SRA OSS,INC.

LPI-JAPAN

OSS-DB Exam Gold 技術解説無料セミナー

2018/12/2

SRA OSS, Inc. 日本支社
OSS技術本部 技術部 PostgreSQL技術グループ
佐藤 友章

目次

- OSS-DB技術者認定資格
- PostgreSQL内部構造編
  - メモリ/プロセス/ストレージ
  - ストレージ上の物理配置
  - データの読み取り/書き込み
- PostgreSQL性能分析編
  - 稼働状況の確認
  - 実行時統計情報
  - 実行計画
- PostgreSQLパラメータチューニング編
  - 共有バッファの設定
  - ワークメモリ・メンテナンスワークメモリの設定
  - その他の設定

オープンソースデータベース（OSS-DB）に関する技術と知識を認定するIT技術者認定

OSS-DB /Silver

データベースシステムの設計・開発・導入・運用ができる技術者

OSS-DB /Gold

大規模データベースシステムの
改善・運用管理・コンサルティングができる技術者

OSS-DB技術者認定資格の必要性

商用/OSSを問わず様々なRDBMSの知識を持ち、データベースの構築、運用ができる、または顧客に最適なデータベースを提案できる技術者が求められている

OSS-DB Gold認定エンジニア

SRA OSS, INC. OSS-DB

大規模データベースシステムの改善、運用管理、コンサルティングができる技術者

- RDBMSとSQLに関する知識を有する
- オープンソースデータベースに関する深い知識を有する
- オープンソースを利用して大規模なデータベースの運用管理ができる
- オープンソースを利用して大規模なデータベースの開発を行う事ができる
- PostgreSQLなどのOSS-DBの内部構造を熟知している
- PostgreSQLなどのOSS-DBの利用方法やデータベースの状態を検証してパフォーマンスチューニングをすることができる
- PostgreSQLなどのOSS-DBの利用方法やデータベースの状態を検証してトラブルシューティングをすることができる

OSS-DB Exam Gold出題範囲

■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
- データベースの構造
- 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
- その他の性能監視

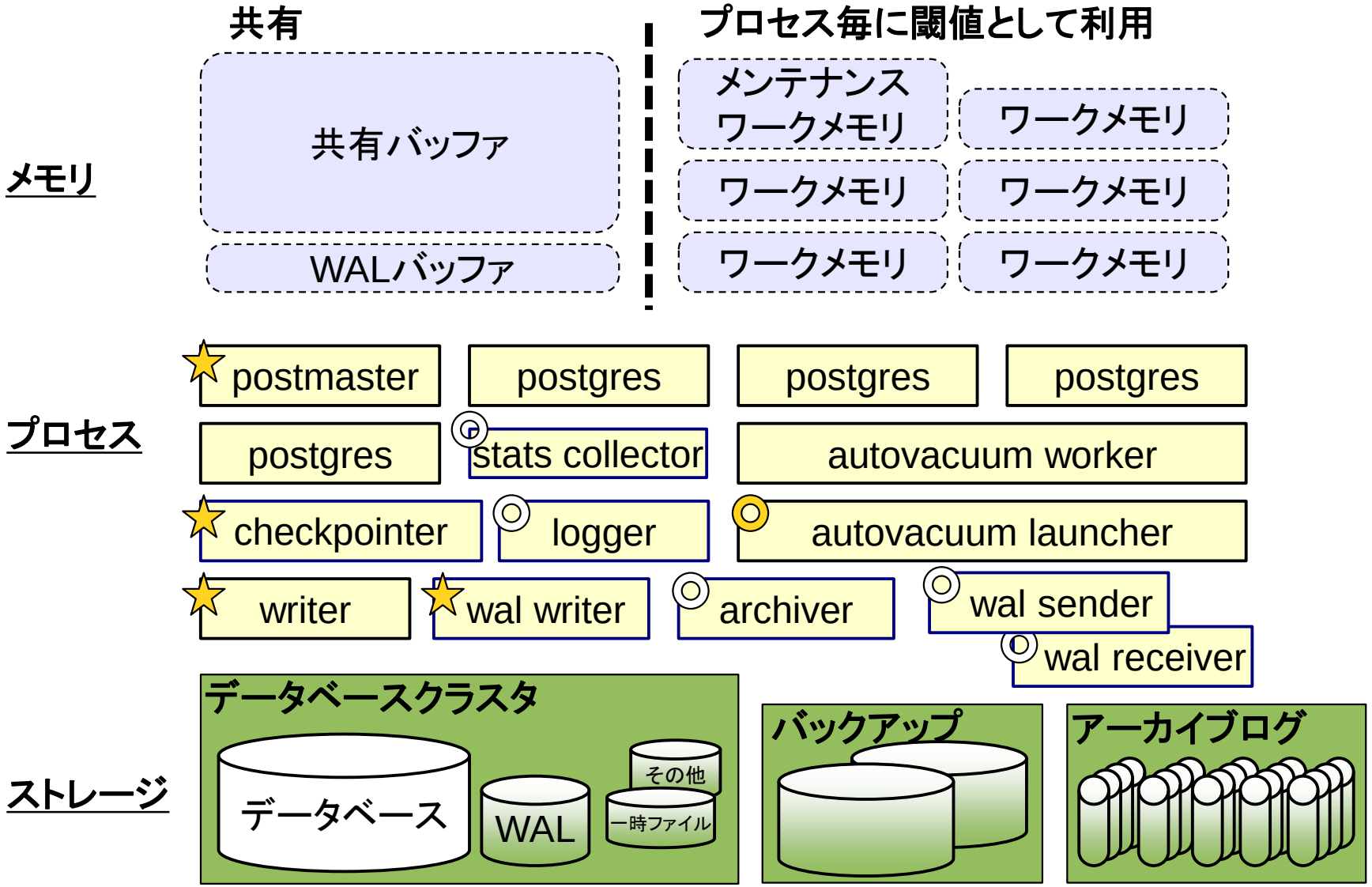
■パフォーマンスチューニング（20%）
- 性能に関するパラメータ
- チューニングの実施

■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

OSS-DB ExamはPostgreSQL 9.0以上を基準のRDBMSとして採用
※2016年現在、9.4まで対応

～PostgreSQL内部構造編～
メモリ/プロセス/ストレージ

メモリ/プロセス/ストレージ



■必ず起動するプロセス ★

postmaster	PostgreSQLの親プロセス。接続を待ち受けるプロセス
stats collector	統計情報を収集するプロセス
wal writer	WAL（Write-Ahead Log）の書き込みを行うプロセス
writer	共有バッファ変更内容をディスクに書き出すプロセス
checkpointer	チェックポイント処理を行うプロセス

```
$ ps xf
   PID TTY      STAT   TIME COMMAND
(省略)
 64722 pts/1    S      0:00 /usr/pgsql-9.4/bin/postgres
 64723 ?        Ss     0:00  ¥_ postgres: logger process
 64725 ?        Ss     0:00  ¥_ postgres: checkpointer process
 64726 ?        Ss     0:00  ¥_ postgres: writer process
 64727 ?        Ss     0:00  ¥_ postgres: wal writer process
 64728 ?        Ss     0:00  ¥_ postgres: autovacuum launcher process
 64729 ?        Ss     0:00  ¥_ postgres: stats collector process
```

プロセスと設定

■設定によって起動するプロセス ◎◎

archiver	WALをアーカイブするプロセス（archive_mode = on）
logger	PostgreSQLのログをファイルに書き出すプロセス（logging_collector = on）
autovacuum launcher	データの更新を監視し、autovacuum workerプロセスを起動するプロセス（autovacuum = on）
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

■常駐せず都度起動するプロセス

postgres	個々のクライアントの要求を処理するプロセス（max_connections = 100）
autovacuum worker	自動バキュームを実行するプロセス（autovacuum_max_workers = 3）

メモリの種類

■共有メモリとして確保

共有バッファ（shared_buffers = 128MB）	データベースの読み書きに使われる共有メモリ。実メモリが1GB以上の場合は1/4程度。Windows上では512MB以内に設定
WALバッファ（wal_buffers = -1）	WAL書き込みに使われる共有メモリ。デフォルト-1はshared_buffersの1/32が自動設定。WALファイルのサイズと同じ16MBが最大値

```
[postgres]$ ipcs -m
------ 共有メモリセグメント --------
キー      shmid      所有者   権限      バイト   nattch     状態
(省略)
0x0052e2c1 40108046   postgres  600        41279488   5
```

■子プロセス毎に確保するメモリ

work memory（work_mem = 4MB）	ソート処理やハッシュ作成処理に使われるメモリ。不足すると一時ファイルを作成して処理する。max_connenctionsとの関係を考慮。
maintenance work memory（maintenance_work_mem = 64MB）	バキュームやインデックス作成に使われるメモリ。autovacuum_max_workersとの関係を考慮

ストレージ上の物理配置

データベースクラスタ（$PGDATAディレクトリ）

```
PG_VERSION
base
global
pg_clog
pg_dynshmem
pg_hba.conf
pg_ident.conf
pg_log
pg_logical
(省略)
pg_stat_tmp
pg_subtrans
pg_tblspc
pg_twophase
pg_xlog
postgresql.auto.conf
postgresql.conf
postmaster.opts
postmaster.pid
```

```
$ ls $PGDATA
PG_VERSION      pg_logical      pg_subtrans
base            pg_multixact    pg_tblspc
global          pg_notify       pg_twophase
pg_clog         pg_replslot     pg_xlog
pg_dynshmem     pg_serial       postgresql.auto.conf
pg_hba.conf     pg_snapshots    postgresql.conf
pg_ident.conf   pg_stat         postmaster.opts
pg_log          pg_stat_tmp     postmaster.pid
```

■設定ファイル

- postgresql.conf
  - PostgreSQLの設定ファイル
- pg_hba.conf
  - クライアント認証

データベースクラスタ

```
PG_VERSION
base
global
pg_clog
pg_dynshmem
pg_hba.conf
pg_ident.conf
pg_log
pg_logical
(省略)
pg_stat_tmp
pg_subtrans
pg_tblspc
pg_twophase
pg_xlog
postgresql.auto.conf
postgresql.conf
postmaster.opts
postmaster.pid
```

baseディレクトリ

- 各データベースに対応したディレクトリ配置
- oid2nameでデータベース名を特定

```
1/
13051/
13056/
16384/
```

```
$ oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  13056       postgres  pg_default
  13051      template0  pg_default
      1      template1  pg_default
  16384           test  pg_default
```

テーブルファイルの物理配置

データベースクラスタ

```
PG_VERSION
base
global
pg_clog
pg_dynshmem
pg_hba.conf
pg_ident.conf
pg_log
pg_logical
(省略)
pg_stat_tmp
pg_subtrans
pg_tblspc
pg_twophase
pg_xlog
postgresql.auto.conf
postgresql.conf
postmaster.opts
postmaster.pid
```

■データベースディレクトリ

- テーブル、インデックス等のデータファイルが格納される（8kB単位）
- oid2nameでテーブルファイル名を特定

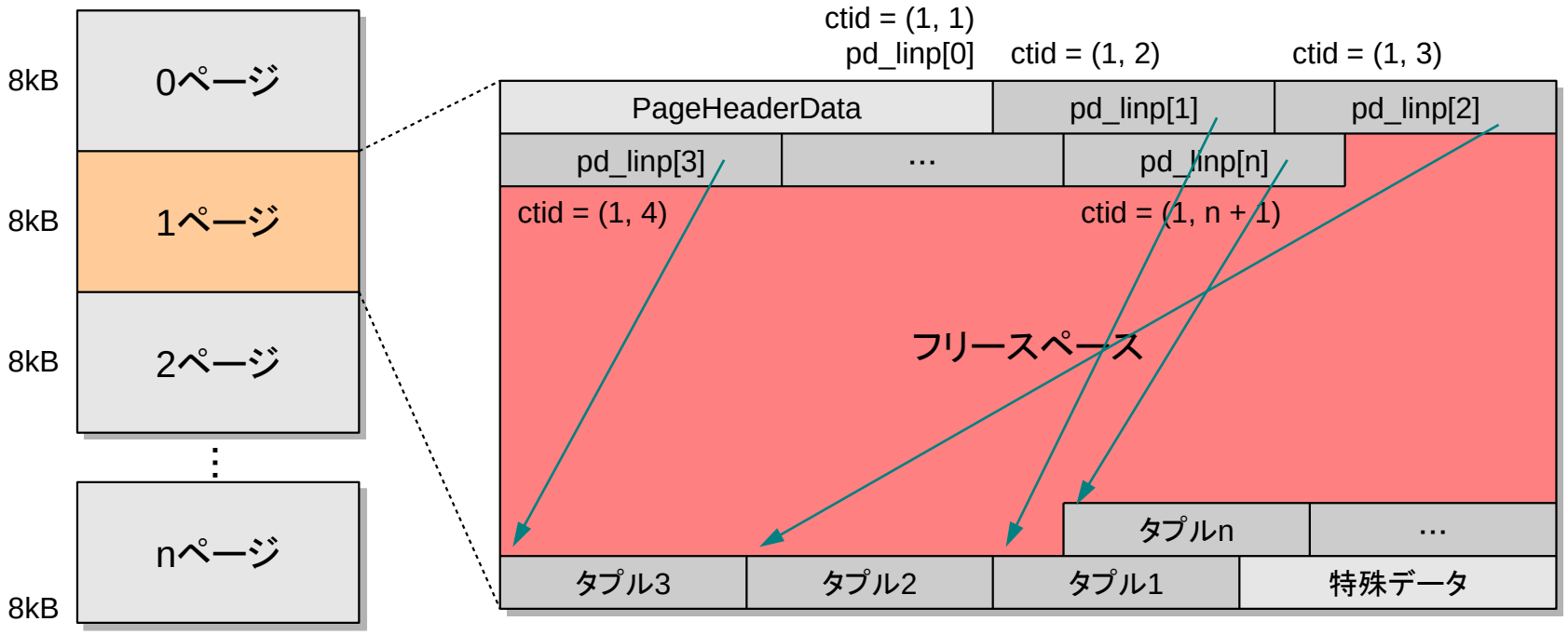
```
1/
13051/
13056/
16384/
```

```
  :
16385
16387
16387_fsm
16387_vm
16391
16393
16394
  :
pgsql_tmp/
```

```
$ oid2name -d test -i
From database "test":
  Filenode  Table Name
----------------------
     16387           t1
     16385   t1_id_seq
     16394     t1_pkey
```

■pgsql_tmpディレクトリには、work_memが不足した場合に一時ファイルが作成される

■タプル（行データ）は8kBのブロック単位でファイルに格納




```
=# SELECT *, ctid FROM t1 WHERE id = 10;
 id |               val                |  ctid
----+----------------------------------+--------
 10 | d3d9446802a44259755d38e6d163e820 | (0,10)
(1 row)
```

WALファイルの物理配置

データベースクラスタ

```
PG_VERSION
base
global
pg_clog
pg_dynshmem
pg_hba.conf
pg_ident.conf
pg_log
pg_logical
(省略)
pg_stat_tmp
pg_subtrans
pg_tblspc
pg_twophase
pg_xlog
postgresql.auto.conf
postgresql.conf
postmaster.opts
postmaster.pid
```

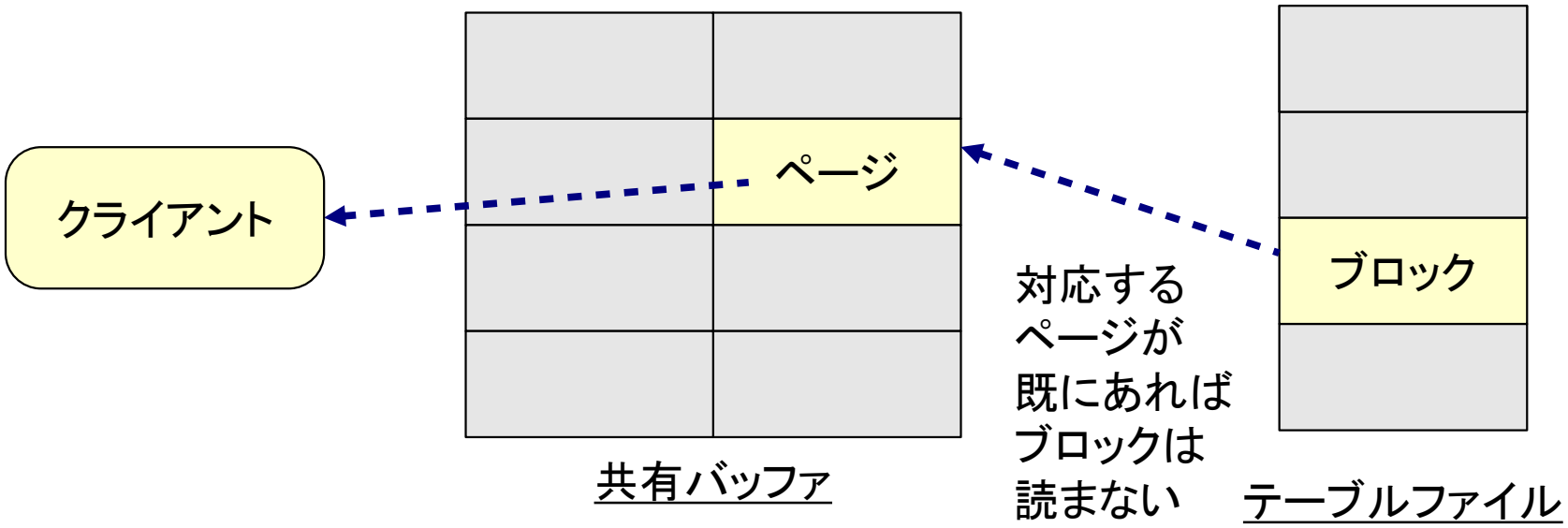
WAL（ログ先行書き込み）ファイル

- トランザクションログとも呼ばれる
- 1ファイル16MB

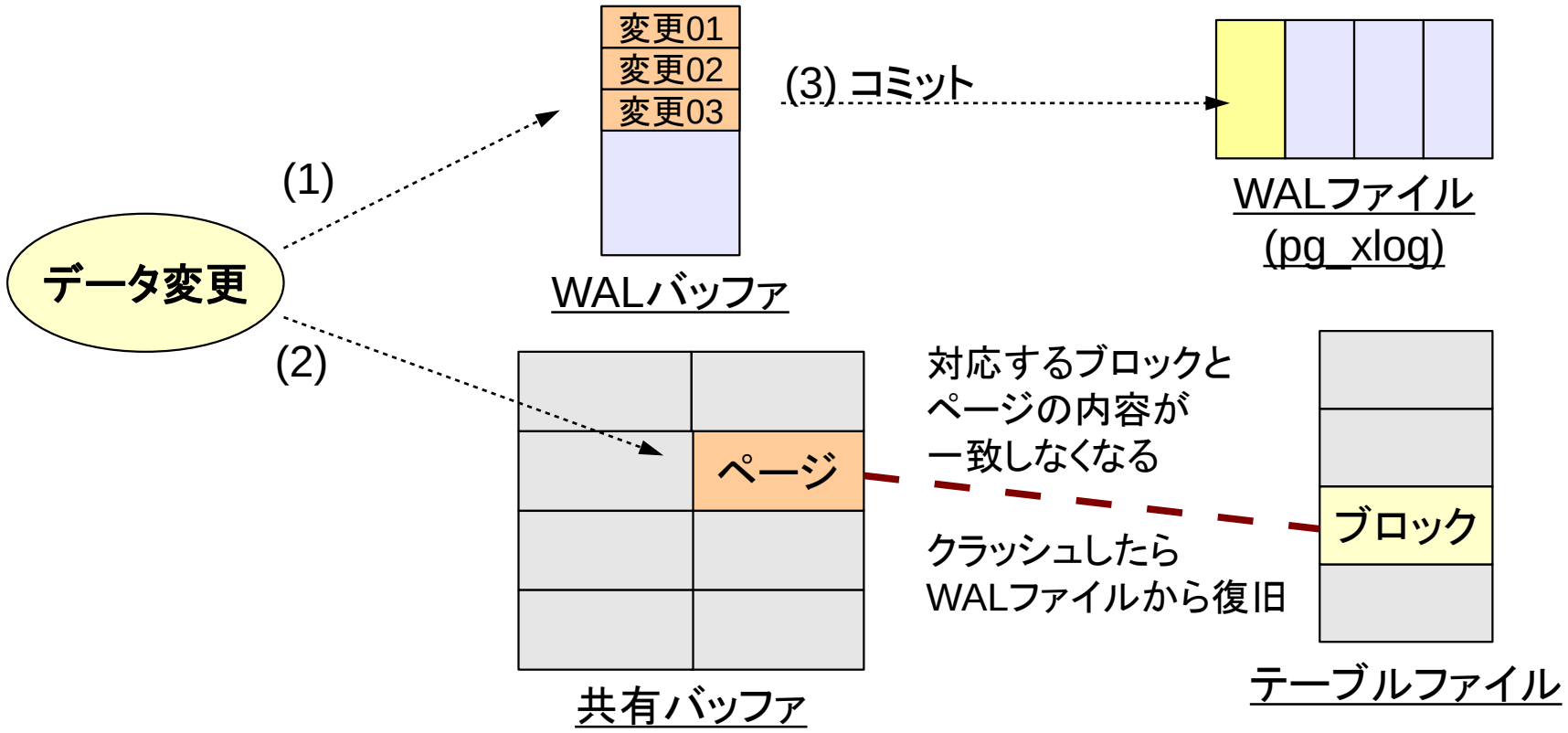
```
$ ls $PGDATA/pg_xlog
000000010000000000000014  000000010000000000000019
000000010000000000000015  00000001000000000000001A
000000010000000000000016  00000001000000000000001B
000000010000000000000017  archive_status
000000010000000000000018
```

データの読み取り/書き込み

- テーブルファイル上のブロックは共有バッファを介して読み込み複数のプロセスで共有する
- ページはブロックをメモリ上にコピーしたもの

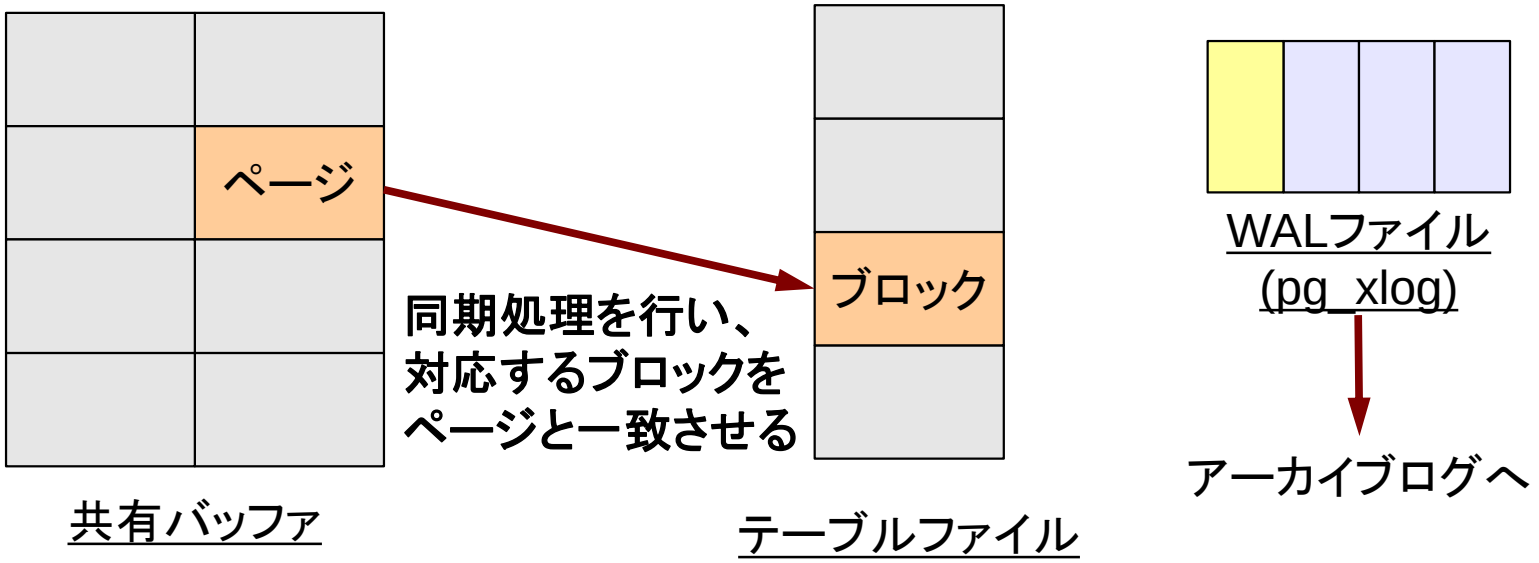


■データ変更はWALバッファと共有バッファに行う
■コミット時にWALファイルに書き込み



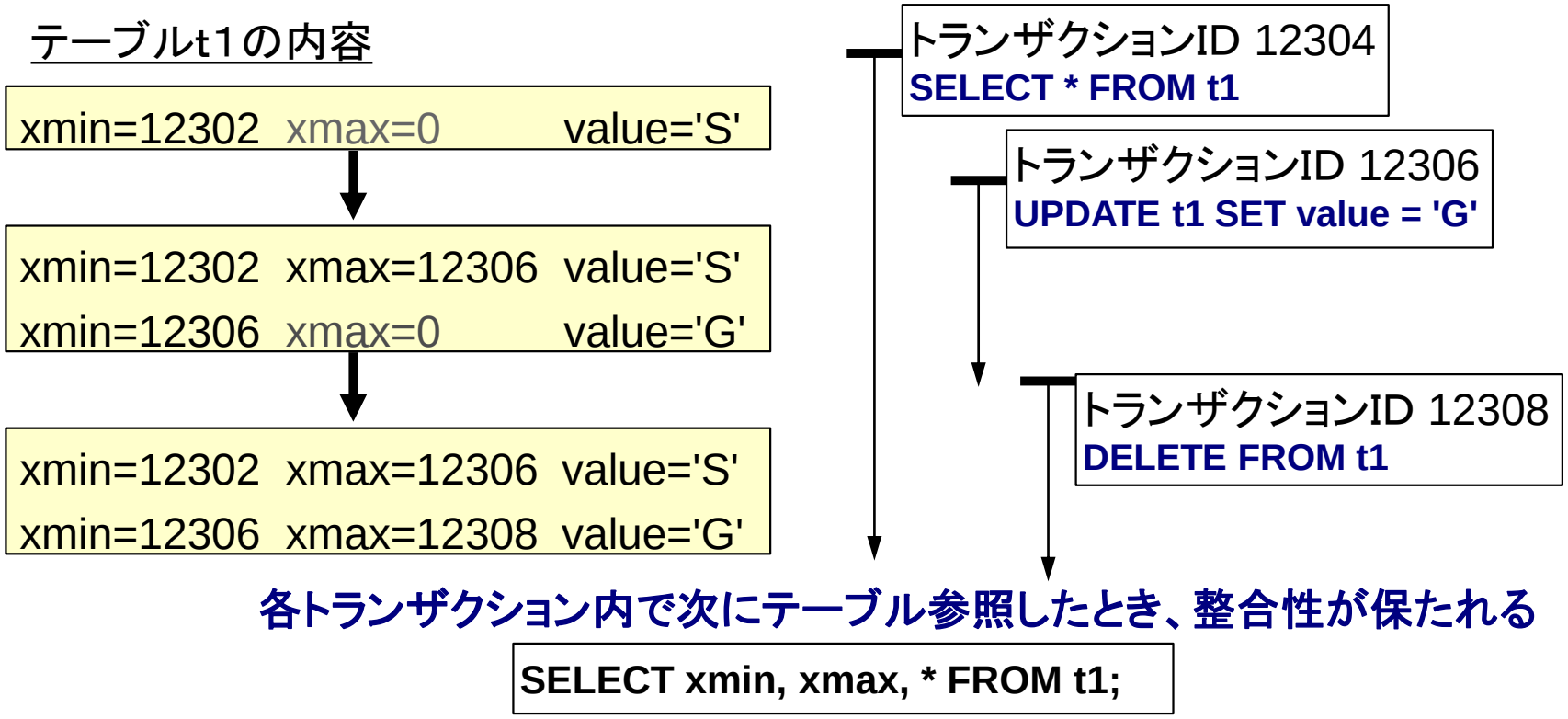
■共有バッファの変更内容をディスクに反映
- チェックポイント
  - checkpoint_timeout = 5min、checkpoint_segments = 3
- キャッシュ追い出し（すべてのページが変更された場合）
- writerプロセスがバックグラウンドで同期処理を行う

■ディスク反映済みのWALファイルは不要になる



■トランザクションIDを用いてバージョン管理

■追記型の書き込み



各トランザクション内で次にテーブル参照したとき、整合性が保たれる

```
SELECT xmin, xmax, * FROM t1;
```

～PostgreSQL性能分析編～
稼働状況の確認

稼働状況の確認

■性能チューニングを実施するために現状を分析し、性能低下の最大の要因を特定

- OSのシステム監視ツール
  - メモリの利用状況
  - CPUの利用率
  - ディスクI/O
- PostgreSQLの機能
  - ログから調査
  - 実行時統計情報の確認

⇒本セミナーでは実行時統計情報についてご紹介

実行時統計情報

実行時統計情報

- 統計情報コレクタ（stats collectorプロセス）によって収集されるデータベースの動作状況に関する情報
  - 収集された統計情報はビューや関数等で参照できる
- 統計情報の収集はデフォルトで有効

```
#track_counts = on
#track_activities = on
```

pg_stat_activity 用

- 統計情報はデータベース単位でリセットできる

```
=# SELECT pg_stat_reset();
```

pg_stat_activity


```
=# SELECT * FROM pg_stat_activity;
-[ RECORD 1 ]----+--------------------------------
datid            | 12788
datname          | postgres
pid              | 32465
usesysid         | 10
usename          | postgres
application_name |
client_addr      | 127.0.0.1
client_hostname  |
client_port      | 33846
backend_start    | 2013-08-15 17:13:51.2
xact_start       | 2013-08-15 17:14:00.9
query_start      | 2013-08-15 17:14:00.9
state_change     | 2013-08-15 17:14:00.9
waiting          | f
state            | active
backend_xid      |
backend_xmin     | 1952
query            | UPDATE pgbench_accour
```

- データベースのOID
- データベース名
- プロセスID
- ユーザのOID
- ユーザ名
- アプリケーション名
- クライアントのIPアドレス
- クライアントのホスト名
- クライアントのTCPポート番号
- プロセス開始時間
- 現在のトランザクション開始時間
- 現在のクエリの開始時間
- state が最後に更新された時間
- 現在ロック待ちであるか
- active, idle など状態を表示
- トランザクションID
- 次トランザクションID（xmin）
- 最後に実行 or 実行中のクエリ

pg_stat_activity

■実行中のSQLに関する情報をプロセスごとに表示

- psコマンドなどと組み合わせてプロセスの特定が可能
- xact_start、query_startなどから長時間実行したまま応答のないSQLの特定が可能
- waiting列でロック待ちの確認

pg_stat_database

```
=# SELECT * FROM pg_stat_database;
-[ RECORD 3 ]--+------------------------------
datid          | 13056
datname        | postgres
numbackends    | 0
xact_commit    | 2680
xact_rollback  | 0
blks_read      | 412541
blks_hit       | 226183
tup_returned   | 8508657
tup_fetched    | 19284
tup_inserted   | 7301180
tup_updated    | 25
tup_deleted    | 283
conflicts      | 0
temp_files     | 2
temp_bytes     | 128139264
deadlocks      | 0
blk_read_time  | 0
blk_write_time | 0
stats_reset    | 2018-10-09 12:41:
```

- データベースのOID
- データベース名
- データベースへの接続数
- コミットされたトランザクション数
- ロールバックされたトランザクション数
- ディスクから読み取られたブロック数
- バッファキャッシュに存在したブロック数
- 取得された行数
- 抽出された行数
- INSERTされた行数
- UPDATEされた行数
- DELETEされた行数
- スタンバイサーバでリカバリ処理と競合した回数
- 問合せにより作成された一時ファイル数
- 問合せにより作成された一時ファイルの累積サイズ
- 検知されたデッドロック回数
- ブロックの読み取りに費やされた累積時間（ミリ秒）
- ブロックの書き込みに費やされた累積時間（ミリ秒）
- 統計情報が最後にリセットされた時間

pg_stat_database

■データベースに関する統計情報ビュー

- numbackends以外の項目は最後にリセットされてからの累積値を表示
  - 設定変更時はカウンタをリセットする
- blks_read、blks_hitにてデータベースごとのキャッシュヒット率を確認
  - blks_read / (blks_hit + blks_read)
  - ただしblks_hitにはOS側のキャッシュヒットは含まれていない
- temp_files、temp_bytesにてwork_memの不足状況を確認

pg_stat_user_tables


```
=# SELECT * FROM pg_stat_user_tables;
-[ RECORD 1 ]-------+------------------------------
relid               | 16387
schemaname          | public
relname             | t1
seq_scan            | 1
seq_tup_read        | 0
idx_scan            | 1
idx_tup_fetch       | 1
n_tup_ins           | 100000
n_tup_upd           | 0
n_tup_del           | 0
n_tup_hot_upd       | 0
n_live_tup          | 100000
n_dead_tup          | 0
n_mod_since_analyze | 0
last_vacuum         | 2018-10-09 12:50:22.28
last_autovacuum     |
last_analyze        |
last_autoanalyze    | 2018-10-09 12:49:19.18
vacuum_count        | 1
autovacuum_count    | 0
analyze_count       | 0
autoanalyze_count   | 1
```

- デーブルのOID
- 属しているスキーマ名
- テーブル名
- 実行されたシーケンシャルスキャンの回数
- シーケンシャルスキャンで取得された有効行数
- 実行されたインデックススキャンの回数
- インデックススキャンで取得された有効行数
- INSERTされた行数
- UPDATEされた行数
- DELETEされた行数
- HOTによる（INDEXの更新を伴わない）更新行数
- 有効行数
- 無効行数
- 最後にANALYZEが実行されてからの変更行数
- 最後に実行された手動VACUUMの時間
- 最後に実行された自動VACUUMの時間
- 最後に実行された手動ANALYZEの時間
- 最後に実行された自動ANALYZEの時間
- 実施された手動VACUUM数
- 実施された自動VACUUM数
- 実施された手動ANALYZE数
- 実施された自動ANALYZE数

pg_stat_user_tables

ユーザが作成したテーブルに関する行単位の統計情報ビュー

- n_live_tup、n_dead_tupにて有効行、無効行を確認
- last_autovacuum、last_autoanalyzeにて自動VACUUMの実行状況を確認
- pg_stat_sys_tables：システムテーブルについて
  pg_stat_all_tables：すべてのテーブルについて
- pg_statio_user_tablesでは、ブロック単位の統計情報が確認できる

pg_stat_user_indexes

```
=# SELECT * FROM pg_stat_user_indexes;
-[ RECORD 1 ]-+--------
relid         | 16387
indexrelid    | 16394
schemaname    | public
relname       | t1
indexrelname  | t1_pkey
idx_scan      | 1
idx_tup_read  | 1
idx_tup_fetch | 1
```

- インデックスが張られているテーブルのOID
- インデックスのOID
- 属しているスキーマ名
- インデックスが張られているテーブル名
- インデックス名
- インデックススキャンが実行された回数
- インデックススキャンによって取得されたノード数
- インデックススキャンによって抽出された行数

ユーザが作成したインデックスに関する行単位の統計情報ビュー

- idx_scanにてインデックスの利用状況を確認
- pg_stat_sys_indexes：システムインデックスについて
  pg_stat_all_indexes：すべてのインデックスについて

実行計画

EXPLAIN

- SQL実行にあたり、内部的にどのような処理方式を組み合わせて実行するかを事前に見積もる
  - 選択できるすべての実行計画を検証
- 実行計画の確認
  - EXPLAIN：実行計画を表示
  - EXPLAIN ANALYZE：実行計画と実行結果に基づく情報

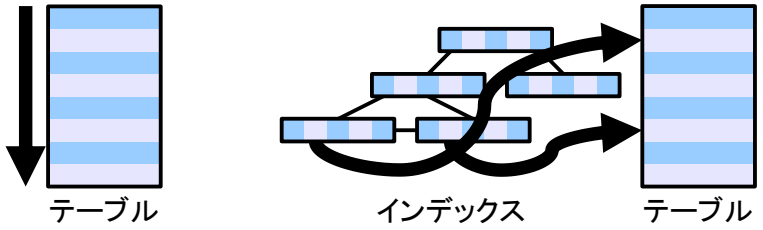
```
=# EXPLAIN SELECT * FROM pgbench_accounts
           JOIN pgbench_branches USING (bid) WHERE aid < 1000;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=1.61..74.17 rows=1106 width=457)
   Hash Cond: (pgbench_accounts.bid = pgbench_branches.bid)
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts
                                          (cost=0.42..57.78 rows=1106 width=97)
         Index Cond: (aid < 1000)
   ->  Hash  (cost=1.08..1.08 rows=8 width=364)
         ->  Seq Scan on pgbench_branches  (cost=0.00..1.08 rows=8 width=364)
(6 rows)
```

■計画タイプ（例：Hash Join）

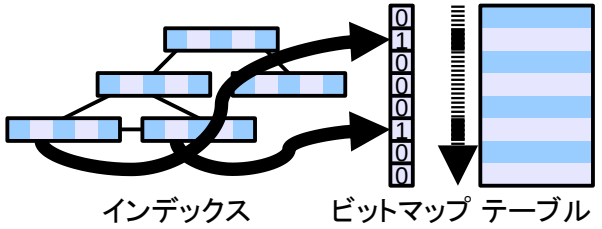
- 内部的な処理の方式
- 同じ処理であっても異なる方式が存在する

```
Hash Join  (cost=1.61..74.17 rows=1106 width=457)
   Hash Cond: (pgbench_accounts.bid = pgbench_branches.bid)
```

■スキャン方式

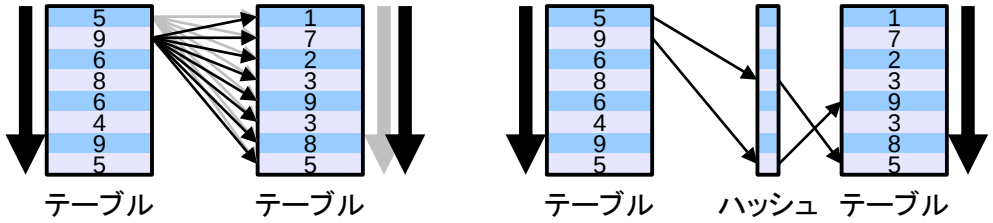


Seq Scan　　Index Scan

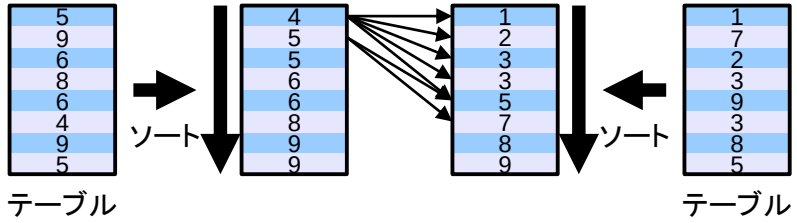


Bitmap Index/Heap Scan

■結合方式



Nested Loop　　Hash Join



Merge Join

■コスト（例：cost=1.61..74.17）

- シーケンシャルアクセスでディスクから1ページ読み取るコストを1とした相対的な数値を表示
- 「最初の行を取得するまでの指定コスト」と「すべての行を取得するまでの推定コスト」（≠実行時間）
- 下位の計画ノードにおける推定コストが含まれる

■行数（例：rows=1106）

- 取得される推定行数

■行の平均サイズ（例：width=457）

- 取得される行の平均サイズ（バイト）

```
Hash Join  (cost=1.61..74.17 rows=1106 width=457)
   Hash Cond: (pgbench_accounts.bid = pgbench_branches.bid)
```

EXPLAIN ANALYZE


```
postgres=# EXPLAIN ANALYZE
           SELECT * FROM pgbench_accounts
           JOIN pgbench_branches USING (bid) WHERE aid < 1000;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=1.61..74.17 rows=1106 width=457)
            (actual time=0.058..3.221 rows=999 loops=1)
   Hash Cond: (pgbench_accounts.bid = pgbench_branches.bid)
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts
                                        (cost=0.42..57.78 rows=1106 width=97)
                                  (actual time=0.011..1.334 rows=999 loops=1)
         Index Cond: (aid < 1000)
   ->  Hash  (cost=1.08..1.08 rows=8 width=364)
             (actual time=0.020..0.020 rows=8 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 1kB
         ->  Seq Scan on pgbench_branches  (cost=0.00..1.08 rows=8 width=364)
                                        (actual time=0.002..0.010 rows=8 loops=1)
 Planning time: 0.624 ms
 Execution time: 4.080 ms
(9 rows)
```

EXPLAIN ANALYZE表示情報

- 実行時間（例： actual time=0.058..3.221）
  - 最初の行を取得するまでの実行時間とすべての行を取得するまでの実行時間（ミリ秒）
  - 下位の計画ノードにおける実行時間を含む
- 行数（例：rows=999）
  - 実際に取得された行数
- ループ回数（例：loops=1）
  - 繰り返して実行された回数
  - 実行時間と行数にはループ回数を乗じる
- 計画作成時間（例：Planning time: 0.624 ms）
- 実行時間（例：Execution time: 4.080 ms）

～PostgreSQLパラメータチューニング編～
共有バッファの設定

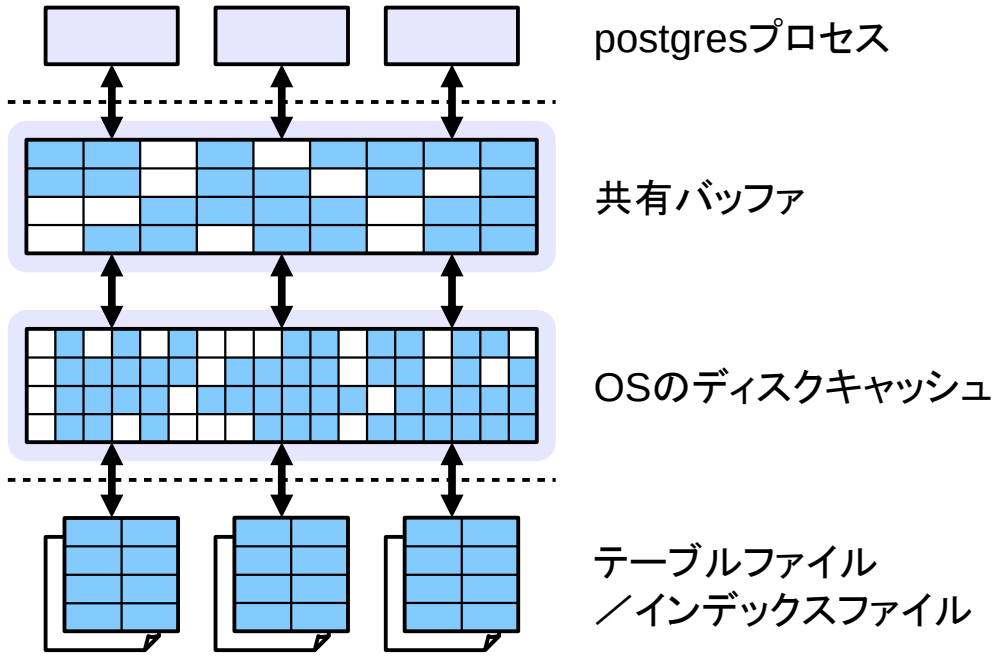
共有バッファの設定

- 共有バッファ（データの読み取り時に使用されるバッファ）を増やして読み取り性能を向上させる
  - デフォルト値（128MB以下）は少なすぎる
  - 物理メモリが1GB以上の場合は共有バッファに4分の1を割り当てる

```
shared_buffers = 256MB
```

共有バッファのメモリ容量（要再起動）

- 共有バッファを大きくしすぎるとメモリ領域を圧迫して逆に性能が低下
  - Linuxでは余ったメモリ領域をディスクI/Oのバッファキャッシュとして使用してくれる



共有バッファのページ割り当て

pg_stat_bgwriterビュー

- バックグランドライタと共有メモリに関する情報
- すべてのデータベースに共通

buffers_backendが
buffers_allocよりも大きいケースでは
shared_buffers不足の傾向

```
=# SELECT * FROM pg_stat_bgwriter;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 384
checkpoints_req       | 14
checkpoint_write_time | 231274
checkpoint_sync_time  | 3067
buffers_checkpoint    | 14572
buffers_clean         | 0
maxwritten_clean      | 0
buffers_backend       | 383151
buffers_backend_fsync | 0
buffers_alloc         | 14098
stats_reset           | 2018-10-09 12:25:17.944034+09

=# SELECT pg_stat_reset_shared('bgwriter'); -- リセット
```

データベース単位のキャッシュヒット

SRA OSS, INC. OSS-DB

pg_stat_database統計情報ビュー

```
=# SELECT * FROM pg_stat_database;
-[ RECORD 4 ]--+------------------------------
datid          | 16384    データベースのOID
datname        | test     データベース名
numbackends    | 1        データベースに接続中のプロセス数
xact_commit    | 4001     コミットされたトランザクション数
xact_rollback  | 0        ロールバックされたトランザクション数
blks_read      | 49959    ディスクから読み取られたブロック数
blks_hit       | 453838   バッファキャッシュ内に存在したブロック数
```

$$\text{キャッシュヒット率} = \frac{\text{blks\_hit}}{\text{blks\_read} + \text{blks\_hit}}$$

テーブル単位のキャッシュヒット

pg_stat_user_tables統計情報ビュー

```
=# SELECT * FROM pg_statio_user_tables;
-[ RECORD 4 ]---+-----------------
relid           | 16387    テーブルのOID
schemaname      | public   スキーマ名
relname         | t1       テーブル名
heap_blks_read  | 838      テーブルに関する
heap_blks_hit   | 104169     ディスク読み取り数、キャッシュヒット数
idx_blks_read   | 276      インデックスに関する
idx_blks_hit    | 200145     ディスク読み取り数、キャッシュヒット数
toast_blks_read | 0        TOASTテーブルに関する
toast_blks_hit  | 0          ディスク読み取り数、キャッシュヒット数
tidx_blks_read  | 0        TOASTテーブルのインデックスに関する
tidx_blks_hit   | 0          ディスク読み取り数、キャッシュヒット数
```

SQL単位のキャッシュヒット

EXPLAIN (ANALYZE, BUFFERS) SQL

```
=# EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM pgbench_accounts AS a JOIN
                                            pgbench_accounts AS b USING (aid);

                            QUERY PLAN
------------------------------------------------------------------------------
 Merge Join  (cost=0.86..319142.86 rows=3200000 width=190)
             (actual time=0.028..8779.856 rows=3200000 loops=1)
   Merge Cond: (a.aid = b.aid)
   Buffers: shared hit=61210 read=61202 written=2
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts a
               (cost=0.43..135571.43 rows=3200000 width=97)
               (actual time=0.009..1719.482 rows=3200000 loops=1)
         Buffers: shared hit=61206
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts b
               (cost=0.43..135571.43 rows=3200000 width=97)
               (actual time=0.006..1917.343 rows=3200000 loops=1)
         Buffers: shared hit=4 read=61202 written=2
 Planning time: 3.498 ms
 Execution time: 10252.869 ms
(9 rows)
```

shared hit =ヒットしたページ数
read＝読み込んだブロック数
written=書き込んだブロック数

ワークメモリ・メンテナンスワークメモリの設定

ワークメモリの設定

- ワークメモリ（ソートやハッシュ作成時などに使用されるメモリ）を増やしてSQLの実行時間を短くする
  - ワークメモリが少ないと、ソート時に一時ファイルの作成が必要になったり、ハッシュ作成に十分なメモリを確保できない
  - 共有メモリとは別にバックエンドごとに確保される
    - work_mem×max_connections（最大接続数）

```
#work_mem = 4MB
```

ワークメモリのメモリ容量

  - log_temp_filesを利用して調査
  - 大きなメモリを設定する場合には、セッション・トランザクションごとに設定

```
SET log_temp_files to 0;
SET work_mem TO '10MB';
SELECT …;
```

- 大きな値を設定すると物理メモリ不足になる可能性がある

一時ファイルの利用状況の確認

- work_memが十分でないとソート処理やハッシュ作成処理などで一時ファイルが利用される
- 大量の一時ファイルを使うSQLを調査
  - 指定サイズ以上の一時ファイルを利用するSQLを特定

```
log_temp_files = 1MB
```

0ですべて

- EXPLAIN ANALYZEでも確認できる
  - Sort Method: external merge Disk: 11912kB
  - Sort Method: quicksort Memory: 25kB
- 一時ファイルが多用されている場合にはwork_memのチューニングの余地あり

メンテナンスワークメモリの設定

- メンテナンスワークメモリを増やし、バキュームを効率的に実行して短時間に完了させる
- メンテナンスワークメモリ
  - VACUUMやREINDEX、CREATE INDEXなどの実行時に使用されるメモリの閾値
  - 共有メモリとは別に確保

```
#maintenance_work_mem = 64MB
```

メンテナンスワークメモリの最大メモリ容量

- VACUUMなどは一般的に同時に実行しないため、基本的に大きな値を設定しても問題ない
  - 自動バキュームでは、複数のプロセスがVACUUMを同時に実行するため（デフォルトでは3プロセス）、値を大きくしすぎないように注意

その他の設定

自動バキュームの活動調査

- 自動バキュームの活動状況をログに記述
- テーブル単位で自動バキュームの実行時間の特定

```
log_autovacuum_min_duration = 1s
```

0ですべての活動状況

```
LOG:  automatic vacuum of table "test.public.t1": index scans: 1
pages: 0 removed, 8334 remain
tuples: 500000 removed, 500000 remain, 0 are dead but not yet removable
buffer usage: 27810 hits, 4 misses, 15363 dirtied
avg read rate: 0.001 MB/s, avg write rate: 3.375 MB/s
system usage: CPU 0.08s/1.60u sec elapsed 35.56 sec
```

- 確認
  - 実行回数/1日 … 各テーブルがどれほどの頻度で実行されるか
  - 実行時間帯 … アクセスが多い時間帯に実行される頻度
  - 実行時間

WALバッファの設定

- WALバッファ（WALファイルの書き込み時に使用されるバッファ）を増やして書き込み性能を向上させる
  - デフォルトでは-1=shared_buffers÷32
    - shared_buffers = 512MBで16MB
  - 更新が多い場合には増やす（最大16MB）
  - サーバ起動時に共有メモリバッファ（shared_buffers）とは別に指定された容量のメモリが確保される

```
#wal_buffers = -1
```

WALバッファのメモリ容量（要再起動）

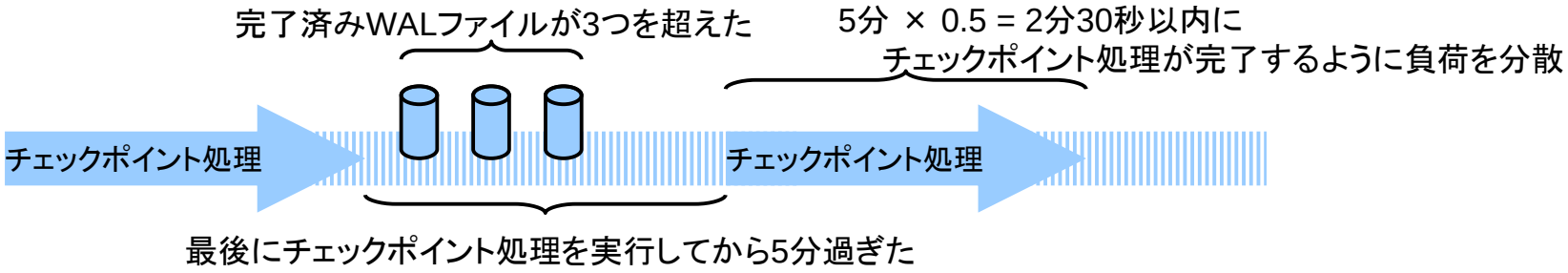
- WALバッファが少なすぎると、コミット時以外にも書き込みが発生してしまう

チェックポイントの設定

- チェックポイント処理の頻度を減らしてディスクI/Oの総量を減少させる
  - 更新が少ない場合にはcheckpoint_timeoutを長く、更新が多い場合にはcheckpoint_segmentsを多く設定
  - チェックポイント処理時の書き込み量が増えるため、一時的なディスクI/Oへの負荷は増加

```
#checkpoint_segments = 3
#checkpoint_timeout = 5min
#checkpoint_completion_target = 0.5
```

チェックポイント処理を実行するまでの完了済みWALファイル数
チェックポイント処理を実行するまでの間隔
チェックポイント処理を完了するまでの時間（間隔に対する割合）



チェックポイントの実行をログに記録し、vmstatなどで取った負荷状況と見比べる

```
log_checkpoints = on
```

- チェックポイントが頻発していないか
- チェックポイント時の負荷が高すぎないか

```
LOG:  checkpoint starting: xlog
LOG:  checkpoint complete: wrote 2 buffers (0.0%); 0 transaction
log file(s) added, 0 removed, 3 recycled; write=0.100 s, sync=0.007
s, total=0.135 s; sync files=1, longest=0.007 s, average=0.007 s
LOG:  checkpoint starting: time
LOG:  checkpoint complete: wrote 2107 buffers (12.9%); 0
transaction log file(s) added, 0 removed, 1 recycled; write=14.924
s, sync=0.116 s, total=15.062 s; sync files=35, longest=0.041 s,
average=0.003 s
```

プランナコスト定数の設定

■プラットフォームの特性に合わせてプランナコスト定数を調整し、より適切な実行計画が作成されるようにする

- プランナはプランナコスト定数をもとにコストを推定

```
#seq_page_cost = 1.0

#random_page_cost = 4.0
#effective_cache_size = 4GB
```

シーケンシャルアクセスで
ディスクから1ブロック読み取るコスト

ランダムアクセスでディスクから1ブロック読み取るコスト

プランナが想定するキャッシュサイズ
(実際にメモリは確保されない)

■インデックススキャンが選択されやすくするには

- random_page_costを減らしてeffective_cache_sizeを増やす
- effective_cache_sizeは共有メモリバッファの2倍（物理メモリの50%）くらいが適切

ご清聴ありがとうございました。

■お問い合わせ■

SRA OSS, Inc. 日本支社
OSS事業本部 マーケティング部
sales@sraoss.co.jp