45分でわかる PostgreSQLの仕組み

2012/10/17

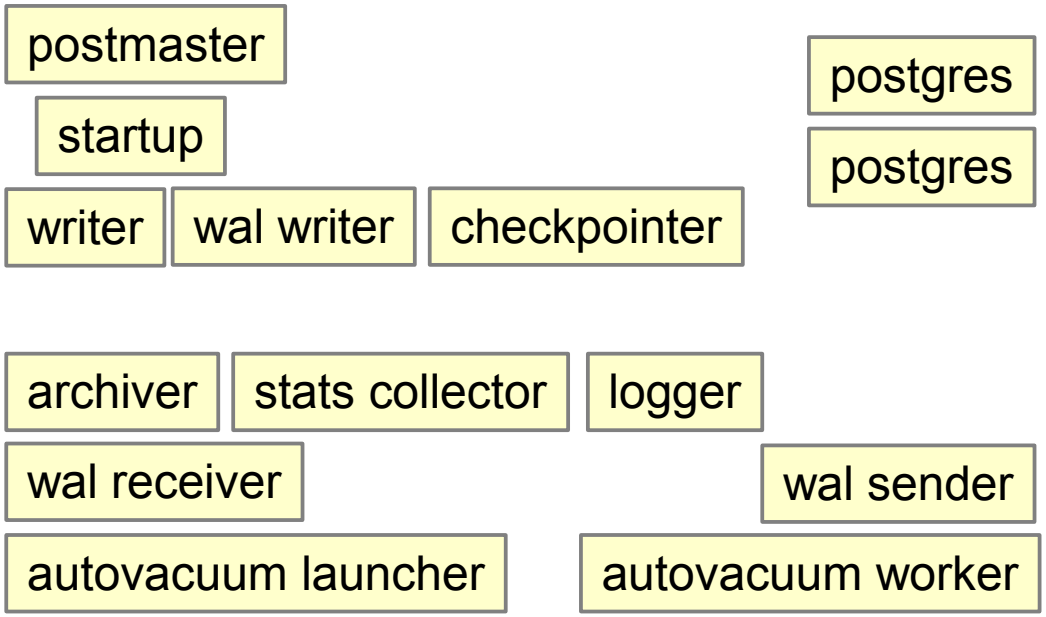
SRA OSS, Inc. Japan
山田 努
tsutomu@sraoss.co.jp

序:目次

- 各サーバプロセス
- クライアント/サーバ通信
- データ格納、ストレージ上の記録方式
- SQL実行
- トランザクション処理
- トランザクションログとリカバリ、レプリケーション

プロセス構成

- postmaterがメイン
- 常駐支援プロセス
  - 一部設定に依存
- 接続要求に応じて子プロセスを起動



各プロセスの役割

- プロセス

postmaster	**PostgreSQL**の親プロセス。接続を待ち受けるプロセス。
postgres	個々のクライアントの要求を処理するプロセス。
writer	共有バッファをディスクに書き出すプロセス。
wal writer	**WAL**書き込みを行うプロセス。
checkpointer	**CHECKPOINT**処理を行うプロセス。(9.2)
archiver	**WAL**ログをアーカイブするプロセス。
logger	**PostgreSQL**のログをファイルへ書き出すプロセス。
stats collector	統計情報を収集するプロセス。
autovacuum launcher	不要領域を監視するプロセス。
autovacuum worker	自動**VACUUM**を実行するプロセス。複数起動することがある。
wal sender	**WAL**をスタンバイサーバへ転送するプロセス。
wal receiver	**WAL**をマスターサーバから受信するプロセス。

プロセス間通信

- signalによる通知(INT,TERM,USR1)
  - Windows版はEvent を使う
  - latch (9.1~) pipeを使ってsignalを捉える仕組み
- SysV IPC
  - 共有メモリ
  - セマフォ
- localhost udp
  - statistic collecter

クライアント/サーバ

- ソケット通信(tcp/stream)
  - unix domain
  - IPv4,v6
- 接続要求を受けてから fork() する
  - ユーザ認証は、子プロセス側で行なわれる
  - pg_hba.conf

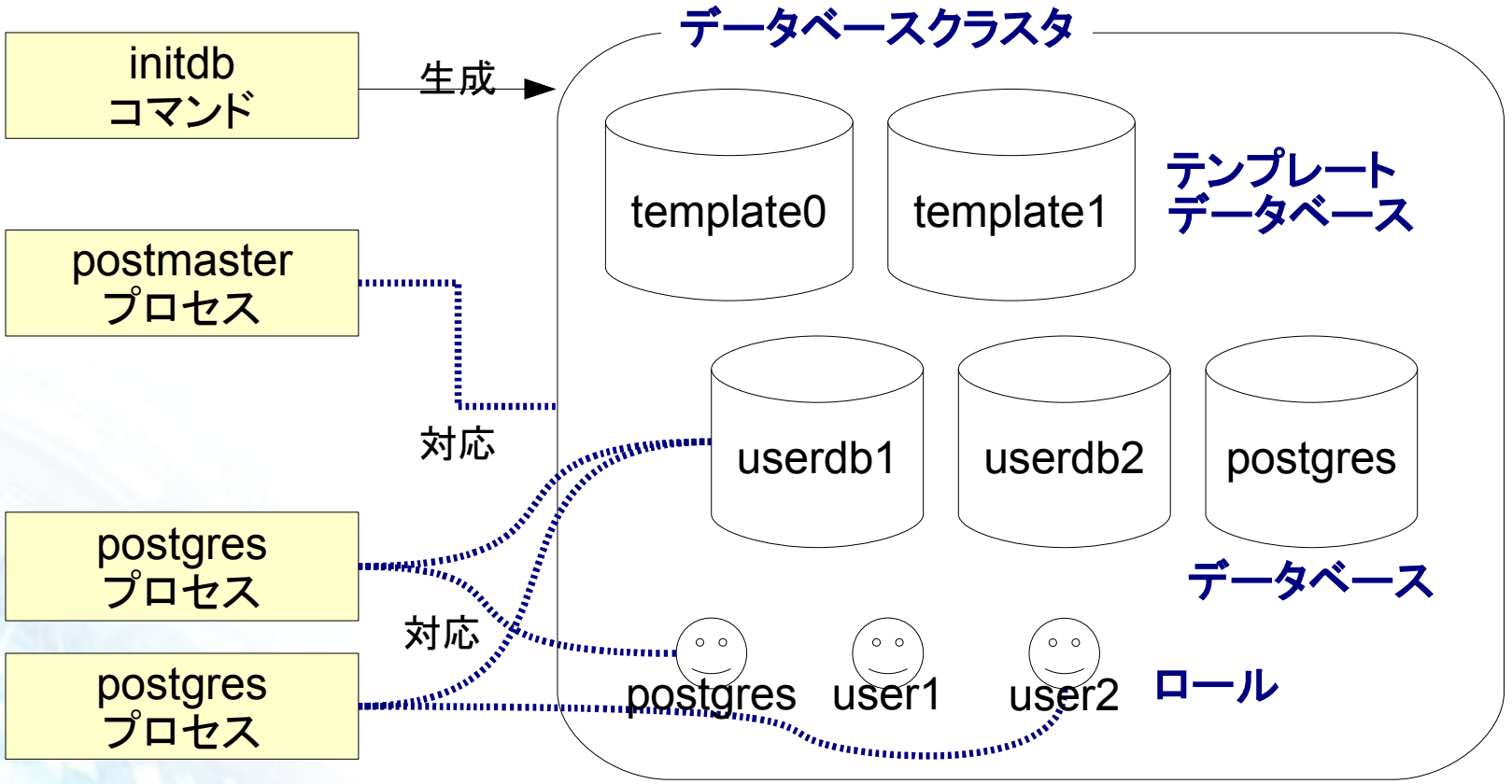
通信プロトコル

- プロトコルバージョン3.0
- 認証
- キャンセルキー
  - 別の接続経路を使って、中止依頼を送信する
- 簡易問い合わせ、SQL文字列で送信
- 拡張問い合わせ、PREPAREとEXECUTE
  - parse/bind/excute
- notice バックエンドからの通知

データベースクラスタ

- データベースクラスタ
  - ファイルシステムに記録されるデーター式
    - initdbコマンドで作成される
  - プロセスの集合体
    - postmasterプロセス他 (前述)
- データベースクラスタで共有される情報
  - ロール、データベース
  - global/ ディレクトリ = pg_globalテーブルスペース

プロセスとデータベースクラスタ



データ格納

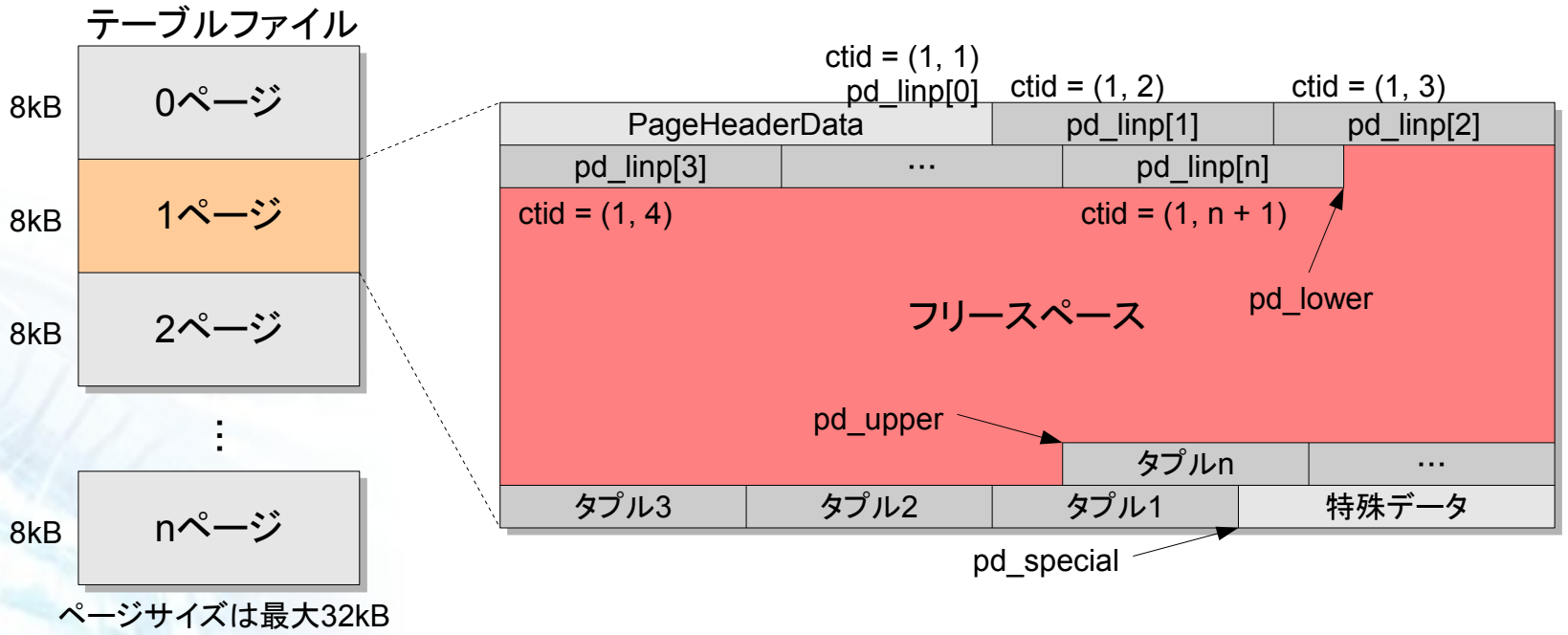
- ファイルシステム上の1ディレクトリ
  - ディレクトリ構成
    - global/
      - pg_control …
    - base/数字(データベース)/数字(リレーション)
    - pg_clog/
    - pg_xlog/16進数24桁(トランザクションログ)
    - pg_tblspc/シンボリックリンク(テーブルスペース)
    - *.conf (設定ファイル)

テーブルデータ

- リレーション単位(テーブル、インデックス)
  - pg_class.relfilenode, oidで管理
    - pg_filenode.map, pg_relation_filenode()
  - 1ファイル 1GBに分割
  - TOAST (The Oversized-Attribute Storage Technique)
- _fsm (Free Space Map),_vm (Visibility Map)
  - タプルが見える = 有効である = VACUUM不要
  - index only scanでも利用

ブロック構造

- タプル(行)データはブロック単位で管理



VACUUM

- 追記型アーキテクチャ
  - 世代管理
  - トランザクションID周回問題
- 不要領域の回収
- HOT (Heap Only Tuple)
  - 更新内容を同じページ内に記録してリンクを作る。インデックスの更新を不要にする。適時不要領域の回収が可能。
  - fill factor

SQL実行処理

- parse
  - prepare/bind
- rewrite
- planner/optimizer
- executer

パース、リライト

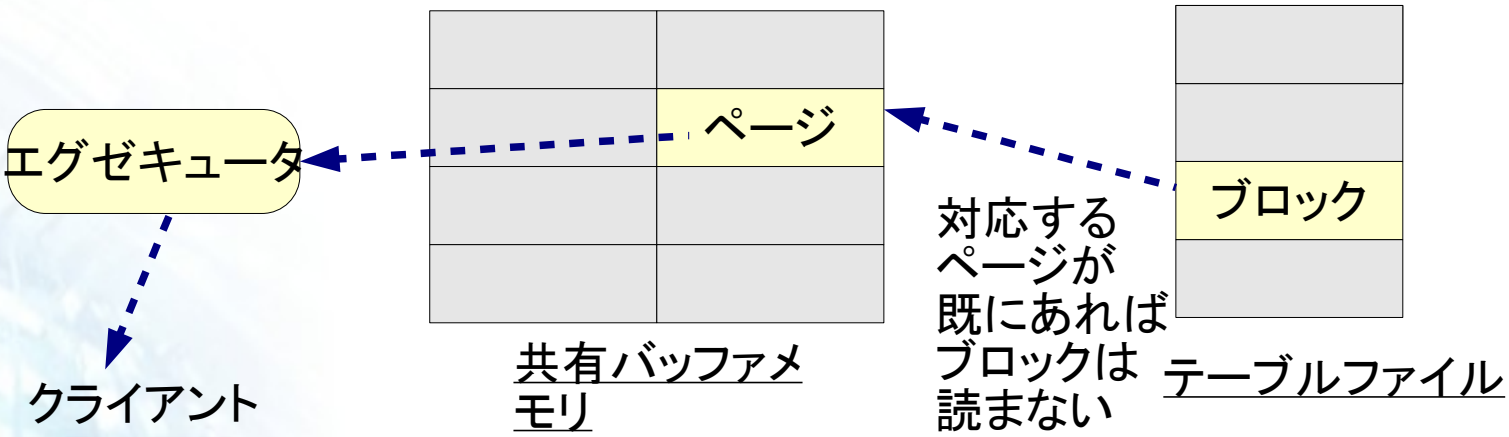
- パースツリー
  - デバックログ
    - debug_print_parse / debug_print_rewritten / debug_print_plan
- SQLの書き換え(rewrite)
  - pg_rules (pg_rewrite)
  - pg_views

最適化処理

- optimizer
  - ルールによるクエリ書き換え
  - FROM JOINの組合せ、順序
- コスト計算
  - スキャン方法の選択
  - ANALYZEによる統計情報
- ヒント句はない
  - 設定パラメータの影響も少ない

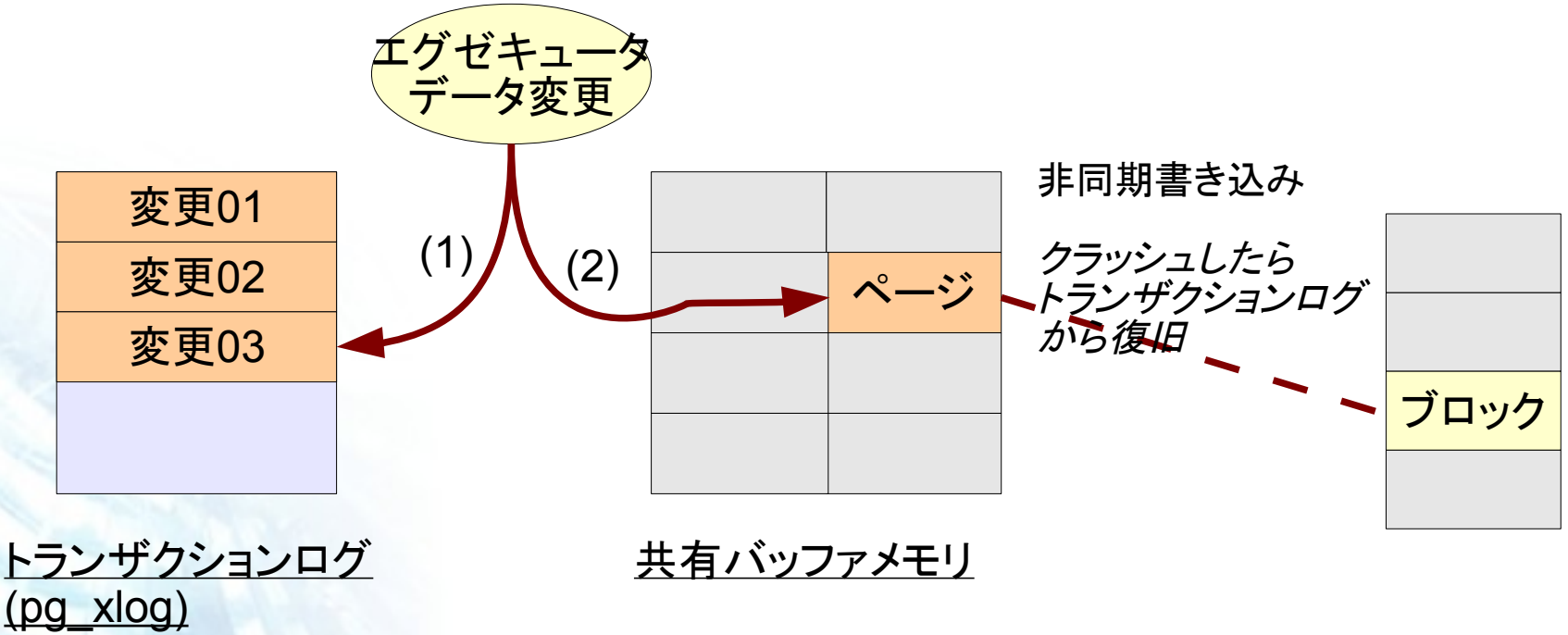
executor

- プランに従って、データの参照・更新を行なう
- 共有バッファの管理



データ書き込み

- 更新はトランザクションログと共有バッファに行なう



トランザクション隔離と同時実行

- MVCC (MultiVersion Concurrency Control)
  多版型同時実行制御
- 追記型
  - トランザクションIDを使った管理
- タプルは特殊なカラムを持っている
  - xmin,xmax,cmin,cmax
  - 作られた時、消された時、その操作をしたトランザクション
    - コマンドID (ex) CURSOR FETCHで見えるか見えないか

コミットログ

- pg_clog/
  - トランザクションの状態
  - 実行中 / COMMITED / ABORTED

- 参考文献
  - Mvcc Unmasked - Bruce Momjian
    - January, 2012
    - http://momjian.us/main/writings/pgsql/mvcc.pdf

トランザクションログ

- pg_xlog/
- WAL (write ahead log)
  - 何を実行したかの記録
    - バッファに対する変更内容を記録
    - XLogInsert()
  - 確実に記録するために同期書き込み
    - 複数の実行バックエンドのための処理が入る
    - 共有バッファの更新前に処理される(共有バッファの同期は後述)

リカバリ

- recoveryモード
  - プログラム起動時の初期処理
- WALファイルを元に再実行(replay)
  - standbyモードやreplicationは、常時リカバリ状態になっている

同期処理

- 共有バッファの更新内容をディスクへ書き込む
- checkpoint
  - 過去のトランザクションログが不要になる
    - WALのリサイクル
    - WALが不足すると強制checkpoint
    - 最後のcheckpoint以降の更新がクラッシュリカバリに必要
- 共有バッファが不足した時は随時書き出す
- バックグラウンド書き込み
  - writerプロセス

レプリケーション

- streaming replication
  - ネットワーク経由でWALを受信
  - 非同期・同期

- 詳しくは、別のところで…

終り

- 各項目についての解説は他にも色々とありますので、探してみて下さい。
- 他データベースでの各処理の仕組みと比較するのも良いだろう。

ご静聴ありがとうございました。