



PostgreSQL 安定運用のレシピ

マニュアルには書かれていないPostgreSQLの真実

篠田典良 / 日本ヒューレット・パッカード株式会社 / 2014年12月5日

PostgreSQL Conference 2014 Hands-On 2

自己紹介

篠田典良(しのだのりよし)

- 所属

- 日本ヒューレット・パッカード株式会社 テクノロジーコンサルティング事業統括

- 現在の業務

- PostgreSQLをはじめ Oracle Database, Microsoft SQL Server, Vertica, Sybase ASE 等 RDBMS 全般に関するシステムの設計、チューニング、コンサルティング
- オープンソース製品に関する調査、検証
- Oracle Database 関連書籍の執筆
- 弊社講習「Oracle Database エンジニアのための PostgreSQL 入門」講師

- 関連する URL

- HP Open Services

- PostgreSQL Internals

- PostgreSQL 9.4 新機能検証報告レポート

<http://h50146.www5.hp.com/services/ci/opensource/>

- Oracle ACE ってどんな人？

<http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-jp.html>



Agenda

PostgreSQL 安定運用のレシピ

1. アーキテクチャと OS 設定
2. 安定稼働のために必要な設定
3. 障害の検知
4. 障害発生時の動作と対処

スライド内で使用される PostgreSQL および OS のバージョンは以下の通り

- PostgreSQL 9.4 RC1 (標準オプションでビルド)
- Red Hat Enterprise Linux 6 Update 5 (x86-64)

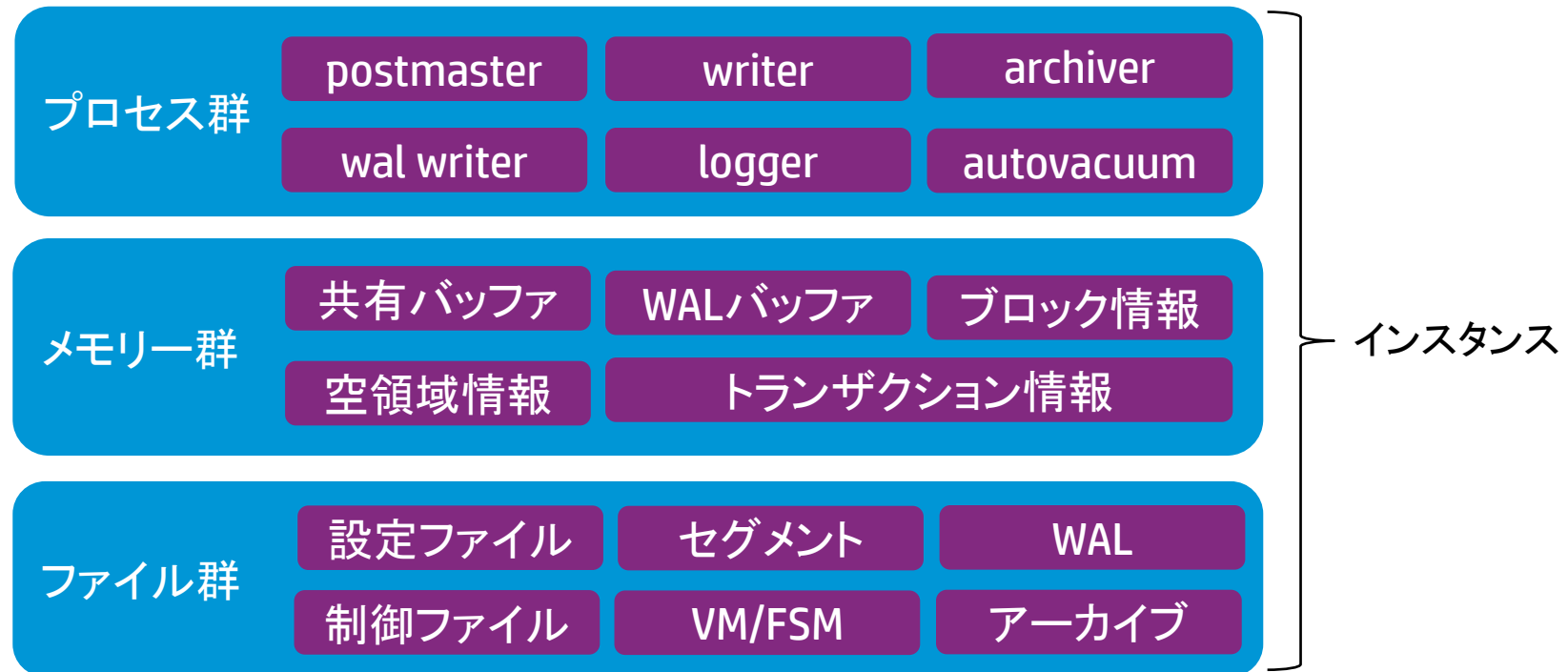
1. アーキテクチャとOS設定



1.1 アーキテクチャ全体

インスタンス

ユーザーに対してデータベースを利用可能な状態にした状態を示す



1.2 プロセス・アーキテクチャ

postmasterを親プロセスとする複数のプロセスから構成

主なプロセス		起動数	用途	備考
postgres		1	親プロセス、接続待ち	postmasterとも呼ばれる
	wal writer	0～1	一部のWAL書き込み	
	writer	1	セグメント書き込み	テーブルやインデックス
	logger	0～1	ログファイル書き込み	
	checkpointer	1	チェックポイント実行	
	autovacuum launcher	0～1	VACUUM 処理開始	
	autovacuum worker	0～複数	VACUUM 処理実行	
	postgres	接続数	クライアントSQL実行	
	wal sender	接続数	スレーブからの接続	レプリケーション・マスター
	wal receiver	0～1	マスターへの接続	レプリケーション・スレーブ
	archiver	0～1	アーカイブログの出力	
	stats collector	1	統計情報の更新	
	bgworker	複数	カスタム・ワーカー	

1.2 プロセス・アーキテクチャ

接続数とプロセス数

クライアントの接続数分のプロセスが起動される

- 起動数が変動するプロセス

プロセス	起動のタイミング	最大数を決めるパラメータ
postgres	クライアントの接続	max_connections
wal sender	レプリケーション・スレーブの接続 pg_basebackup コマンド実行時	max_wal_senders
autovacuum worker	自動 VACUUM 実行時	autovacuum_max_workers
bgworker	設定に依存	max_worker_processes

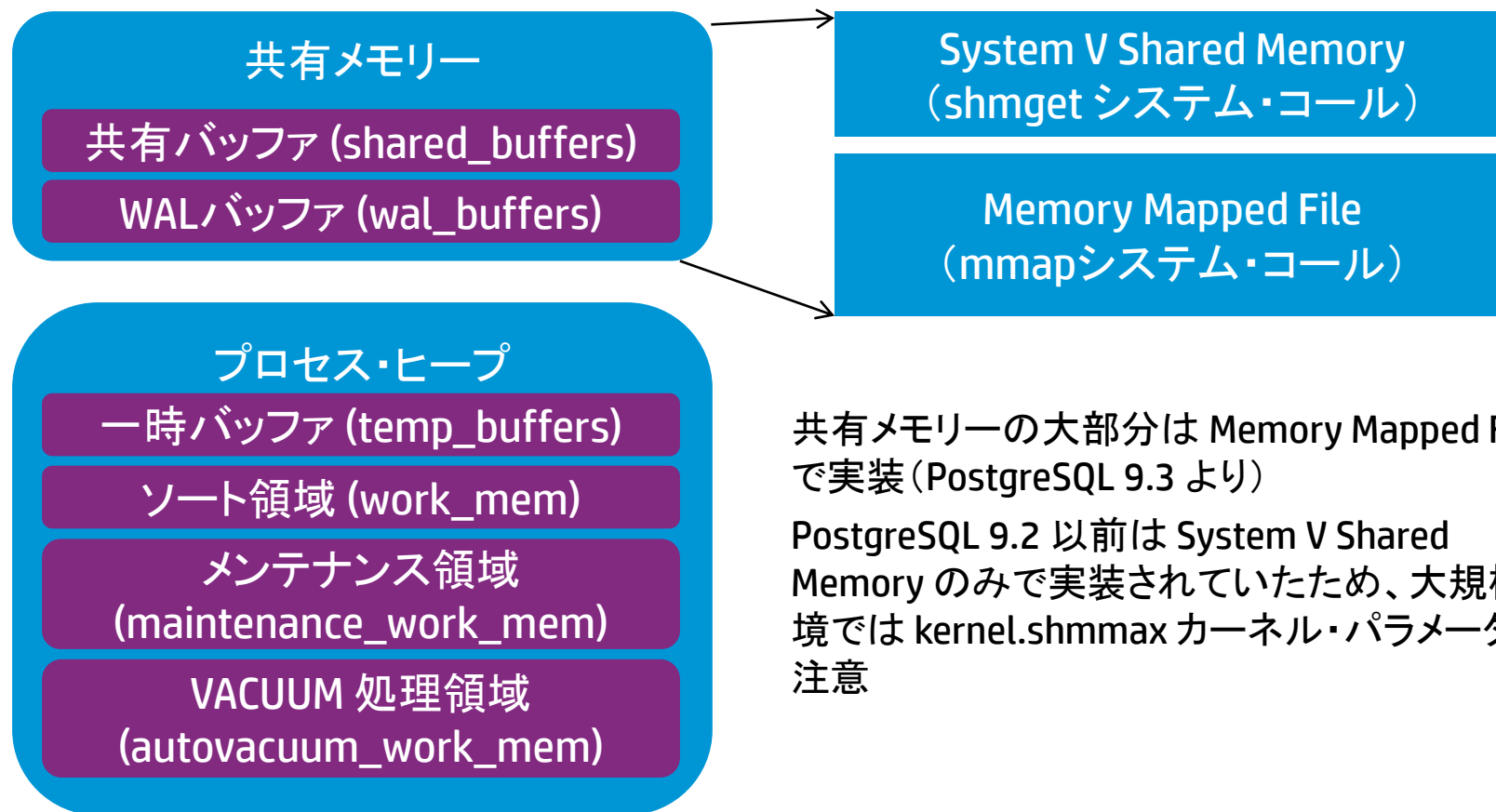
- ユーザーのプロセス制限を考慮(ファイル /etc/security/limits.conf)

postgres	soft	nproc	1024
postgres	hard	nproc	1024

1.3 メモリー・アーキテクチャ

共有メモリーとヒープ

共有メモリーの物理構造とプロセス・ヒープ



共有メモリーの大部分は Memory Mapped File で実装 (PostgreSQL 9.3 より)

PostgreSQL 9.2 以前は System V Shared Memory のみで実装されていたため、大規模環境では kernel.shmmax カーネル・パラメータに注意

1.3 メモリー・アーキテクチャ

Huge Pages と Semaphore

大規模メモリー環境では Huge Pages が有効(9.4～)

- パラメータ huge_pages を変更する
 - デフォルト値 (try) は Huge Pages 機能が使えれば使う
 - 使用を強制する場合には on を指定
- Linux カーネル・パラメータ vm.nr_hugepages で使用するページ数(2 MB単位)を指定
 - Huge Pages を使用できない環境で パラメータ huge_pages を on にするとインスタンス起動エラー
 - vm.nr_hugepages で指定された値が小さすぎる場合もインスタンス起動エラー
 - mmap システムコールに MAP_HUGETLB マクロが指定される
- プロセス間の同期にはセマフォを使用する
 - セマフォ集合はインスタンス起動時に作成
 - インスタンス起動時に必要なセマフォ集合数が確保できない場合は起動不可
 - 作成されるセマフォ数は下記計算式で決定

- 最大バックエンド数 = $\text{max_connections} + \text{autovacuum_max_workers} + \text{max_worker_processes} + 1$
- 最大セマフォ数 = 最大バックエンド数 + 4
- セマフォ集合数 = $(\text{最大セマフォ数} + 15) / 16$

1.3 メモリー・アーキテクチャ

共有バッファと WAL バッファ

- 共有バッファは読み込んだデータのキャッシュとして使用される領域
 - パラメータ `shared_buffers` で指定
 - デフォルト値は 128 MB であり、小さすぎる(旧バージョンのデフォルト値は更に小さい)
 - 一般的にはチューニングにより物理メモリーの 20～40% を指定
- WAL バッファはトランザクション情報を一時的に保存する領域
 - パラメータ `wal_buffers` で変更
 - 大きくても数 MB～ 32 MB程度
 - デフォルトは -1 で、自動設定
 - 自動設定時はパラメータ `shared_buffers / 32` が指定
 - 自動設定時の下限は 64 KB、上限は 16 MB

1.3 メモリー・アーキテクチャ

リング・バッファ

大規模 I/O 発生時に既存のキャッシュを保護する機能

- 大量データの読み込みを行う場合、既存のキャッシュを保護するためにリング・バッファ使用
 - テーブルのサイズが共有バッファの 25 % を超える場合に使用
- リング・バッファのサイズは固定されているため変更できない

処理	サイズ	主な操作
一括読み込み	256 KB	Seq Scan (シーケンシャル・スキャン) CREATE MATERIALIZED VIEW 文 COPY TO 文
一括書き込み	16 MB	COPY FROM 文 CREATE TABLE AS SELECT 文
VACUUM	256 KB	VACUUM 文

- 上記表のサイズが、shared_buffers の 1/8 を超える場合は shared_buffers の 1/8 が使用される

1.3 メモリー・アーキテクチャ

ヒープ・メモリー

プロセス毎に確保される仮想メモリー

- 稼働されるプロセス数と物理メモリー容量を考慮して設定
デフォルト値はバージョンによって異なる
- 以下のパラメータを指定可能、最適値はチューニングによって決定

パラメータ	説明	デフォルト
temp_buffers	一時テーブル用バッファ	8 MB
work_mem	ソート処理、ハッシュ処理用バッファ	4 MB
maintenance_work_mem	VACUUM, インデックス作成等の処理用	64 MB
autovacuum_work_mem	自動VACUUM 専用(9.4～)	-1

- autovacuum_work_mem のデフォルト値(-1) は maintenance_work_mem を使う設定

1.4 ストレージ・アーキテクチャ

データベース・クラスタ

永続データの格納領域

- ファイルシステムのディレクトリを指定する (XFSまたは ext4を推奨)
 - ext4 の場合、マウント・オプション noatime, nodiratime を推奨
 - XFS の場合、マウント・オプション nobarrier, noatime, noexec, nodiratime を推奨
- インスタンスと1対1対応
- initdb コマンドで空きディレクトリを指定して作成
- 設定ファイル、データファイル等ほとんどの永続データを保存
- 通常データベース・クラスタ外に保存されるデータ
 - アーカイブログ・ファイル (パラメータ archive_command)
 - 標準外表領域 (CREATE TABLESPACE 文により作成)
 - プロセスIDファイル (パラメータ external_pid_file)

1.4 ストレージ・アーキテクチャ

データベース・クラスタの内部

ディレクトリ構造とファイル群

データベース・クラスタ(環境変数 PGDATA)

システム・カタログ(global)

pg_control

テーブル

インデックス

設定ファイル

トランザクション・ログ(pg_xlog)

WALファイル

デフォルト・テーブル空間(base)

データベースOID

テーブル

インデックス

ログ(pg_log)

ログ・ファイル

アーカイブログ

外部プロセスID

標準外テーブル空間

1.4 ストレージ・アーキテクチャ

データベース・クラスタの内部

主なディレクトリ

主なディレクトリ	用途	備考
base	標準のテーブル空間 (pg_default)	テーブル、インデックスを保存
global	システム・カタログ	
pg_clog	トランザクション情報	
pg_log	ログ・ファイル	
pg_replslot	レプリケーション・スロット	9.4 ~
pg_stat, pg_stat_tmp	稼働統計情報	
pg_tblspc	外部表領域用のシンボリック・リンク	
pg_xlog	トランザクション・ログ	

1.4 ストレージ・アーキテクチャ

データベース・クラスタの内部

主なファイル

主なファイル	用途	変更方法等
pg_hba.conf	ホストによる接続制限設定	エディタで編集
pg_ident.conf	OS ユーザーによる接続制限設定	エディタで編集
postgresql.auto.conf	パラメータ設定	ALTER SYSTEM 文で設定
postgresql.conf	パラメータ設定	エディタで編集
postmaster.opts	postmaster プロセス起動パラメータ	インスタンス起動時に再作成
postmaster.pid	postmaster プロセスの PID	インスタンス起動時に再作成
PG_VERSION	バージョン情報	クラスタ作成時に生成
global/pg_control	制御ファイル	チェックポイント発生時に更新

1.4 ストレージ・アーキテクチャ

パラメータ・ファイル

パラメータ設定が記述されたテキスト・ファイル

- ファイル名 postgresql.conf
- インスタンス起動時または pg_ctl reload コマンド実行時に参照
- 内部は「パラメータ名 = パラメータ値」の集合体
- # 以降はコメント
- 例

```
listen_addresses = '*'      # what IP address(es) to listen on;  
port = 5432                 # (change requires restart)  
max_connections = 100      # (change requires restart)
```

- 複数ファイルに分割する場合は include 句を使う
- 例

```
include_dir = 'conf.d'  
include_if_exists = 'exists.conf'  
include = 'special.conf'
```

1.4 ストレージ・アーキテクチャ

パラメータ・ファイル

「ALTER SYSTEM SET パラメータ名 = 値」文で設定されるパラメータ・ファイル(9.4～)

- ファイル名は postgresql.auto.conf
- 直接編集しない
- ALTER SYSTEM 文はファイルの編集のみ、動作中インスタンスのパラメータを変更するわけではない
- postgresql.conf と同じ名前のパラメータを設定した場合は postgresql.auto.conf 優先
- 例

```
postgres=# ALTER SYSTEM SET work_mem = '8MB';
ALTER SYSTEM
postgres=# \q
$ cat postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by ALTER SYSTEM command.
work_mem = '8MB'
```

1.4 ストレージ・アーキテクチャ

セグメント・ファイル

テーブルやインデックスを構成するファイル

- テーブルは複数のファイルから構成
- 8 KB のブロック(ページ)の集合として構成
- ファイル名はテーブル作成時は pg_class カタログの oid 列と同じだが、テーブルに対して TRUNCATE や VACUUM FULL を行うと変更される
- テーブル名からファイルを検索する例

```
postgres=> SELECT pg_relation_filepath('demo1');
```

```
pg_relation_filepath
```

```
-----
```

```
base/16385/16386
```

セグメント・ファイル

データベースOID (pg_database カタログ oid 列)

{データベース・クラスター}/base ディレクトリ

1.4 ストレージ・アーキテクチャ

セグメント・ファイル

テーブルを構成するその他のファイル群

属性	説明	備考
セグメントファイル	データ格納用 (pg_class カタログ relfilenode 列)	
セグメントファイル.{9}	データ格納用 (1 GB を超えると作成される) 拡張子 .1, .2, .3 が順次作成される	
セグメントファイル_vm	Visibility Map / ガベージ・ページを特定するファイル	後述
セグメントファイル_fsm	Free Space Map / ページの空き容量を管理するファイル	後述

- プロセスが同時にオープンするファイルの上限はパラメータ max_files_per_process で決定
- 同時に OS ユーザーのリソース上限を考慮 (ファイル /etc/security/limits.conf)

postgres	soft	nofile	1024
postgres	hard	nofile	1024

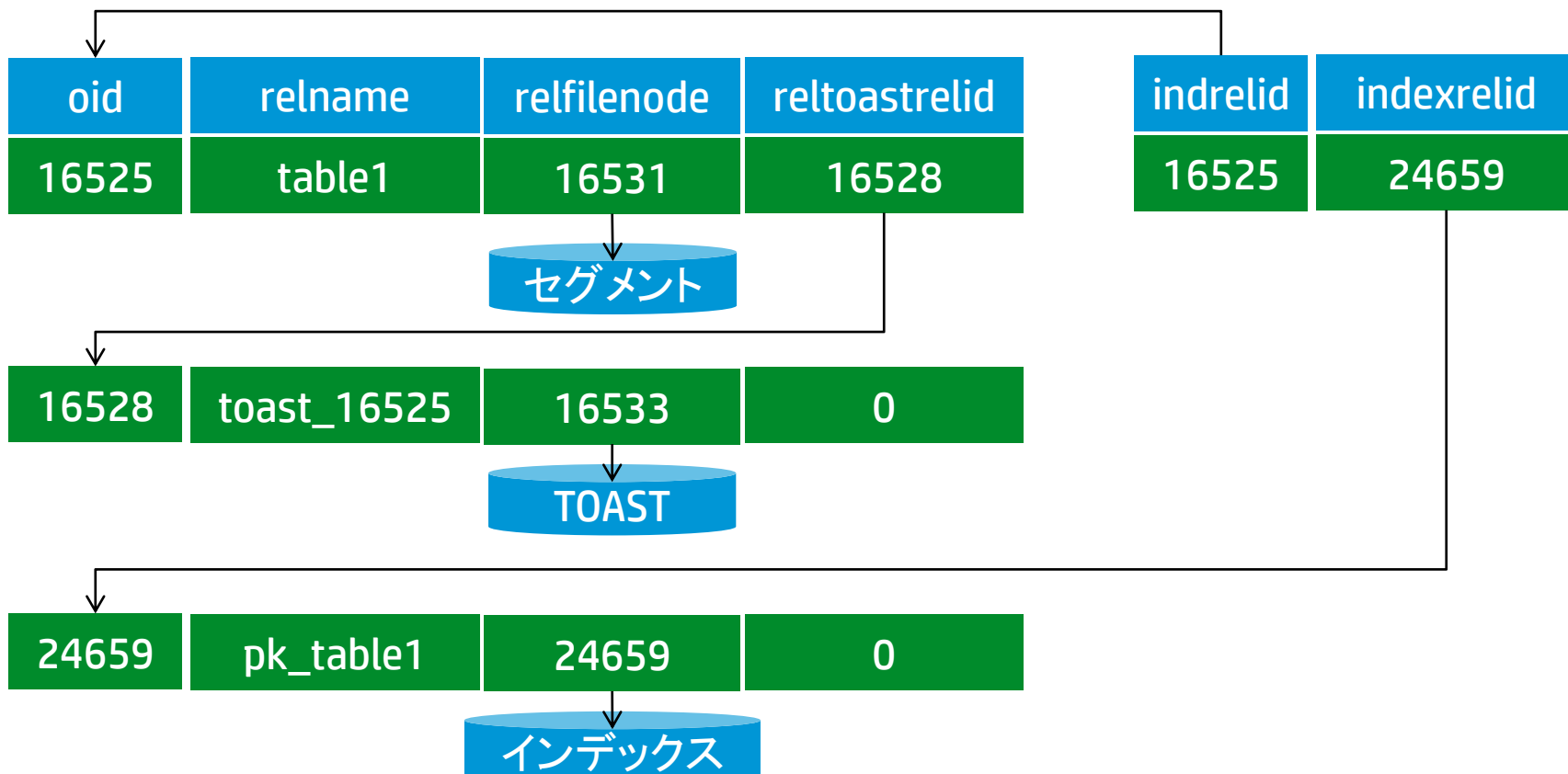
1.4 ストレージ・アーキテクチャ

TOAST ファイルとインデックス・ファイル

TOAST ファイルはページ・サイズ(8 KB)を超えるサイズのレコード用ファイル

pg_class カタログ

pg_index カタログ




1.4 ストレージ・アーキテクチャ

WAL ファイル

データベースに対する変更履歴を保存

- WAL ファイルは更新トランザクションの情報を記録する、固定サイズ(16 MB)のファイル
- インスタンスがクラッシュした場合のリカバリに使用
- トランザクションの確定と同時に更新(同期／非同期)
- pg_xlog ディレクトリに複数作成される

pg_xlog Directory



00000001000000030000004F
000000010000000300000050
000000010000000300000051



ファイル名を変更しながら循環使用
(checkpoint_segments / wal_keep_segments
で数を指定)

- パラメータは以下の通り

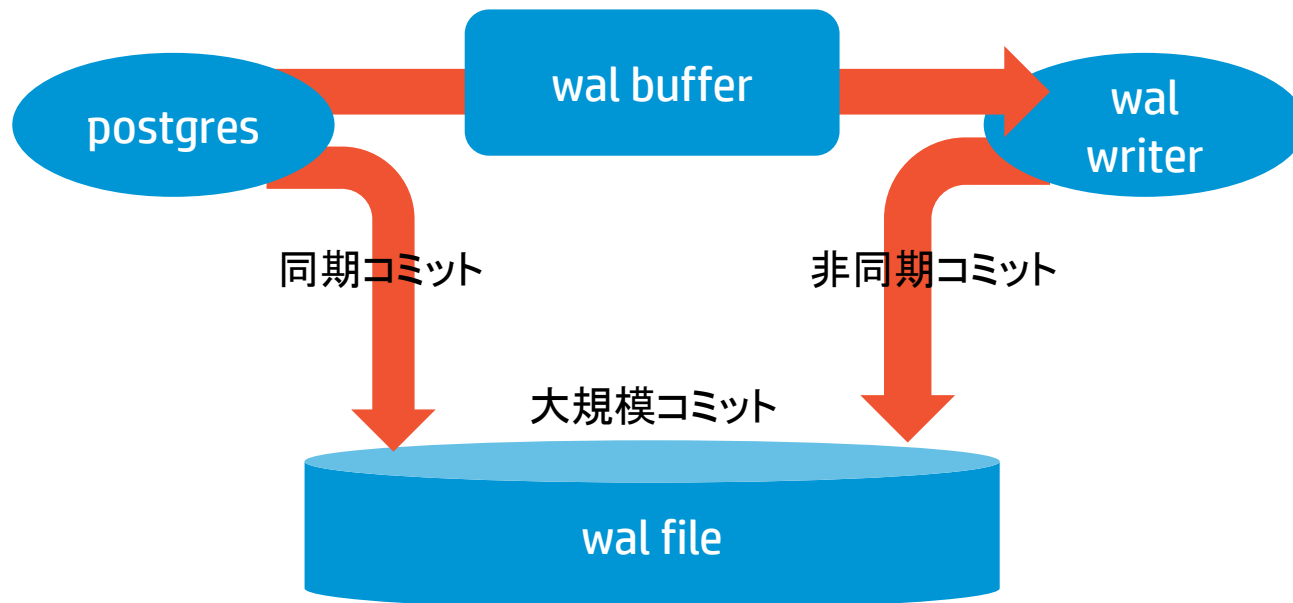
パラメータ	用途	推奨値(デフォルト)
wal_level	WAL出力内容	archive (minimal)
synchronous_commit	トランザクション確定時に同期的にWAL出力	on (on)
wal_writer_delay	wal writer プロセスの活動周期(ミリ秒)	200 (200)

1.4 ストレージ・アーキテクチャ

WAL ファイルの書き込み

postgres プロセスと wal writer プロセスが書き込み

- トランザクション状況やパラメータ `synchronous_commit` の設定により書き込みプロセスは変化
- 小規模な同期コミットの場合、postgres プロセスが直接 WAL ファイルを更新
- 小規模な非同期コミットの場合、wal writer プロセスが WAL ファイルを更新
- トランザクションの更新量が wal buffer に格納できない場合は postgres / wal writer が同期を取りながら wal ファイルを更新



1.4 ストレージ・アーキテクチャ

アーカイブログ

WAL ファイルのコピー

- アーカイブログは、WAL ファイルがすべて書き込まれるとコピーとして作成
 - コピーが完了したWALファイルはファイル名変更
 - チェックポイント発生時やバックアップ実行時にも作成される場合がある
- アーカイブログはバックアップからのリカバリに使用



- パラメータは以下の通り

パラメータ	用途	推奨値(デフォルト)
archive_mode	アーカイブログを出力するか	on (off)
archive_command	アーカイブログのコピーコマンド	'cp %p /arch/%f' ('')
archive_timeout	時間ベースのアーカイブログ出力	0 (0)

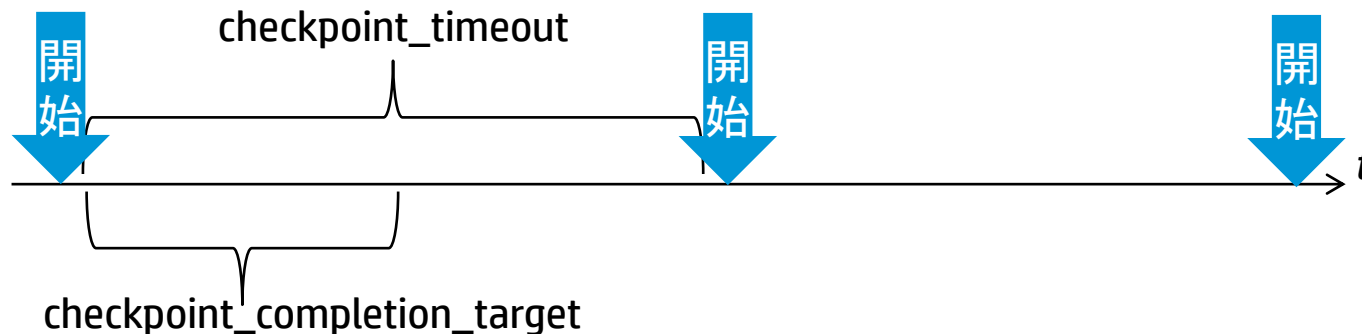
1.4 ストレージ・アーキテクチャ

チェックポイント

メモリーとセグメント・ファイルの同期

- checkpointer プロセスが実行
- 時間ベースのチェックポイント（パラメータ `checkpoint_timeout`）
- WAL ファイル書き込み量によるチェックポイント（パラメータ `checkpoint_segments`）
- チェックポイントのチューニング

チェックポイント間隔	I/O 量	リカバリ時間
間隔を短く	増加	短時間
間隔を長く	減少	長時間



1.4 ストレージ・アーキテクチャ

チェックポイント

パラメータの推奨値

パラメータ	用途	推奨値(デフォルト)
checkpoint_segments	チェックポイントを開始するWALファイル書き込み数	128 (3)
checkpoint_timeout	チェックポイント時間間隔	30min (5min)
checkpoint_warning	頻繁なチェックポイントを警告する間隔	30min (30s)
checkpoint_completion_target	チェックポイント完了までの時間割合	0.9 (0.5)
log_checkpoints	チェックポイント情報のログ出力	on (off)

checkpoint_warning 期間内に WAL ファイルベースのチェックポイントに到達すると以下のログ出力

LOG: **checkpoints are occurring too frequently** (2 seconds apart)

HINT: Consider increasing the configuration parameter "checkpoint_segments".

1.4 ストレージ・アーキテクチャ

セグメント・ファイルの書き込み

チェックポイントは I/O スパイクになりやすい

- writer プロセスが一定間隔毎に少しずつ書き込みを行うことで I/O を平滑化
- 直近の必要とされたページ数の平均と bgwriter_lru_multiplier から計算される値よりも再利用可能なページ数よりも大きい場合にのみ書き込み
- writer プロセスに関するパラメータ

パラメータ	用途	デフォルト
bgwriter_delay	writer プロセスの書き込み間隔	200ms
bgwriter_lru_maxpages	書き込み最大ページ数、0にすると書き込みは行われない	100
bgwriter_lru_multiplier	平均要求ページに掛ける値	2.0

1.4 ストレージ・アーキテクチャ

追記型アーキテクチャ

PostgreSQL の特徴的な動作

- PostgreSQL は追記型アーキテクチャ
 - UPDATE 文でレコード(タプル)が更新されると更新後レコードが追記
 - 変更前レコードには旧データを示すフラグが付与される
 - DELETE 文でレコードが削除されても物理的には削除されない
 - 削除済レコードであるフラグを付与
 - 不要レコードはいずれ削除しなければならない
- あらかじめ追記用の領域を作成しておくことで、更新時の負荷を削減できる(FILLFACTOR)

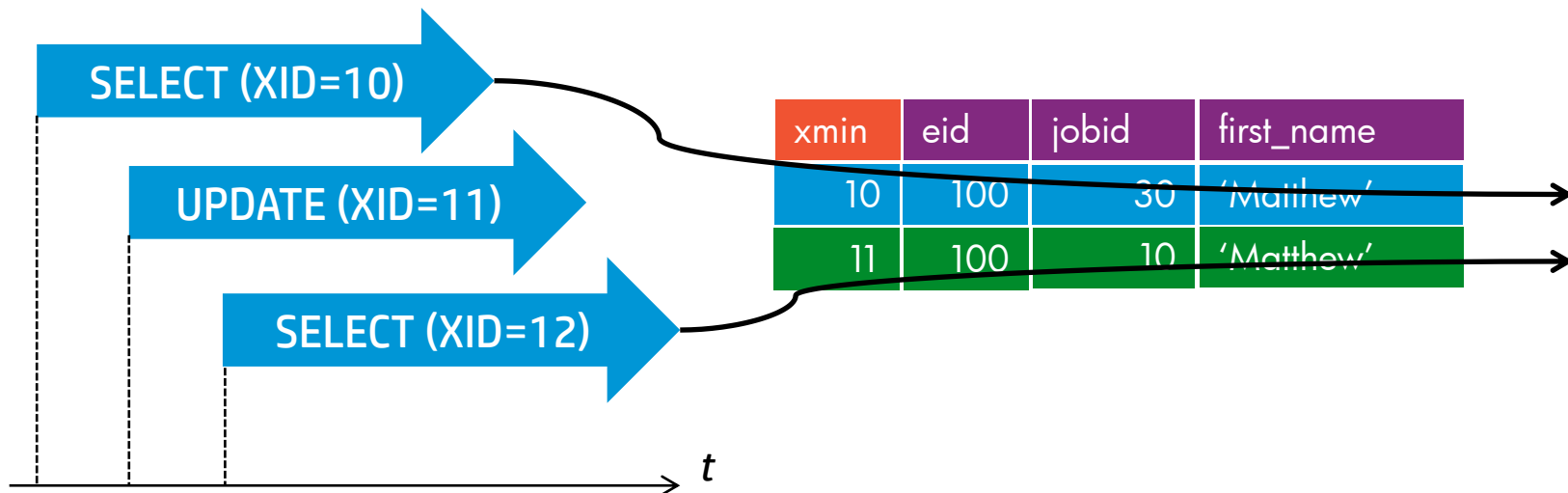


1.4 ストレージ・アーキテクチャ

追記型アーキテクチャ

読み取り一貫性を維持するための動作

- 変更前レコードをページ内に保持することで「読み取り一貫性」を保障
 - SQL 実行開始時点のレコードを参照する
 - コミット前のレコードを参照させない
 - トランザクションの前後関係を示すためにトランザクション ID (XID) を使用
 - 現在の XID は txid_current 関数で取得可能



1.4 ストレージ・アーキテクチャ

VACUUMとは

不要レコードの削除処理

- 不要レコードを削除し、ページに空き領域を作る処理
- 不要レコード数は、`pg_stat_{all | user | sys}_tables` カタログの `n_dead_tup` 列から確認可能
- 実行契機
 - 自動VACUUM
 - VACUUM 文または VACUUM FULL文実行時(テーブル単位、データベース単位)
- VACUUM CONCURRENT
 - ページ内の不要レコードを切り詰めて空き領域を作成
 - 標準設定では自動的に実行
 - ファイル末尾の空きブロックを切り詰め
- VACUUM FULL
 - ブロックをまたがって切り詰めを行うため、セグメント・ファイルを縮小
 - 元ファイルを読んで、新規ファイルを作成するため、一時的に2倍の領域が必要になることがある(処理はテーブル単位)
 - テーブルを排他ロック
 - 手動実行が必要

1.4 ストレージ・アーキテクチャ

VACUUM の自動実行

デフォルトで自動的に行われる

- 統計情報の再計算
 - 以下の計算式以上のレコードが更新 (UPDATE/DELETE) されたテーブルは自動的に 実行
- $$\text{autovacuum_vacuum_threshold} + \text{autovacuum_vacuum_scale_factor} * \text{レコード数}$$
- パラメータはテーブル単位でも決定可能
 - 大規模テーブルではテーブル単位に autovacuum_vacuum_scale_factor を 0 に、autovacuum_vacuum_threshold を適切な値に変更し、頻繁に VACUUM を実行することを推奨
 - 主なパラメータ

パラメータ	用途	デフォルト
autovacuum	自動VACUUM / 統計情報取得を実行するか	on
autovacuum_max_workers	VACUUM ワーカー最大数	3
autovacuum_naptime	自動 VACUUM 実行チェック間隔	1min
autovacuum_vacuum_threshold	自動 VACUUM のための更新レコード数	50
autovacuum_vacuum_scale_factor	自動 VACUUM のための更新レコードの割合	0.2

1.4 ストレージ・アーキテクチャ

HOT と FILLFACTOR

VACUUM に頼らずに領域を再利用

- HOT (Heap on Tuples)
 - テーブル・ページの更新時にインデックス更新を防ぐ仕組み
 - 自動的に実行
 - ページ内に空き領域が必要
- テーブル・インデックス属性FILLFACTOR
 - INSERT 時にページ内にレコードを格納できる割合
 - テーブルのデフォルト値は 100 (%) であるため、ページ全体にレコードが格納される
 - 更新が実行されるテーブルでは 90 (%) 程度に下げることが推奨

1.4 ストレージ・アーキテクチャ

FREEZE

古いトランザクション情報を更新する処理

- PostgreSQL のトランザクションID (XID) は単純増加する 32 ビット整数値
- 32 ビット値を使い切ると循環する
- 古いトランザクション ID を持つレコードを FROZEN XID (=2) に変更する処理を FREEZE と呼ぶ
- 自動的に実行されるが、更新頻度が多いデータベースではメンテナンス時に VACUUM FREEZE 文の実行を考慮する
- 主なパラメータ(9.3.3 でパラメータ追加あり)

パラメータ	説明	デフォルト
vacuum_freeze_table_age	テーブルの全体チェックを実行する世代 標準では VM ファイルを参照して実行される	150000000
vacuum_freeze_min_age	指定値より古い世代のXIDは FrozenXID に変更	50000000
autovacuum_freeze_max_age	自動的に FREEZE 操作が行われるまでの世代数(自動 VACUUM を停止しても実行)	200000000

1.4 ストレージ・アーキテクチャ

統計情報の取得

SQL 文の実行計画を決定するためのデータ

- SQL文の実行計画作成には統計情報を使用
- 統計情報とはテーブルやインデックスのデータ情報
 - レコード数、ブロック数
 - ヒストグラム(データの最頻値、ばらつき)
- 取得契機
 - 自動 VACUUM
 - VACUUM ANALYZE文の実行
 - ANALYZE 文の実行
- 統計情報の確認
 - pg_statistic カタログ
 - pg_stats カタログ (pg_statistic カタログを見やすく整形している)

1.4 ストレージ・アーキテクチャ

統計情報の取得

SQL 文の実行計画を決定するためのデータ

- 統計情報はサンプリングによって行われる
 - サンプリング量はデフォルト 30,000 レコード (MAX(列の STATISTICS 値) × 300 レコード)
 - 列の STATISTICS 値とは？
 - ヒストグラムを収集するバケット数 (データの範囲を入れる箱の数)
 - デフォルト値はパラメータ default_statistics_target で決まる (デフォルト 100)
 - 「ALTER TABLE table_name ALTER column_name SET STATISTICS value」文で変更
 - サンプリング・レコード数の計算式 (src/backend/ commands/analyze.c)

$$\begin{aligned} \text{rows} &= 4 * k * \ln(2 * n / \text{gamma}) / f^2 \\ &= 4 * k * \ln(2 * 1000000 / 0.01) / 0.5^2 = 305.82 * k \end{aligned}$$

k = histogram size = default_statistics_target

n = table rows

gamma = error probability

f = maximum relative error in bin size

- 現状ではテーブルのレコード数を 1,000,000 レコードと想定。10,000,000 レコードの場合、k = 114 程度になる

1.4 ストレージ・アーキテクチャ

統計情報の自動取得

VACUUM と合わせて実行される

- autovacuum worker プロセスが実行
- 統計情報の再計算
 - 以下の計算式以上のレコードが変更 (INSERT / UPDATE / DELETE) されたテーブルは自動的に統計情報を再取得

$\text{autovacuum_analyze_threshold} + \text{autovacuum_analyze_scale_factor} * \text{レコード数}$

- 変更されたレコード数は pg_stat_{user|all|sys}_tables カタログの n_mod_since_analyze 列で確認可能
- テーブル単位でも決められる
 - 大規模テーブルでは列 STATISTICS 値を拡大して、サンプル・レコード数の拡大を推奨
- パラメータ

パラメータ	用途	デフォルト
autovacuum_analyze_threshold	統計情報再計算のための更新レコード数	50
autovacuum_analyze_scale_factor	統計情報再計算のための更新レコードの割合	0.1

1.5 ネットワーク・アーキテクチャ

パラメータと pg_hba.conf ファイル

- postmaster プロセスがクライアントからの接続を待つ
- 接続待ちのポート番号は、パラメータ port で決定される(デフォルト 5432)
 - デフォルトを使用する場合でもパラメータ・ファイルに記述することを推奨
 - パラメータ・ファイルに記述が無い場合、環境変数 PGPORT が使用される
- 接続待ちの TCP/IP アドレスは、パラメータ listen_addresses で決定される(デフォルト localhost)
 - '*' を指定すると全インターフェースが使用される
- 接続クライアントの認証は pg_hba.conf ファイルで決定される
 - 必要最低限の接続許可設定を推奨
- クライアントの切断を検知するために Keep Alive 関連パラメータを設定(デフォルトは OS 設定を継承)

パラメータ	説明	デフォルト
tcp_keepalives_idle	アイドル・セッションにKeep Alive パケットを送信する時間間隔(秒)	0
tcp_keepalives_interval	Keep Alive パケットの再送間隔(秒)	0
tcp_keepalives_count	Keep Alive パケットの再送上限(回数)	0

DEMO

アーキテクチャの確認



2. 安定稼働のために必要な設定

2.1 データベース・クラスタ設計

エンコーディングの決定

データベースに保存する文字コードを決定

- テーブルに対する保存文字コードを指定
- データベース・クラスタ作成時にデフォルト決定、データベース単位に指定可能
- 文字集合が大きい UTF8 を推奨 (Windows Vista で文字集合が JIS X 0213 になった為)
- 異なるエンコーディングでデータベースを作る方法

```
postgres=# CREATE DATABASE eucdb1 ENCODING='eucjp' LC_COLLATE='ja_JP'
```

```
LC_CTYPE='ja_JP' TEMPLATE=template0 ;
```

```
postgres=# \l
```

List of databases

Name	Owner	Encoding	Collate	Ctype	Access privileges
eucdb1	postgres	<u>EUC_JP</u>	ja_JP	ja_JP	
postgres	postgres	UTF8	ja_JP.UTF-8	ja_JP.UTF-8	=c/postgres + postgres=CTc/postgres

2.1 データベース・クラスタ設計

ロケール機能の使用

通貨や言語に応じた大小関係

- ロケールは文字種類の大小関係や区別を示す情報
 - 全角文字列に対する upper / lower 関数による変換
 - ひらがな、カタカナの大小関係
 - money 型列の検索結果に通貨記号がつく
- データベース・クラスタ作成時にデフォルトを決定
- ロケールなしを推奨
 - データベース・クラスタ作成時に --no-locale 指定
 - データベース作成時に LC_COLLATE='C' LC_CTYPE='C' を指定

2.1 データベース・クラスタ設計

ロケール機能の影響

使用することによる影響と設計

- ロケールありの場合、文字列型列に対する LIKE 文で、インデックス範囲検索ができない場合がある
 - CREATE INDEX 文実行時にバイナリ比較オプションを指定することで回避可能
 - 構文 CREATE INDEX インデックス名 ON テーブル名 (列名 オプション)

列データ型	オプション
char	bpchar_pattern_ops
varchar	varchar_pattern_ops
text	text_pattern_ops
name	name_pattern_ops

- バイナリ比較オプションを使用すると、<, > 演算子でインデックスが使用できない

2.1 データベース・クラスタ設計

ページ・チェックサム

データ破損の検知

- データベース・クラスタ作成時に指定（9.3～）
 - initdb コマンドに-kパラメータを指定。デフォルトでは無効
 - pg_controldata コマンドまたはパラメータ data_checksums の設定値を確認（変更不可）
 - ページ書き込み時にチェックサム付与、ページ読み取り時にチェックサム検証
- パフォーマンスへの影響
 - チェックサムを付与してもブロック・サイズが変更されないため I/O 量は変化しない
 - チェックサムの検証／確認のため、CPU 使用率が数 % 上昇する可能性あり

2.2 CORE ファイル設定

トラブル解析のためにCOREファイルは重要

- Red Hat Enterprise Linux 6 の CORE ファイル出力設定
 - Red Hat Enterprise Linux 6 では 標準で ABRT 機能が有効になっている
 - 標準ではサイン済のバイナリのみ CORE を出力

/etc/abrt/abrt-action-save-package-data.conf ファイル

```
OpenPGPCheck = no
```

- CORE ファイルサイズの制限解除

/etc/security/limits.conf ファイル

```
postgres      -            core    unlimited
```

\${HOME}/.bashrc ファイル

```
ulimit -c unlimited
```

2.3 メモリー・オーバーコミット設定

メモリーの確保を実際に必要になるまで遅延する機能

- Red Hat Enterprise Linux 6 では標準でメモリー・オーバーコミットの機能が有効
- 物理メモリーが不足すると OOM Killer により特定プロセスが強制終了
- メモリー・オーバーコミットを解除するために以下の設定を推奨

カーネル・パラメータ	用途	推奨値(デフォルト)
vm.overcommit_memory	大規模メモリー要求を受け入れるか • 0: 発見的なオーバーコミット • 1: 常にオーバーコミット • 2: オーバーコミット無効	2 (0)
vm.overcommit_ratio	メモリー要求を受け入れる割合	99 (50)

2.4 スワップ設定

スワップ

メモリー不足が発生した場合の動作

- メモリー不足が発生すると、メモリー上の情報はスワップアウト
- カーネル・パラメータ `vm.swappiness` はスワップされる頻度を調整
- 以下の設定を推奨

カーネル・パラメータ	用途	推奨値(デフォルト)
<code>vm.swappiness</code>	スワップ処理頻度 <ul style="list-style-type: none">• 0: メモリー不足になるとスワップ• 100: より積極的にスワップ	0 (60)

2.5 I/O 性能の安定化

I/O Scheduler の設定変更

- Red Hat Enterprise Linux のデフォルトである cfq はパフォーマンスは悪くないが、安定しない場合がある
 - 一般的には deadline を推奨
 - SSD 環境では noop が候補となる
- パフォーマンス試験をして決定
- サーバー全体の設定は/etc/grub.conf ファイルを更新

```
kernel /vmlinuz-2.6.32-358.el6.x86_64 ro root=/dev/mapper/vg_rel641-lv_root  
rd_LVM_LV=vg_rel641/lv_root KEYBOARDTYPE=pc KEYTABLE=us  
rd_LVM_LV=vg_rel641/lv_swap rd_NO_DM rhgb quiet initrd /initramfs-2.6.32-  
358.el6.x86_64.img elevator=deadline
```

DEMO

ロケールとエンコーディング



3. 障害の検知と監視

3.1 ログファイルの設定

ログの種類

- サーバーログ
 - パラメータ `logging_collector` を `on` に設定することで取得 (デフォルト `off`)
 - SYSLOG (Windows 環境ではイベントログ) またはテキストファイルを選択可能
 - 標準では `pg_log` ディレクトリに作成される。パラメータ `log_directory` で指定 (データベース・クラスタからの相対、または絶対パス)
 - OSクラッシュ時には保存されていない場合がある (書き込み時に `flush` していないため)
 - ログのエンコードはデータベースと同じ
- `pg_ctl` コマンドのログ
 - インスタンスの起動／停止等のエラー解析に使用
 - ログファイルを指定しない場合は標準エラー (`stderr`) に出力
 - `-l` パラメータでファイル名を指定
 - 既にファイルが存在する場合は追記
 - インスタンス起動時にログが書けない場合は起動エラー


3.1 ログファイルの設定

ログファイルのフォーマット

設定が必要なパラメータ

- ログには「レベル」と「説明文字列」が出力される
- ログの先頭にはパラメータ `log_line_prefix` で指定された文字列が出力される
 - デフォルトは空文字
 - 少なくとも以下のパラメータを指定することを推奨

パラメータ指定	説明	備考
%m	タイムスタンプ	タイムゾーンはパラメータ <code>log_timezone</code>
%a	アプリケーション名	
%u	データベース・ユーザー名	
%d	データベース名	
%r	リモートホスト名 : ポート番号	ローカル接続の場合は <code>[local]</code>

- パラメータ `log_line_prefix` の最後には空白を含める
 - 例 `log_line_prefix = '%m %a %u %d %r '` 

3.1 ログファイルの設定

ログのレベル

下記表の上位ほど影響が大きい

- どのレベルからログに出力するかは、パラメータ `log_min_messages` で指定
- どのレベルからログにSQLを出力するかは、パラメータ `log_min_error_statement` で指定
- どのレベルからクライアントに送信するかは、パラメータ `client_min_messages` で指定
- デフォルト設定を推奨

レベル	用途	備考
PANIC	全セッションが切断される	
FATAL	カレントセッションが切断される	
LOG	管理者向けメッセージ	
ERROR	現在のコマンドが中断される	<code>log_min_error_statement</code> デフォルト
WARNING	ユーザーへの警告	<code>log_min_messages</code> デフォルト
NOTICE	ユーザーへの補助情報	<code>client_min_messages</code> デフォルト
INFO	ユーザーからの要求による情報	
DEBUG1...5	開発者向けのメッセージ	

3.1 ログファイルの設定

ログファイルの出力先

設定が必要なパラメータ

- 出力フォーマットはパラメータ `log_destination` で指定
 - 設定可能な値は `csv`, `syslog`, `stderr` (デフォルト `stderr`)
 - `syslog` を選択した場合にはパラメータ `syslog_facility` (デフォルト `local0`)、`syslog_ident` (デフォルト `postgres`) も指定
- ファイル名はパラメータ `log_filename` で指定する
(デフォルト `postgresql-%Y-%m-%d_%H%M%S.log`)
 - ローテーションを考慮してファイル名を決定
 - 主な指定パラメータ

パラメータ	意味
%Y	4桁西暦
%y	2桁西暦
%m	月番号(01~12)
%d	月内の日付(01~31)
%A	曜日名(英語)

パラメータ	意味
%H	時間(00~23)
%M	分(00~59)
%S	秒(00~59)
%h	時間(00~11)
%p	AM / PM

3.1 ログファイルの設定

ローテーション

- サイズおよび時間でローテーション可能
- サイズでローテーションする場合には、原則としてファイル名に時刻が必要
- ローテーションしようとした場合に、新ファイルが同じ名前になるとローテーションしない
- ログファイルの監視ツールがサポートしていることを確認する
- 関連するパラメータ

パラメータ	説明	デフォルト
log_truncate_on_rotation	ローテーション時にファイル切り詰め	off
log_rotation_age	ローテーション時間を指定	1d
log_rotation_size	ローテーションサイズを指定	10MB

3.1 ログファイルの設定

ログを記録する事象の設定

設定を推奨するパラメータ

パラメータ	説明	推奨値(デフォルト)
log_min_duration_statement	実行時間が長い SQL 文情報	システム依存 (-1)
log_checkpoints	チェックポイント情報	on (off)
log_autovacuum_min_duration	実行時間が長い自動 Vacuum 情報	システム依存 (-1)
log_temp_files	外部ソート情報、work_mem で指定されたメモリ不足の可能性	0 (-1)
log_lock_waits	長時間ロック待ちセッション情報	on (off)

3.2 インスタンスの管理

pg_ctl コマンドの実行結果をチェック

- pg_ctl コマンドの終了ステータス・チェック
 - ステータス0以外はエラー
- インスタンス起動エラーの検知
 - デフォルトではインスタンス起動失敗はチェック不可
 - -w パラメータを指定して同期処理を推奨
 - -l パラメータにログを指定して障害発生時のログ取得を推奨
 - ただし、ログ作成を失敗するとインスタンス起動不可
- インスタンス停止エラーの検知
 - スマート・モード(-m smart)による停止失敗
 - インスタンス停止中ステータスになるため新規ユーザー接続不可
 - -m fast を指定して再度停止処理を実行

3.3 稼働状況の監視

パフォーマンスと障害発生の定期チェック

pg_stat* カタログのチェック

- 正常稼働時の状態を知っておくことが重要 (特にパフォーマンス系)

システムカタログ	確認できる内容	備考
pg_stat_activity	セッション数、実行中のSQL文	COUNT(*) で取得
pg_stat_bgwriter	チェックポイント情報	
pg_stat_archiver	アーカイブ情報 (失敗を検知)	
pg_stat_database	データベース全体の状況、 一時領域 (ディスクソート) の使用状況	パラメータ work_mem を検討
pg_stat_replication	レプリケーション情報	
pg_stat_*_tables	VACUUM 情報、更新レコード情報	
pg_statio_*_tables	I/O 方法 (インデックス検索、全権検索)	
pg_stat_*_indexes	インデックス使用状況、アクセス状況	インデックスが有効か
pg_statio_*_indexes	インデックス I/O 状況	
pg_stat_user_functions	ファンクション実行状況	

3.4 リモート・インスタンスの監視

pg_isready コマンド

- pg_isready コマンドを実行すると、リモート・インスタンスの稼働をチェックできる(9.3～)
- 以下のパラメータが指定できる。データベース名 -d, ユーザー名 -U は無視される

パラメータ	説明	デフォルト
-h ホスト名	接続先ホスト名	localhost
-p ポート番号	接続先ポート番号	5432
-t タイムアウト	接続待ちタイムアウト(秒)	3
-q	画面表示を抑制	-

戻り値	説明	備考
0	インスタンス稼働中で接続可能	
1	インスタンス稼働中だが接続不可	
2	サーバーと通信不可	
3	パラメータ不正	

3.5 バックアップ

ストレージ障害やユーザーの誤更新に対応

pg_basebackup コマンドによるバックアップ(9.1～)

安全なオンライン・バックアップ

- 空きディレクトリを指定して、安全にフル・バックアップを実行可能
- tar フォーマット、元と同じフォーマットを選択可能(圧縮も可)
- レプリケーション用の基準データ作成として使用可能
- wal sender プロセスを使用するため、パラメータ max_wal_senders を調整
- -x パラメータを指定してバックアップ完了後に WAL スイッチを行うことを推奨
- PostgreSQL 9.4 の新機能
 - --xlogdir パラメータにより、WAL 用ディレクトリを指定可能
 - --tablespace_mapping パラメータにより、外部テーブル空間用ディレクトリの変更可能

3.5 バックアップ

不要なアーカイブログの削除

pg_archivecleanup コマンドによりアーカイブログ・ファイルの削除

安全なアーカイブログの削除

- アーカイブログを安全に削除できる contrib モジュール (9.0～)
- アーカイブログ・ファイルは、バックアップと最新データベースの間を埋める重要なファイル
- フル・バックアップ以前のアーカイブログのみ削除可能
 - pg_basebackup コマンドが完了すると、アーカイブログ用ディレクトリに拡張子 .backup のファイルが作成される
 - 最新の .backup ファイルを指定することで安全にアーカイブを削除することができる

```
#!/bin/bash
ARCHDIR=/usr/local/pgsql/arch
LASTWALPATH=`ls $ARCHDIR/*.backup | sort -r | head -1`
LASTWALFILE=`basename $LASTWALPATH`

pg_archivecleanup $ARCHDIR $LASTWALFILE
exit $?
```

4. 障害発生時の動作と対処

4.1 プロセス障害

バックエンド・プロセスの異常終了

- postmaster プロセス障害
 - インスタンス全体が停止
 - 実行中のトランザクションは次回起動時にロールバック
 - 再起動等はクラスタリング・ウェア、pgpool-II 等の役割
- postgres プロセス障害
 - 全セッションはリセットされ、障害発生直後のSQL文はエラーになる
 - 実行中の全トランザクションはロールバック
 - アプリケーションによる障害検知と対処が必要
 - 以下のログが出力される

LOG: server process (PID 3571) was terminated by signal 9: Killed

LOG: terminating any other active server processes

WARNING: terminating connection because of crash of another server process

DETAIL: The postmaster has commanded this server process to roll back the current transaction and exit, because another server process exited abnormally and possibly corrupted shared memory.

HINT: In a moment you should be able to reconnect to the database and repeat your command.

4.1 プロセス障害

バックエンド・プロセスの異常終了

- デフォルトでは自動的に再起動する(パラメータ restart_after_crash = on)
 - restart_after_crash = off の場合は、全プロセス停止
- writer, wal writer, checkpointner, autovacuum launcher プロセスは全体で再起動
 - 実行中の 全トランザクションはロールバック
 - クライアントとの接続は維持されるが、再起動直後の SQL 文はエラーになる
 - アプリケーションの対処が必要
 - その他のプロセス(logger, wal sender, wal sender, wal receiver 等)は単純に再起動
- レプリケーション環境ではスレーブ・インスタンスで起動している startup process が異常終了するとスレーブ・インスタンス全体が停止
- 以下のログが出力される(wal writer の場合)

LOG: WAL writer process (PID 3929) was terminated by signal 9: Killed

LOG: terminating any other active server processes

WARNING: terminating connection because of crash of another server process

DETAIL: The postmaster has commanded this server process to roll back the current transaction and exit, because another server process exited abnormally and possibly corrupted shared memory.

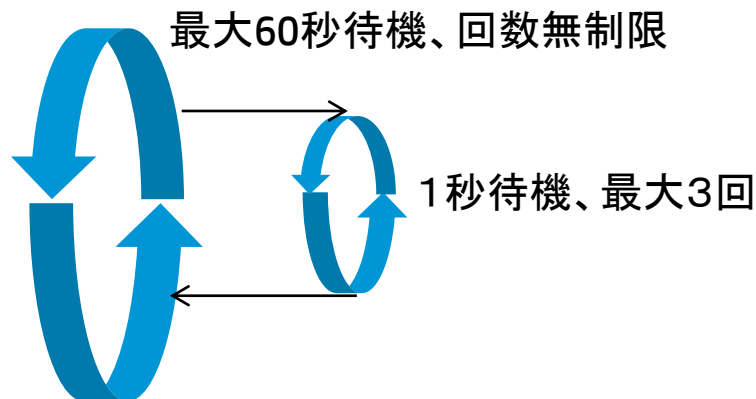
HINT: In a moment you should be able to reconnect to the database and repeat your command.

4.1 プロセス障害

アーカイブログの作成エラー

アーカイブログ保存先のデバイスフル等によるアーカイブログ作成エラー発生時

- パラメータ `archive_command` で指定されたコマンドが0以外の値を返すとアーカイブ失敗と見なされる
 - デバイスフル、プロセス起動エラー、パラメータ設定失敗、ディレクトリ削除等が原因
 - `pg_stat_archiver` カタログからも検知できる
 - アーカイブ化に失敗した場合、`pg_xlog` ディレクトリの WAL ファイルは再利用されない
 - このためアーカイブ化に失敗すると `pg_xlog` ディレクトリに大量の WAL ファイルが作成される
- ユーザーのトランザクションは停止しない
- 失敗したファイルについては自動的に再実行
 - 1秒間隔で、3回トライしてエラー→最大1分待つ→繰り返し



4.1 プロセス障害

アーカイブログの作成エラー

ログの例

- エラー・レベルが LOG / WARNING になる場合がある

```
cp: cannot create regular file `/arch/00000001000000040000006A': Permission denied
2014-11-07 16:39:57.089 JST LOG: archive command failed with exit code 1
2014-11-07 16:39:57.089 JST DETAIL: The failed archive command was: cp
pg_xlog/00000001000000040000006A arch/00000001000000040000006A
cp: cannot create regular file `/arch/00000001000000040000006A': Permission denied
2014-11-07 16:39:58.106 JST LOG: archive command failed with exit code 1
2014-11-07 16:39:58.106 JST DETAIL: The failed archive command was: cp
pg_xlog/00000001000000040000006A arch/00000001000000040000006A
cp: cannot create regular file `arch/00000001000000040000006A': Permission denied
2014-11-07 16:39:59.122 JST LOG: archive command failed with exit code 1
2014-11-07 16:39:59.122 JST DETAIL: The failed archive command was: cp
pg_xlog/00000001000000040000006A arch/00000001000000040000006A
2014-11-07 16:39:59.123 JST WARNING: archiving transaction log file
"00000001000000040000006A" failed too many times, will try again later
```

4.2 ストレージ障害

ブロック破損

- セグメントからの読み込み時にチェック
- チェックサム・エラーが検知されたテーブルは以降使用不可
 - パラメータ `ignore_checksum_failure` を on 指定することでチェックサムを無視できる(デフォルト off)
 - 対処
 - バックアップからリカバリ
 - チェックサムを無視したデータを使って再構築
- チェックサム・エラーが検知されると以下のログが出力される

```
WARNING: page verification failed, calculated checksum 1094 but expected 51919
ERROR: invalid page in block 0 of relation base/24577/24578
STATEMENT: select * from demo1 ;
```

4.2 ストレージ障害

WAL 保存先ストレージ故障

- WAL 障害
 - トランザクション確定時にインスタンス異常終了
 - 前回のチェックポイントから最新トランザクションまでの更新情報は喪失
 - 対処
 - pg_resetxlog コマンドにより既存データを救済 (-f オプションを指定)
 - ストリーミング・レプリケーションによるスレーブを昇格
- 以下のログが出力

2014-11-07 17:21:17.912 JST PANIC: could not open transaction log file

"pg_xlog/00000001000000040000006D": Permission denied

2014-11-07 17:21:17.918 JST LOG: startup process (PID 10750) was terminated by signal 6: Aborted

2014-11-07 17:21:17.918 JST LOG: aborting startup due to startup process failure

4.2 ストレージ障害

OS クラッシュによるファイル削除

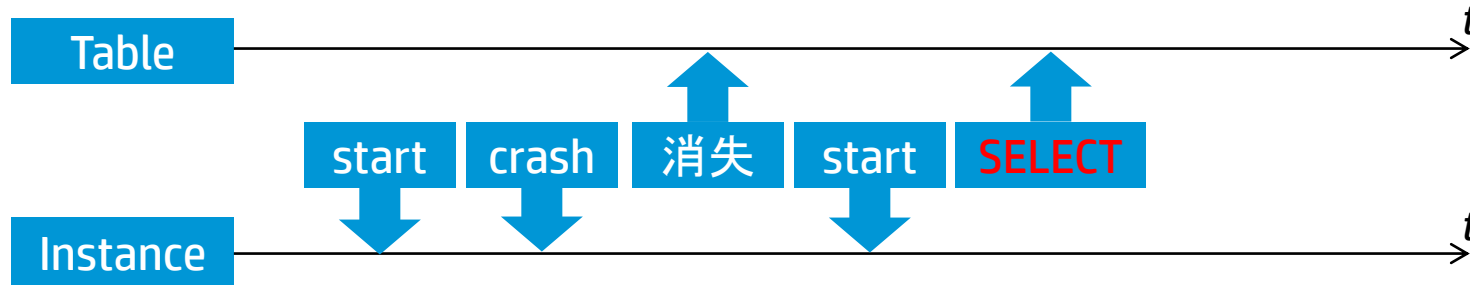
削除されるファイルにより影響が異なる

主なファイル	影響	対処
pg_control	インスタンス起動不可	リストア + pg_resetxlog コマンドによるWAL再作成
WALファイル	インスタンス起動不可	pg_resetxlog コマンドによるWAL再作成
PG_VERSION	インスタンス起動不可	リストア
VM / FSM	影響なし	自動 VACUUM による生成
セグメントファイル	対象テーブルに依存	リストア + リカバリ
postgresql.conf	インスタンス起動不可	リストア
pg_hba.conf	インスタンス起動不可	リストア

4.2 ストレージ障害

OS クラッシュによるファイル削除

変更されていないセグメント・ファイルの削除



- インスタンスは正常起動
- SQL 文はエラーになるが、セッションは維持
- 以下のログが出力

ERROR: could not open file "base/16385/24628": そのようなファイルやディレクトリはありません

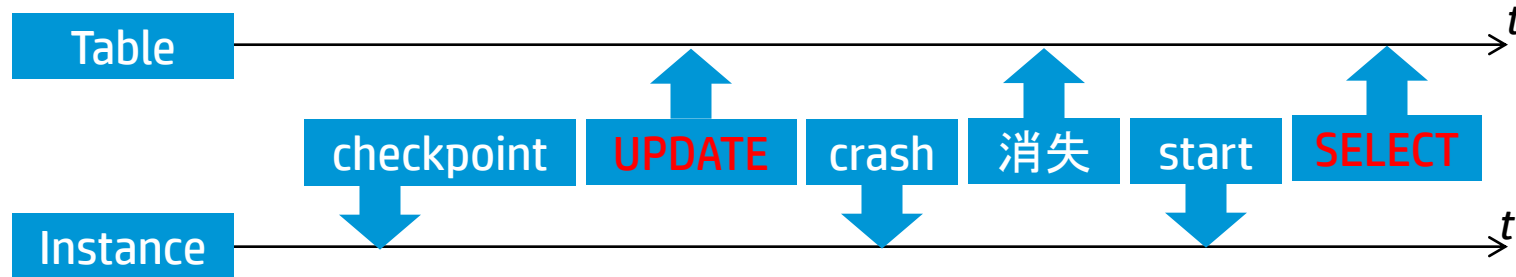
STATEMENT: select * from drop1 ;

- バックアップからデータファイルをリストアし、リカバリを実施することで復旧可能

4.2 ストレージ障害

OS クラッシュによるファイル削除

チェックポイント以降更新されているファイルの削除



- チェックポイント以降のトランザクション情報はWALファイルに保存
- インスタンス再起動時にクラッシュ・リカバリが正常に完了
- 再度実行するとSQL文は成功(エラーは発生しない)
- WALに含まれていない、削除されたファイルに含まれていたレコードは失われる
- 現状の PostgreSQL では検知不可

DEMO

クラッシュとファイル削除



まとめ

まとめ

まとめ

- アーキテクチャと OS の関係の正しい理解が必要
- システムの安定は、日常的な監視が不可欠
- すべてのリスクを防ぐことは不可能だが、可能性を削減することは可能

Thank you

篠田 典良
テクノロジー事業統括
サービス統括本部 オープンソース部
シニアアーキテクト



Noriyoshi.Shinoda@hp.com

日本ヒューレット・パカード株式会社
本社
〒136-8711
東京都江東区大島2-2-1

