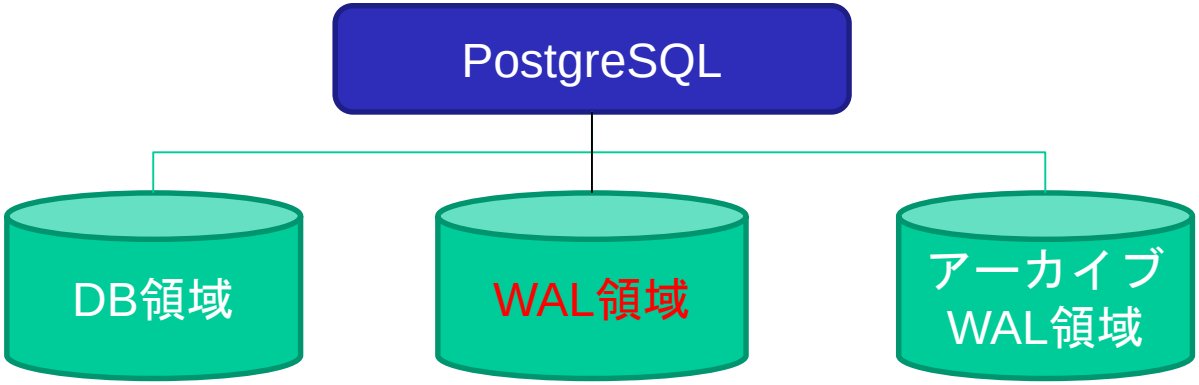
- データベースに作成するテーブル・インデックスの容量を見積もる(見積もり方法は後述)
  - システムに対してどのようにデータが追加/更新/削除されるのかを正しく把握して見積もることが重要



容量監視

WAL領域のディスク容量監視

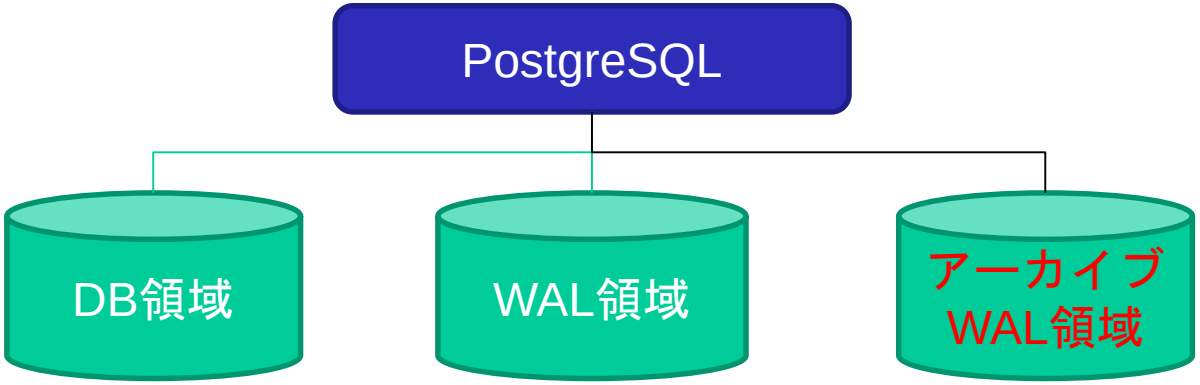
- トランザクションログ(WAL)の容量を見積もる
  - WALは循環して利用されるため最大容量を把握する
  - 1つのWALセグメントファイルのサイズは16MB
  - WAL領域のサイズはmax_wal_sizeパラメータで制御
    - 9.4以前はcheckpoint_segmentsパラメータで制御
    - →ざっくり「**16MB × (checkpoint_segments × 3 ＋ 1)**」程度必要



容量監視

OSS-DB

- アーカイブWAL領域のディスク容量監視
  - アーカイブされるWALの容量を見積もる
    - アーカイブWALはPITRによるリカバリ(後述)で利用する
    - バックアップ取得方針(どの程度の間隔でベースバックアップを取得するか、何世代のバックアップを保持するか等)を考慮し、容量を見積もる



容量監視

OSS-DB

ディスク容量の実測方法

- 以下のSQL関数/OSコマンドで確認する

分類	コマンド例	備考

データベース領域	pg_database_size('db');
	pg_relation_size('table');
	pg_toatl_relation_size('table');
		pg_relation_sizeにはインデックス名も指定できる
		pg_total_relation_sizeにはインデックスなどのサイズも含まれる

WAL領域 アーカイブWAL領域	dfコマンド/duコマンド	

容量監視

OSS-DB

- テーブル・インデックス容量見積り
- 見積もりに必要な情報
  - 各データ型とデータサイズ
    - テーブルに定義した列がどのような型なのか
    - インデックスを定義した列がどのような型なのか
    - それぞれの型がどの程度のデータサイズなのか
  - テーブル・インデックスファイルの使われ方
    - 各ファイルがどのようなレイアウトで構成されているかを理解する(後述)
  - データの増加/減少の傾向
    - システム毎の特徴を理解して見積もる

各データ型のデータサイズ

- 代表的なものは以下の通り

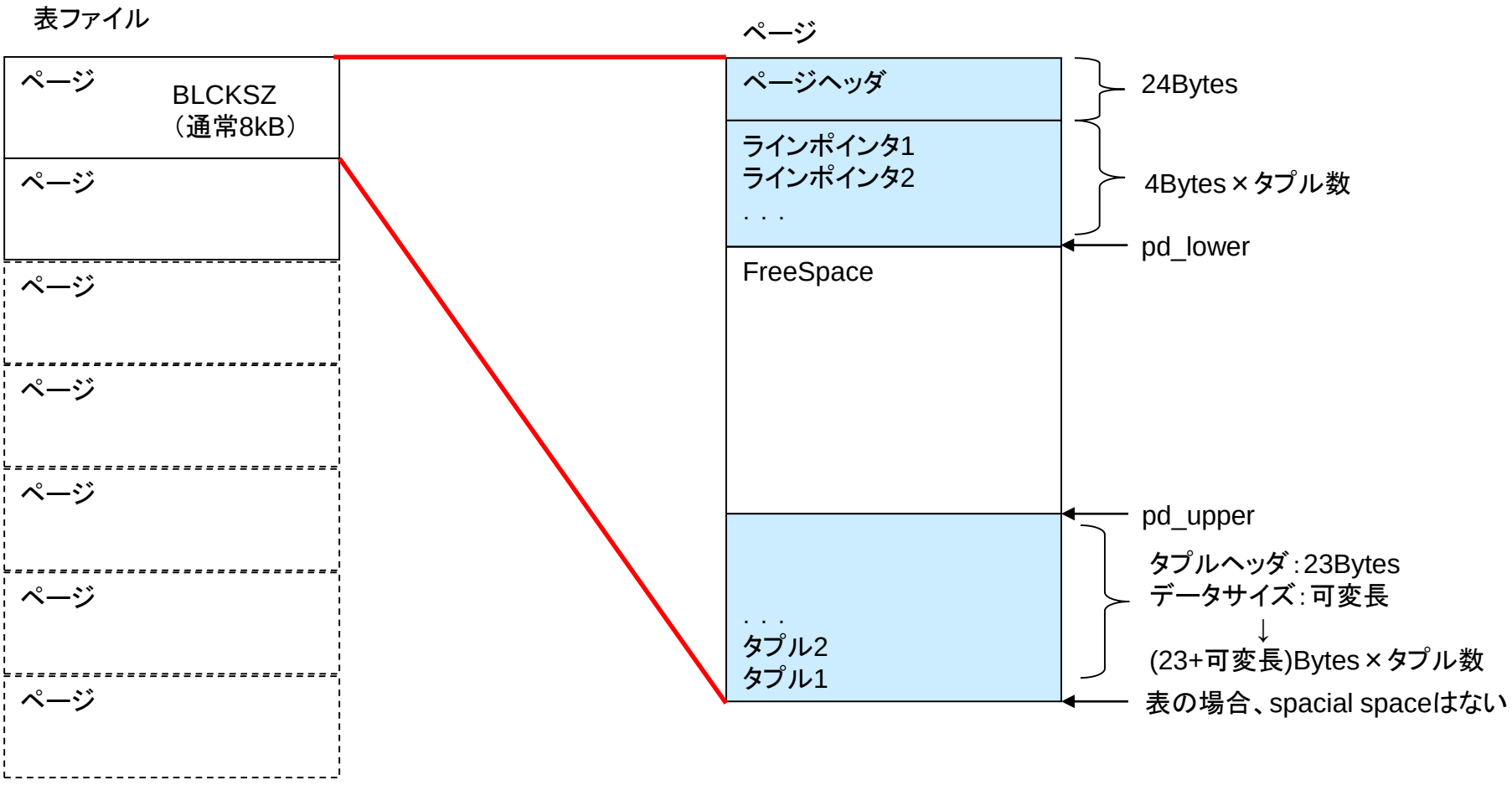
データ型	サイズ
smallint	2バイト
integer	4バイト
bigint	8バイト
real	4バイト
Timestamp	8バイト
date	4バイト
interval	16バイト

データ型	サイズ
varchar(n)	N<126の場合 1 + Nバイト N>=126の場合 4 + Nバイト
char(n)	
text	

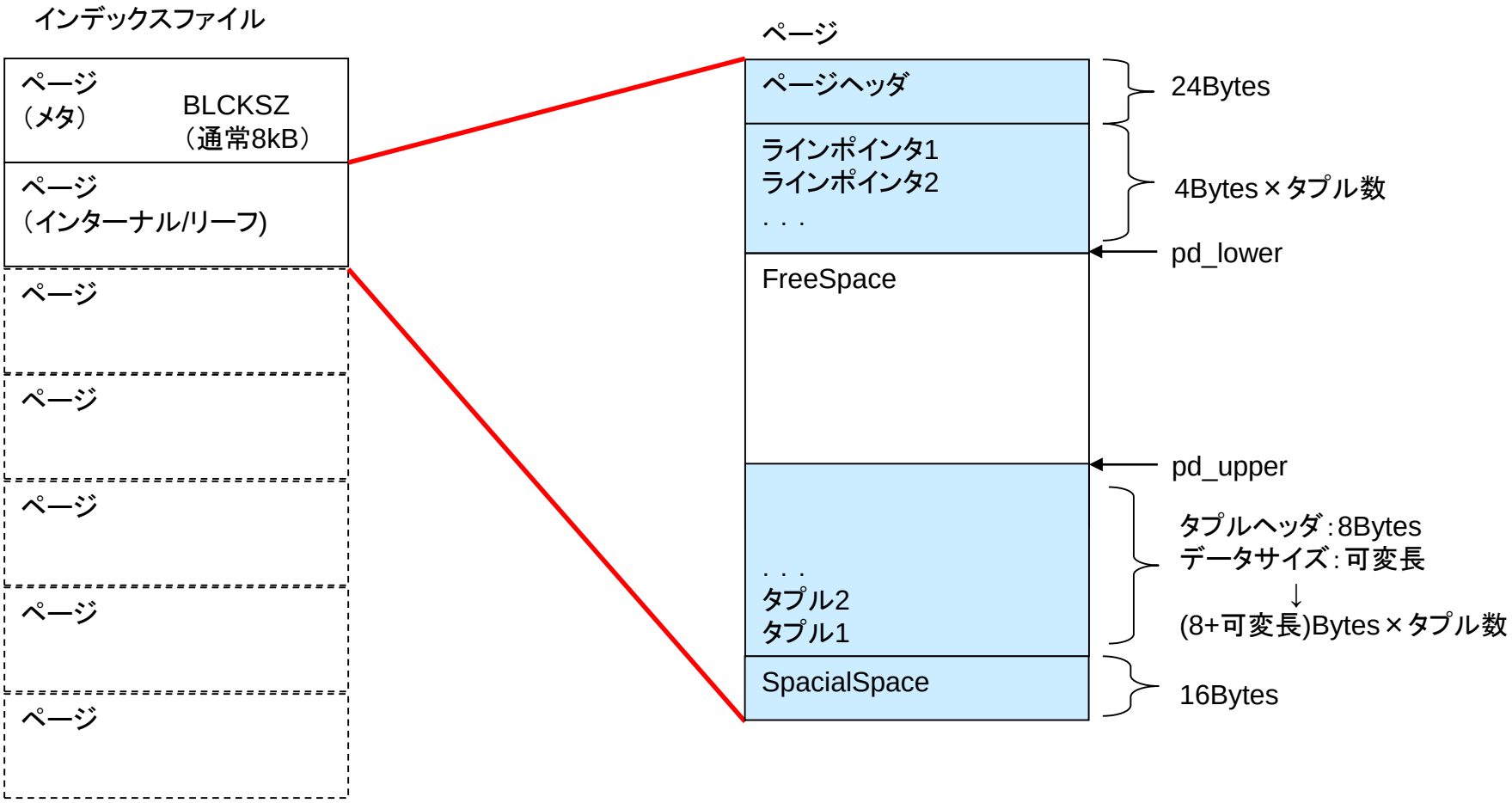
※Nは、文字数nの各エンコードでのバイト数

【参考】 PostgreSQL 9.0.4文書　第8章 データ型
http://www.postgresql.jp/document/9.0/html/datatype.html

■テーブルファイルのレイアウト



インデックスファイルのレイアウト



容量監視

OSS-DB

- テーブル・インデックス容量見積もり方
  - テーブルファイル
    - テーブルのスキーマ(DDL)から1行あたりのデータサイズを算出する
      - タプルヘッダ分のデータも考慮すること
    - 1ページ(8192Byte)あたりに何行分のデータが格納可能か算出する
      - FILLFACTORの分も考慮すること
    - 想定する行数を格納するためには、何ページが必要かを算出する

容量監視

OSS-DB

テーブル・インデックス容量見積もり方

- インデックスファイル
  - インデックス定義(DDL)から1エントリあたりのデータサイズを算出する
    - タプルヘッダ分のデータも考慮すること
  - 1リーフページ(8192Byte)に何エントリ分のデータが格納可能か算出する
    - FILLFACTORの分も考慮すること
  - 想定するエントリを格納するのに必要なリーフページ数を算出する
  - 全てのリーフページをカバーするために必要なルートページおよびインターナルページ数を算出する
    - ルートページ、インターナルページのFILLFACTORは70%固定

メンテナンス

OSS-DB

定常的なメンテナンス

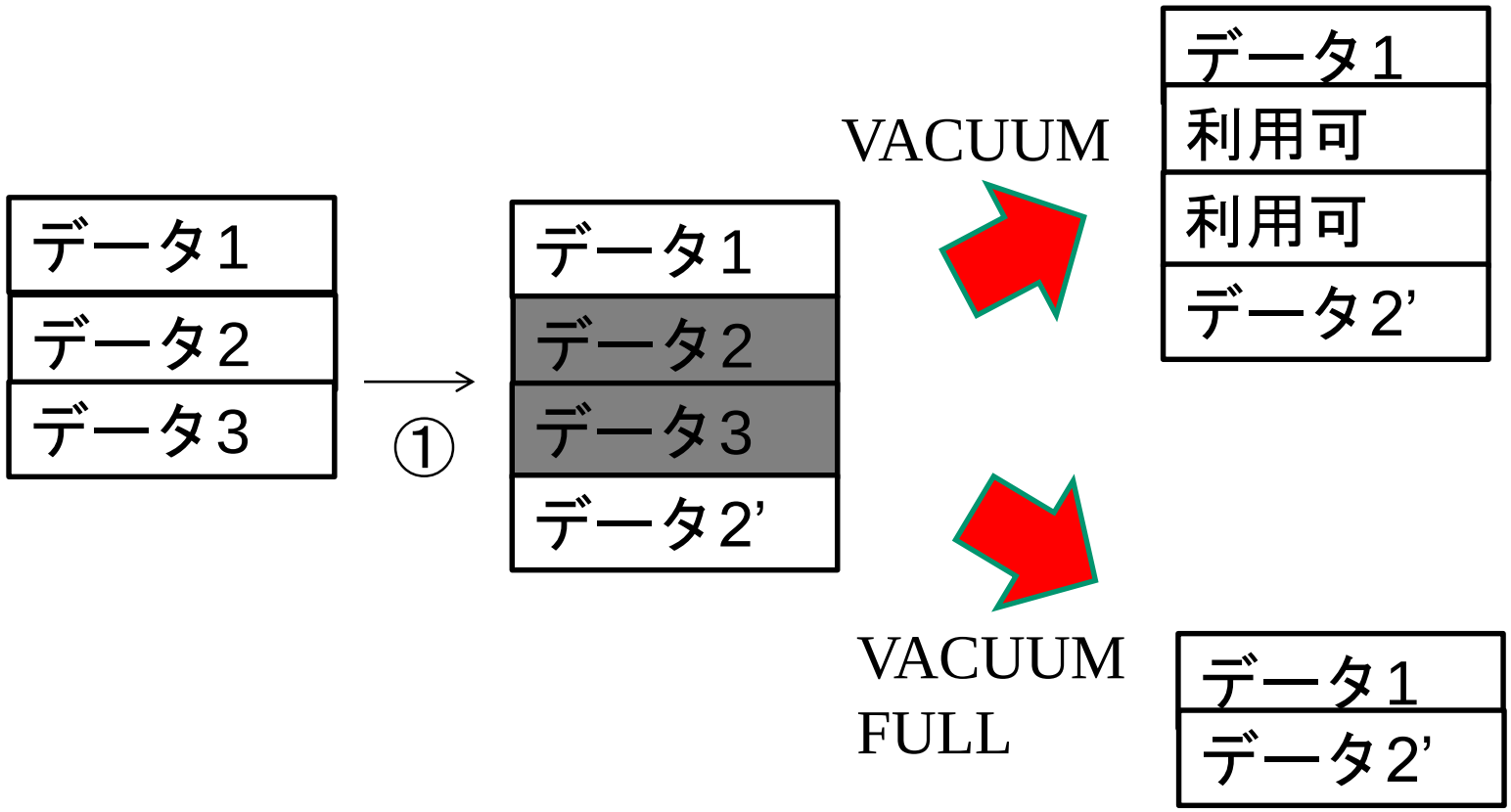
- PostgreSQLが安定して稼働するためには、定期的なメンテナンスが重要である
- ここでは、下記のメンテナンス作業について解説する
  - VACUUM/自動VACUUM
    不要領域の再利用とトランザクションID周回防止
  - ANALYZE
    統計情報の収集
  - REINDEX
    インデックスの再構築

メンテナンス

OSS-DB

- VACUUM
  - VACUUMの必要性
  - VACUUMとVACUUM FULLがあるが、実施する目的は異なる
  - VACUUMを実行する目的
    - 更新/削除された行の再利用
    - トランザクションID周回の回避
  - VACUUM FULLを実行する目的
    - 更新/削除された行を切り詰める

VACUUM/VACUUM FULLのイメージ



①データ2を更新、データ3を削除

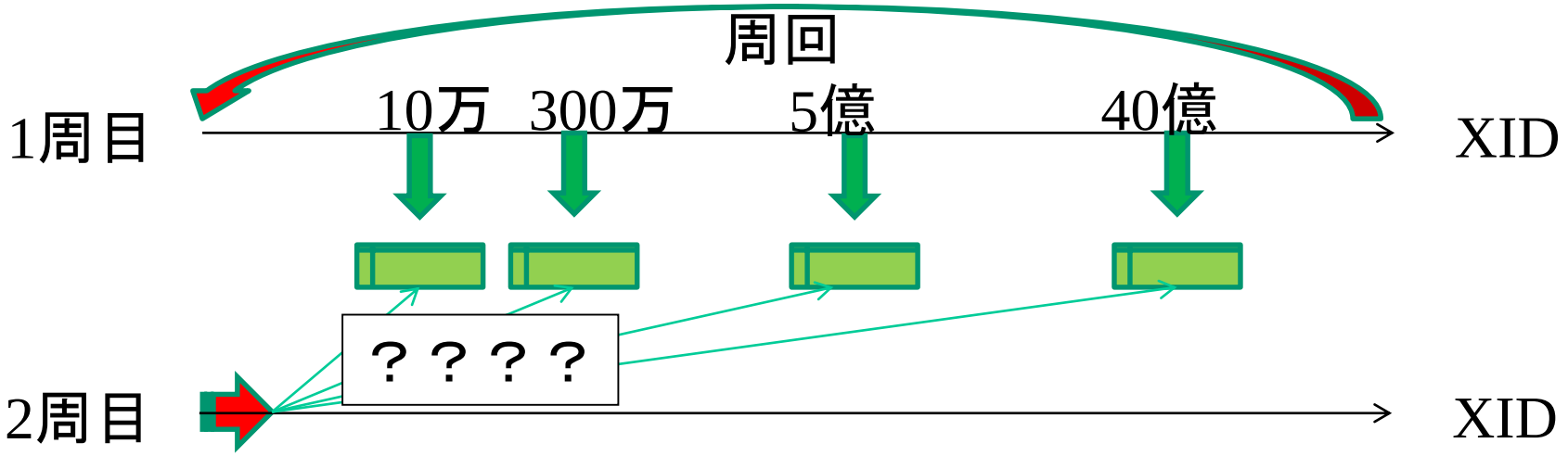
トランザクションID周回問題(1/3)

- 走行している更新トランザクションにはトランザクションID(XID)が割り振られる
- テーブルにもXIDが付与され、他トランザクションから可視とするか不可視とするかの判断がなされる(MVCC)
- XIDは20億の古いIDと20億の新しいIDの40億で管理されており、周回を繰り返す

【参考】 PostgreSQL 9.0.4文書　第23章 23.1.4. トランザクションIDの周回エラーの防止
http://www.postgresql.jp/document/9.0/html/routine-vacuuming.html#VACUUM-FOR-WRAPAROUND

トランザクションID周回問題(2/3)

- 周回を迎えたXIDを持つトランザクションからは全てのテーブルデータが未来のものとなり、不可視となる。これを「トランザクションID周回問題」と呼ぶ



トランザクションID周回問題(3/3)

- 定期的にVACUUM(20億トランザクションまでにすべてのDBのすべてのテーブルをVACUUM)することで、特殊なXID(XID=2)を設定し、すべてのトランザクションから可視とすることで問題を回避する
- ある時点でDB、テーブル上の最古のXIDから現在のXIDまでのトランザクション数を確認するには、以下のクエリを実行する
  - SELECT datname, age(datfrozenxid) FROM pg_database;
  - SELECT relname, age(relfrozenxid) FROM pg_class WHERE relkind = 'r';

メンテナンス

OSS-DB

遅延VACUUM

- VACUUM中のI/O操作と同時実行しているデータベース活動のI/O操作が競合するのを避ける目的で設定する
- VACUUM中のI/O操作をコストとしてカウンタに積算していき、閾値を超えた時点でVACUUM処理を中断する
- 中断後はカウンタをリセットし、継続する

【参考】 PostgreSQL 9.0.4文書　第18章 18.4.3. コストに基づくVACUUM
http://www.postgresql.jp/document/9.0/html/runtime-config-resource.html#RUNTIME-CONFIG-RESOURCE-VACUUM-COST

遅延VACUUM関連のパラメータ(1/2)

- vacuum_cost_delay
  - 閾値を超えた際にどの程度の期間、処理を中断するかを指定する(単位はミリ秒)
- vacuum_cost_page_hit
  - 共有バッファキャッシュから取得したページに対してVACUUM処理したときに積算するコスト(デフォルト1)
- vacuum_cost_page_miss
  - ディスクから読み込んだページに対してVACUUM処理したときに積算するコスト(デフォルト10)

メンテナンス

OSS-DB

遅延VACUUM関連のパラメータ(2/2)

- vacuum_cost_page_dirty
  - VACUUM処理済みのページを書き戻すときに積算するコスト(デフォルト20)
- vacuum_cost_limit
  - VACUUM処理を中断するかどうかを判断する積算コストの閾値(デフォルト200)

自動バキューム

- 自動バキュームの設定をすることでVACUUMとANALYZE(後述)の自動実行が可能となる
- 設定を有効にすると、自動バキュームランチャが常駐し、テーブルへの挿入/更新/削除回数に応じて自動バキュームワーカを起動する
- 挿入/更新/削除回数をカウントするためには、稼働統計情報収集機能を有効にする必要がある
  - track_countsをonに設定する

メンテナンス

OSS-DB

自動バキューム関連のパラメータ(1/2)

- autovacuum
  - 自動バキュームを行うか否かを設定(デフォルトon)
  - ただし、offを設定していても、トランザクションIDの周回が発生しそうになると強制的にVACUUMが動く
- log_autovacuum_min_duration
  - 指定した時間(ミリ秒)以上に処理に時間を要した場合に、ログを出力する(デフォルト-1：無効)
- autovacuum_max_workers
  - 同時に実行される自動バキュームワーカ数(デフォルト3)

メンテナンス

OSS-DB

自動バキューム関連のパラメータ(2/2)

- autovacuum_naptime
  - 自動バキュームランチャがバキュームの必要性を確認する期間(デフォルト1min)
- autovacuum_vacuum_threshold … (1)
- autovacuum_vacuum_scale_factor … (2)
  - 自動バキュームランチャがバキュームが必要か否かを判断する際の閾値計算で使用する
  - それぞれのデフォルト値は、50(行)、0.2(20%)
  - 閾値=(1) + (2) * pg_class.reltuples
    - UPDATE/DELETE回数が、この閾値を超えたらVACUUMを自動実行する

- VACUUM/自動バキュームの動作確認
  - pg_stat_user_tablesのlast_vacuum列、last_autovacuum列で手動VACUUM、自動VACUUMが起動した時刻を確認できる
  - 同じくpg_stat_user_tablesのn_dead_tup列で不要な行(dead-tuple)が除去されたことを確認できる

- pg_stat_user_tablesの例
  - 500行の不要行を自動バキュームで除去

```
# SELECT last_vacuum, last_autovacuum, n_dead_tup FROM pg_stat_user_tables ;
-[ RECORD 1 ]---+----
last_vacuum     |
last_autovacuum |
n_dead_tup      | 500


# SELECT last_vacuum, last_autovacuum, n_dead_tup FROM pg_stat_user_tables ;
-[ RECORD 1 ]---+------------------------------
last_vacuum     |
last_autovacuum | 2013-06-20 00:51:26.872624+09
n_dead_tup      | 0
```

メンテナンス

OSS-DB

ANALYZE

- ANALYZEの必要性
- PostgreSQLの問い合わせはプランナにより作成される実行計画に基づき実行される
- 実行計画は統計情報を元に作成されるため、データ分布の変更に応じて、統計情報を更新することが重要である
- 統計情報を更新するコマンドがANALYZEコマンドである

メンテナンス

REINDEX

- REINDEXの必要性
- **完全に空になった**インデックスページは再利用される
- システムのインデックスに対する挿入/更新/削除のパターンによっては、数個のエントリを残している可能性がある
- このような状況では、無駄なページが散在しており、性能への影響がでる
- 性能への影響が顕著である場合には、REINDEXによるインデックス再構築が有効となる

レプリケーションとホットスタンバイ

OSS-DB

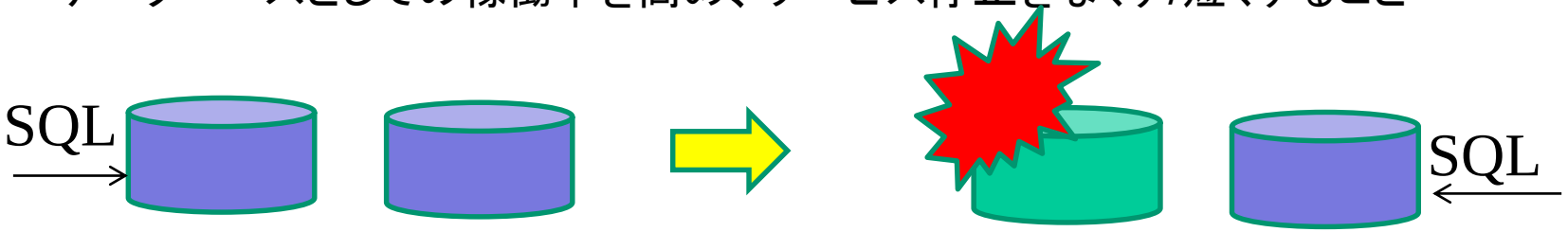
ホットスタンバイ運用

- PostgreSQLのホットスタンバイ運用について、以下の2つの機能を理解する
  - ストリーミング・レプリケーション
  - ホットスタンバイ
- 各機能で実現できること/できないこと、両機能を組み合わせることで実現できること/できないことを理解し、利用することが重要

レプリケーションとホットスタンバイ

レプリケーションの目的

- 高可用化
  - データベースとしての稼働率を高め、サービス停止をなくす/短くすること



- 負荷分散
  - 大量のSQLを複数のデータベースで処理し、1台あたりの負荷を軽減すること

レプリケーションとホットスタンバイ

- PostgreSQLで利用できるレプリケーション
  - PostgreSQL9.0で本体機能として「ストリーミング・レプリケーション」が実装された
  - ストリーミング・レプリケーションでは、シングルマスタ/マルチスレーブの冗長構成をとれる
  - マスタはWALをレコード単位でスレーブに送信。スレーブは受信したWALレコードをリカバリすることでレプリケーションを実現する

レプリケーションとホットスタンバイ

OSS-DB

ストリーミング・レプリケーションの設定

- マスタ側の設定
  - postgresql.conf
    - listen_addresses　　別サーバから接続可能にする
    - wal_level　　archive/hot_standbyにする
    - archive_mode　　onにする
    - archive_command　　適切にアーカイブする
    - max_wal_senders　　1以上に設定する
  - pg_hba.conf
    - 第2列目(接続データベース名)に、"replication"を設定
    - 例:スレーブサーバ(192.168.1.2)からpostgresユーザでレプリケーションする場合

```
host  replication  postgres  192.168.1.2/32  trust
```

レプリケーションとホットスタンバイ

OSS-DB

ストリーミング・レプリケーションの設定

- スレーブ側の設定
  - recovery.conf
    - standby_mode　　onにする
    - primary_conninfo　　マスタの情報を記述する
    - trigger_file　　停止のトリガとなるファイル名
    - restore_command　　適切にリストアする
  - primary_conninfoの設定例
    - マスタサーバが192.168.1.1の5432ポートで起動している場合の例

```
primary_conninfo='host=192.168.1.1 port=5432 user=postgres'
```

レプリケーションとホットスタンバイ

OSS-DB

- ストリーミング・レプリケーションの手順
  i. 「マスタ側の設定」を行う
  ii. マスタのベースバックアップを取得する
  iii. ベースバックアップをスレーブで展開する
  iv. 「スレーブ側の設定」を行う
  v. スレーブを起動する
- スレーブの起動確認
  - スレーブ側のログに以下のメッセージが出力されていることを確認する

LOG: streaming replication successfully connected to primary

レプリケーションとホットスタンバイ

OSS-DB

ストリーミング・レプリケーションの手順

- フェールオーバ
  - PostgreSQLには「自動フェールオーバ」の機能はない
  - PostgreSQLの「コマンド」でフェールオーバを実行するものはない
  - recovery.confのtrigger_fileパラメータで設定したトリガファイルを作成することで、フェールオーバ(スタンバイをマスタに昇格)を行うことができる
    - trigger_fileに ‘/tmp/trigger.file’ を指定した場合

```
$ touch /tmp/trigger.file
```

レプリケーションとホットスタンバイ

OSS-DB

ストリーミング・レプリケーションの監視

- 状態確認関数
  - pg_current_xlog_location()
    - (マスタの)WAL書き出し位置を返却
  - pg_last_xlog_receive_location()
    - (スタンバイの)WAL受信位置を返却
- 上記の位置を比較することで、レプリケーションの遅延具合を確認できる

レプリケーションとホットスタンバイ

OSS-DB

ストリーミング・レプリケーションの負荷分散

- PostgreSQL9.0から加わったホットスタンバイ機能により参照負荷分散が可能になった
- ホットスタンバイとはサーバがリカバリ中に、そのサーバに接続し問い合わせを実施できることを指す
- 実行可能な問い合わせは参照クエリおよび一部の制御クエリのみで、更新クエリの実行は不可

レプリケーションとホットスタンバイ

OSS-DB

- ホットスタンバイの設定
  - マスタ側の設定
    - postgresql.conf
      - wal_level　hot_standbyに設定
  - スレーブ側の設定
    - postgresql.conf
      - hot_standby　　onに設定

アクセス統計情報

OSS-DB

概要

- 統計情報コレクタによってデータベースの　　活動状況が収集される
  - ANALYZEによって収集される統計情報とは別物
- 標準統計情報ビューや統計情報アクセス関数にて収集された情報を参照可能
- デフォルトで有効設定
  - 自動バキュームを有効にするために統計情報が必要
- 公式ドキュメントではアクセス統計情報という　名称では説明されていない
  - 実行時統計情報およびpg_locksを指すと思われる

アクセス統計情報

OSS-DB

標準統計情報ビュー

- pg_stat_database
  - データベースあたり1行の形式でデータベース全体の情報を表示する
  - blks_hit列とblks_read列の値を用いてキャッシュヒット率を算出可能
    - blks_hit / (blks_hit + blks_read)
  - blks_hitにはPostgreSQLのバッファキャッシュに　おけるヒットのみが含まれ、OSのファイルシステムキャッシュは含まれない
  - 同一トランザクション内では同一結果を取得する
    - pg_stat_clear_snapshot関数で最新情報に更新可能


```
=# SELECT * FROM pg_stat_database WHERE datname = 'postgres';
-[ RECORD 1 ]--+------------------------------
datid          | 12896
datname        | postgres
numbackends    | 1
xact_commit    | 13596
xact_rollback  | 9
blks_read      | 2924
blks_hit       | 1710543
tup_returned   | 1006171
tup_fetched    | 646564
tup_inserted   | 106754
tup_updated    | 19700
tup_deleted    | 13
conflicts      | 0
temp_files     | 0
temp_bytes     | 0
deadlocks      | 0
blk_read_time  | 0
blk_write_time | 0
stats_reset    | 2013-11-22 11:57:30.824046+09
```

アクセス統計情報

OSS-DB

標準統計情報ビュー

- pg_stat_bgwriter
  - チェックポイントやバックグラウンドライタに関するデータベースクラスタ全体の統計情報を表示する
    - 共有バッファに関する情報も表示される
  - パラメータチューニングの指標値として利用可能
    - buffers_backendがbuffers_allocより大きい場合、shared_buffersの値が不足している可能性がある　等

列名	概要
buffers_checkpoint	チェックポイントにより書き出されたdirtyバッファ数
buffers_clean	バックグラウンドライタにより書き出されたdirtyバッファ数
buffers_backend	新しいバッファ割り当てを行う必要があったためにバックエンドプロセスにより書き出されたdirtyバッファ数

```
=# SELECT * FROM pg_stat_bgwriter ;

-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 32
checkpoints_req       | 0
checkpoint_write_time | 321157
checkpoint_sync_time  | 807
buffers_checkpoint    | 4005
buffers_clean         | 0
maxwritten_clean      | 0
buffers_backend       | 3515
buffers_backend_fsync | 0
buffers_alloc         | 2988
stats_reset           | 2013-11-22 11:57:26.409143+09
```

アクセス統計情報

OSS-DB

標準統計情報ビュー

- pg_stat_all_tables(1/2)
  - テーブルあたり1行の形式で、テーブルへのアクセス　統計情報を表示する
  - テーブルスキャン1回分の読み取り行数を確認可能
    - seq_tup_read / seq_scan
    - テーブルスキャン1回分の読み取り行数が予想より大きい場合、インデックスが想定通りに利用されていない可能性がある
  - バキュームの対象量を確認可能
    - n_dead_tup列でバキュームの対象となる行数を確認可能
    - pg_relation_size関数等と併せて利用することで大凡の　バキュームの対象量を把握できる


```
=# SELECT * FROM pg_stat_all_tables WHERE relname = 'pgbench_history';
-[ RECORD 1 ]-----+------------------------------
relid             | 16406
schemaname        | public
relname           | pgbench_history
seq_scan          | 0
seq_tup_read      | 0
idx_scan          |
idx_tup_fetch     |
n_tup_ins         | 6540
n_tup_upd         | 0
n_tup_del         | 0
n_tup_hot_upd     | 0
n_live_tup        | 1000
n_dead_tup        | 0
last_vacuum       | 2013-11-27 17:13:05.288006+09
last_autovacuum   |
last_analyze      | 2013-11-27 17:13:05.288405+09
last_autoanalyze  | 2013-11-27 17:22:25.897194+09
vacuum_count      | 1
autovacuum_count  | 0
analyze_count     | 1
autoanalyze_count | 5
```

アクセス統計情報

OSS-DB

- 標準統計情報ビュー
  - pg_stat_all_tables(2/2)
    - HOT更新の比率を確認可能
      - n_tup_hot_upd / n_tup_upd
      - HOT更新の比率が予想より小さい場合、不要なインデックスの存在やロングトランザクションの影響を調査する必要がある
    - pg_*_sys_*やpg_*_user_*ビューは、選択対象以外はpg_*_all_*ビューと内容は同じ


```
=# SELECT definition FROM pg_views WHERE viewname = 'pg_stat_sys_tables';

-[ RECORD 1 ]----------------------------------------------------------
definition |  SELECT pg_stat_all_tables.relid,
           |     pg_stat_all_tables.schemaname,
           |     pg_stat_all_tables.relname,
           |     pg_stat_all_tables.seq_scan,
           |     pg_stat_all_tables.seq_tup_read,

                          ～省略～

           |     pg_stat_all_tables.last_autoanalyze,
           |     pg_stat_all_tables.vacuum_count,
           |     pg_stat_all_tables.autovacuum_count,
           |     pg_stat_all_tables.analyze_count,
           |     pg_stat_all_tables.autoanalyze_count
           |    FROM pg_stat_all_tables
           |   WHERE ((pg_stat_all_tables.schemaname = ANY
           |  (ARRAY['pg_catalog'::name, 'information_schema'::name])) OR
           |  (pg_stat_all_tables.schemaname ~ '^pg_toast'::text));
```

アクセス統計情報

OSS-DB

- 標準統計情報ビュー
  - pg_statio_all_tables
    - テーブルあたり1行の形式で、ブロック単位のI/Oに 関する統計情報を表示する
    - *_blks_readが*_blks_hitより大幅に低い場合は 共有バッファが有効動作していると判断できる
      - 但し、*_blks_readはOSキャッシュから読み取られた場合も カウントアップされるため、ディスクアクセス回数をそのまま表示しているわけではない


```
=# SELECT * FROM pg_statio_all_tables WHERE relname = 'pgbench_accounts';

-[ RECORD 1 ]---+-----------------
relid           | 16412
schemaname      | public
relname         | pgbench_accounts
heap_blks_read  | 1672
heap_blks_hit   | 34694
idx_blks_read   | 276
idx_blks_hit    | 35802
toast_blks_read |
toast_blks_hit  |
tidx_blks_read  |
tidx_blks_hit   |
```

アクセス統計情報

OSS-DB

標準統計情報ビュー

- pg_stat_all_indexes
  - インデックス毎のアクセスに関する統計情報を表示
  - 使用されていないインデックスの特定が可能
- pg_statio_all_indexes
  - インデックス毎のI/Oに関する統計情報を表示

列名	概要
pg_stat_all_indexes列	
idx_scan	インデックススキャンの実行回数
idx_tup_read	インデックススキャンで返されたノード数
idx_tup_fetch	インデックススキャンで取り出されたレコード数
pg_statio_all_indexes列	
idx_blks_read	共有バッファ以外からブロックを読み込んだ回数
idx_blks_hit	共有バッファからブロックを読み込んだ回数


```
=# SELECT * FROM pg_stat_all_indexes
-# NATURAL JOIN pg_statio_all_indexes
-# WHERE indexrelname = 'pgbench_accounts_pkey';

-[ RECORD 1 ]-+----------------------
relid         | 16412
indexrelid    | 16423
schemaname    | public
relname       | pgbench_accounts
indexrelname  | pgbench_accounts_pkey
idx_scan      | 13080
idx_tup_read  | 14863
idx_tup_fetch | 13080
idx_blks_read | 276
idx_blks_hit  | 35802
```

アクセス統計情報

OSS-DB

標準統計情報ビュー

- pg_stat_activity
  - バックエンドプロセス単位にプロセス情報を表示する
  - track_activitiesパラメータが有効の場合、実行中のSQLの内容を表示可能
  - 発行されてから長時間経過しているSQLや、ロック待ち状態となっているSQLを調査可能
  - procpid列からプロセスIDを特定し、クエリを取り消す場合にはpg_cancel_backend(pid)関数、接続を強制切断する場合にはpg_terminate_backend(pid)関数を実行


```
=# SELECT * FROM pg_stat_activity ;

-[ RECORD 1 ]----+---------------------------------
datid            | 12896
datname          | postgres
pid              | 1774
usesysid         | 10
username         | postgres
application_name | psql
client_addr      |
client_hostname  |
client_port      | -1
backend_start    | 2013-12-04 09:57:00.780122+09
xact_start       | 2013-12-04 10:46:58.007787+09
query_start      | 2013-12-04 10:46:58.007787+09
state_change     | 2013-12-04 10:46:58.007795+09
waiting          | f
state            | active
query            | SELECT * FROM pg_stat_activity ;
```

実行計画

OSS-DB

概要

- 与えられたSQL文に対し、プランナが統計情報を参照して作成する
- SQL文が参照するテーブルをスキャンする方法やテーブルを結合するアルゴリズム等を示す
- SQL文の前にEXPLAINを付与して実行する
  - EXPLAINは実行計画を表示するのみで実際にSQLは実行されないがANALYZEオプションを付与すると実際にSQLが実行され、実行結果に基づく情報も併せて表示される

```
=>    EXPLAIN SELECT 1;
                              QUERY PLAN
       ----------------------------------------
        Result  (cost=0.00..0.01 rows=1 width=0)
```

EXPLAINおよびEXPLAIN ANALYZEの実行結果

```
=>    EXPLAIN SELECT 1;
                            QUERY PLAN
-----------------------------------------------------
 Result  (cost=0.00..0.01 rows=1 width=0)
```

プランタイプ

初期コスト

総コスト

行数

行の平均サイズ

```
=>    EXPLAIN ANALYZE SELECT 1;
                                QUERY PLAN
-----------------------------------------------------------------------
 Result  (cost=0.00..0.01 rows=1 width=0)
(actual time=0.001..0.001 rows=1 loops=1)
 Total runtime: 0.011 ms
```

実行時間

ループ回数

合計実行時間

EXPLAINおよびEXPLAIN ANALYZEの実行結果

```
=>    EXPLAIN ANALYZE SELECT 1 UNION SELECT 2;
                                  QUERY PLAN
      ------------------------------------------------------------------------
       Unique  (cost=0.05..0.06 rows=2 width=0)
               (actual time=0.004..0.005 rows=2 loops=1)
         ->  Sort  (cost=0.05..0.06 rows=2 width=0)
                   (actual time=0.004..0.004 rows=2 loops=1)
               Sort Key: (1)
               Sort Method:  quicksort  Memory: 25kB
               ->  Append  (cost=0.00..0.04 rows=2 width=0)
                           (actual time=0.001..0.002 rows=2 loops=1)
                     ->  Result  (cost=0.00..0.01 rows=1 width=0)
                                 (actual time=0.001..0.001 rows=1 loops=1)
                     ->  Result  (cost=0.00..0.01 rows=1 width=0)
                                 (actual time=0.000..0.001 rows=1 loops=1)
       Total runtime: 0.015 ms
```

実行計画のチューニング

OSS-DB

概要

- INDEXチューニング
- ANALYZE
- work_mem
- effective_cache_size
- プランナメソッド設定の変更
- プランナコスト定数の変更
- テーブル結合最適化処理の実行制御

実行計画のチューニング

OSS-DB

INDEXチューニング

- インデックスを追加して検索性能を向上させる
- SELECT句で取得する列がすべてインデックスに含まれる場合はIndexOnlyScan(テーブルにアクセスしない！)が使われる
- WHERE句で使われる列以外でもインデックスは利用されることに注意が必要
  - 結合条件として使われている場合
  - ソート条件として使われている場合
- 想定通りにインデックスが利用されているかをEXPLAINで確認する
  - 関数インデックスや複数列インデックスの追加を検討

実行計画のチューニング

OSS-DB

ANALYZE

- EXPLAIN ANALYZEを実行し、推定行と実際の行が乖離していないか確認する
  - 通常運用時は自動バキュームによりANALYZEも自動的に　実行されるため手動でのANALYZEは基本的に不要
  - 一時テーブルに対しては自動バキュームは実行されないため、手動でのANALYZE実行が必要

実行計画のチューニング

OSS-DB

- work_mem
  - ソートやハッシュ表作成時等に利用できるメモリ量を指定する
    - プランナは実行計画作成時にwork_memの値を考慮するため、値が小さすぎると効率的なプランタイプが選択されなくなる
  - shared_buffersとは別個に確保される
    - 単一のSQL中にwork_memを必要とする処理が複数存在する場合、その処理毎にwork_memが確保される
    - よって、大きすぎる値を設定するのは危険
    - log_temp_filesパラメータで値過不足の調査が可能

実行計画のチューニング

OSS-DB

- effective_cache_size
  - カーネルのバッファキャッシュを含めた、利用可能なディスクキャッシュの推定値を設定する
  - 増加させるとインデックススキャンが、減少させるとシーケンシャルスキャンが選択されやすくなる
  - プランナの実行計画の作成時のみに参照される値であって、実際に値が確保されるわけではない

- プランナコスト定数の変更
  - オプティマイザが生成する実行計画を制御する
    - 最適と思われる実行計画が生成されない場合に、オプティマイザに異なる計画を生成させる為の手段の一つ
  - ストレージ性能やメモリ容量等、ハードウェアの特性を考慮してパラメータを調整する必要がある
    - データベースの大半がディスクキャッシュに乗る場合では、seq_page_costとrandom_page_costを減らす
  - インデックススキャンを選択させやすくするには
    - random_page_costを減らす
    - effective_cache_sizeを増やす
      - shared_buffersの2倍程度が目安

実行計画のチューニング

OSS-DB

プランナメソッド設定の変更

- オプティマイザが生成する実行計画を制御する
  - 指定した計画タイプを無効化し実行計画を変更させる
- パラメータは全てboolean型で、デフォルトはon
- offにしても完全に使われなくなるわけではない
  - enable_seqscanをoffにしてもシーケンシャルスキャンが全く行われなくなるわけではない
- postgresql.confで設定してしまうと全てのSQLに影響が出てしまう為、SET文を用いてセッションもしくはトランザクション毎に設定をするべき

実行計画のチューニング

OSS-DB

テーブル結合最適化処理の実行制御

- 結合条件に3つ以上のテーブルが含まれる場合、どのような順番で各テーブルを結合すれば最も 効率的か判断するための処理が行われるが、テーブル数が増加すると処理時間が長大化する
- 下記パラメータに小さな値を設定することで、 最適化処理時間を短縮化できるが、実行計画の 精度が低下する可能性が増加する
  - from_collapse_limit
  - join_collapse_limit

ご清聴ありがとうございました。

■お問い合わせ■

NTTテクノクロス株式会社　ソフト道場

https://www.ntt-tx.co.jp/contact/soft_dojyo.html