>uptime

LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Gold 技術解説無料セミナー

2012/1/21

アップタイム・テクノロジーズ合同会社
共同創業者
永安 悟史

自己紹介

OSS-DB

氏名

- 永安 悟史（ながやす さとし）

略歴

- 2004/4-2007/9（3年6ヵ月）
  - 株式会社NTTデータ入社。
  - PostgreSQLによる並列分散RDBMSの研究開発。
  - SIプロジェクトの技術支援、並列分散PostgreSQLミドルウェアの製品サポートおよび保守。
- 2007/10-2008/9（1年）
  - データセンタ企画部門にて、次世代ITプラットフォームサービスの企画・開発。
- 2008/10-2009/10（1年1ヵ月）
  - データセンタ運用部門にて、OSS系システムの基盤保守・運用、および運用チームの統括。
  - 株式会社NTTデータ退職。
- 2009/11-
  - アップタイム・テクノロジーズ創業（共同創業者兼CEO）。

専門分野

- データベースシステム、並列分散システム、クラスタシステム
- オープンソース・インフラ技術
- ITサービスマネジメント（ITIL）、ITインフラ運用管理（運用設計～運用）

執筆等

- 翔泳社「PostgreSQL徹底入門 ~ 8対応」（共著）
- 技術評論社「PostgreSQL安定運用のコツ」（WEB+DB PRESS vol.32～37連載）、他

本セッションのねらい

OSS-DB

- PostgreSQL でシステムを構築して実運用をするためには、データベース管理者(DBA)として ある程度内部構造を理解しておく必要があります。

- 本講演では、開発や運用において必要とされる技術的知識について、 PostgreSQL の基本的な仕組みからバックアップ＆リカバリ、レプリケーションまで、 PostgreSQL の動作原理を俯瞰して解説を行います。

- 主に PostgreSQL中級者向けの内容です。

- 特に以下のような方にオススメです。
  - データベースの特に運用管理・パフォーマンス管理に詳しくなりたい方。
  - コンピュータアーキテクチャに詳しくなりたい方。
  - コンピュータエンジニアリングの基礎を知りたい方。
  - 他のRDBMSを利用していて、PostgreSQLについて知りたい方。
  - OSS-DB Goldの受験を検討している方、認定を取得したい方。

オープンソースデータベース技術者認定資格とは

OSS-DB

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技術者認定資格の試験体系

OSS-DB

OSS-DB Exam /Silver

＜出題範囲＞

- **一般知識（20%）**
  - オープンソースデータベースの一般的特徴
  - ライセンス
  - コミュニティと情報収集
  - リレーショナルデータベースの一般的知識
- **運用管理（50%）**
  - インストール方法
  - 標準付属ツールの使い方
  - 設定ファイル
  - バックアップ方法
  - 基本的な運用管理作業
- **開発/SQL（30%）**
  - SQLコマンド
  - 組み込み関数
  - トランザクションの概念

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
  - 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%）**
  - 性能に関係するパラメータ
  - 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

※　試験問題の向上の為にお客様に通知することなく試験内容・出題範囲等を 変更することがあります。

PostgreSQLアーキテクチャ入門 アジェンダ

OSS-DB

（1）アーキテクチャ概要
（2）クエリの処理
（3）I/O処理詳細
（4）領域の見積もり
（5）初期設定
（6）パフォーマンス管理
（7）データベースの監視
（8）バックアップ・リカバリ
（9）PITRによるバックアップ
（10）PITRによるリカバリ
（11）データベースのメンテナンス
（12）パフォーマンスチューニング（GUC）
（13）冗長化

（1）アーキテクチャ概要

プロセス

OSS-DB

```
$ ps -aef | grep postgres
postgres 22169     1  0 23:37 ?        00:00:00 /usr/pgsql-9.0/bin/postmaster -p 5432 -D
  /var/lib/pgsql/9.0/data
postgres 22179 22169  0 23:37 ?        00:00:00 postgres: logger process
postgres 22182 22169  0 23:37 ?        00:00:00 postgres: writer process
postgres 22183 22169  0 23:37 ?        00:00:00 postgres: wal writer process
postgres 22184 22169  0 23:37 ?        00:00:00 postgres: autovacuum launcher process
postgres 22185 22169  0 23:37 ?        00:00:00 postgres: archiver process   archiving
  000000010000000D60000004E
postgres 22187 22169  0 23:37 ?        00:00:00 postgres: stats collector process
postgres 23436 22169 16 23:42 ?        00:00:34 postgres: postgres pgbench [local] UPDATE waiting
postgres 23437 22169 16 23:42 ?        00:00:34 postgres: postgres pgbench [local] UPDATE waiting
postgres 23438 22169 16 23:42 ?        00:00:34 postgres: postgres pgbench [local] COMMIT
postgres 24283 22169  5 23:45 ?        00:00:02 postgres: postgres postgres [local] idle
postgres 24301 22169  0 23:45 ?        00:00:00 postgres: postgres postgres [local] idle
postgres 24581 22169  0 23:45 ?        00:00:00 postgres: autovacuum worker process   pgbench
postgres 24527 22185  0 23:45 ?        00:00:00 cp pg_xlog/000000010000000D60000004E
  /var/lib/pgsql/9.0/backups/archlog/000000010000000D60000004E
$
```

データベースクラスタ

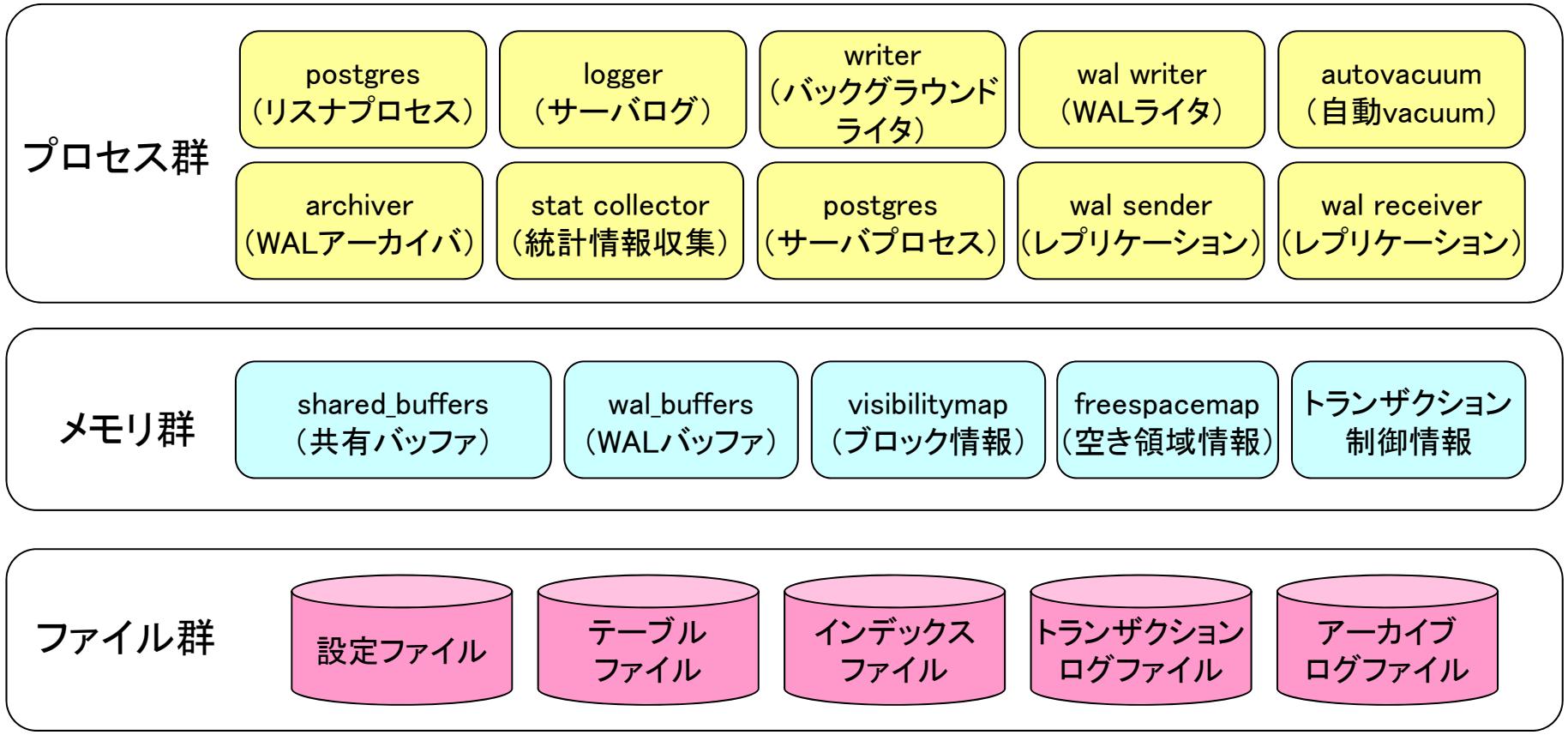
OSS-DB

```
# ls -l
total 116
drwx------ 10 postgres postgres  4096 Dec 14 19:00 base
drwx------  2 postgres postgres  4096 Jan 10 00:28 global
drwx------  2 postgres postgres  4096 Dec 13 08:40 pg_clog
-rw-------  1 postgres postgres  3768 Dec 14 15:50 pg_hba.conf
-rw-------  1 postgres postgres  1636 Dec  4 13:47 pg_ident.conf
drwx------  2 postgres postgres  4096 Jan 10 00:00 pg_log
drwx------  4 postgres postgres  4096 Dec  4 13:47 pg_multixact
drwx------  2 postgres postgres  4096 Jan  8 10:14 pg_notify
drwx------  2 postgres postgres  4096 Jan 10 15:43 pg_stat_tmp
drwx------  2 postgres postgres  4096 Dec 28 14:41 pg_subtrans
drwx------  2 postgres postgres  4096 Dec  4 14:47 pg_tblspc
drwx------  2 postgres postgres  4096 Dec  4 13:47 pg_twophase
-rw-------  1 postgres postgres     4 Dec  4 13:47 PG_VERSION
drwxr-xr-x  3 postgres postgres  4096 Jan 10 15:40 pg_xlog
-rw-------  1 postgres postgres 18015 Dec 14 15:50 postgresql.conf
-rw-------  1 postgres postgres 17952 Dec 14 15:05 postgresql.conf.orig
-rw-------  1 postgres postgres    71 Jan  8 10:14 postmaster.opts
-rw-------  1 postgres postgres    49 Jan  8 10:14 postmaster.pid
#
```

PostgreSQLの構成要素

OSS-DB

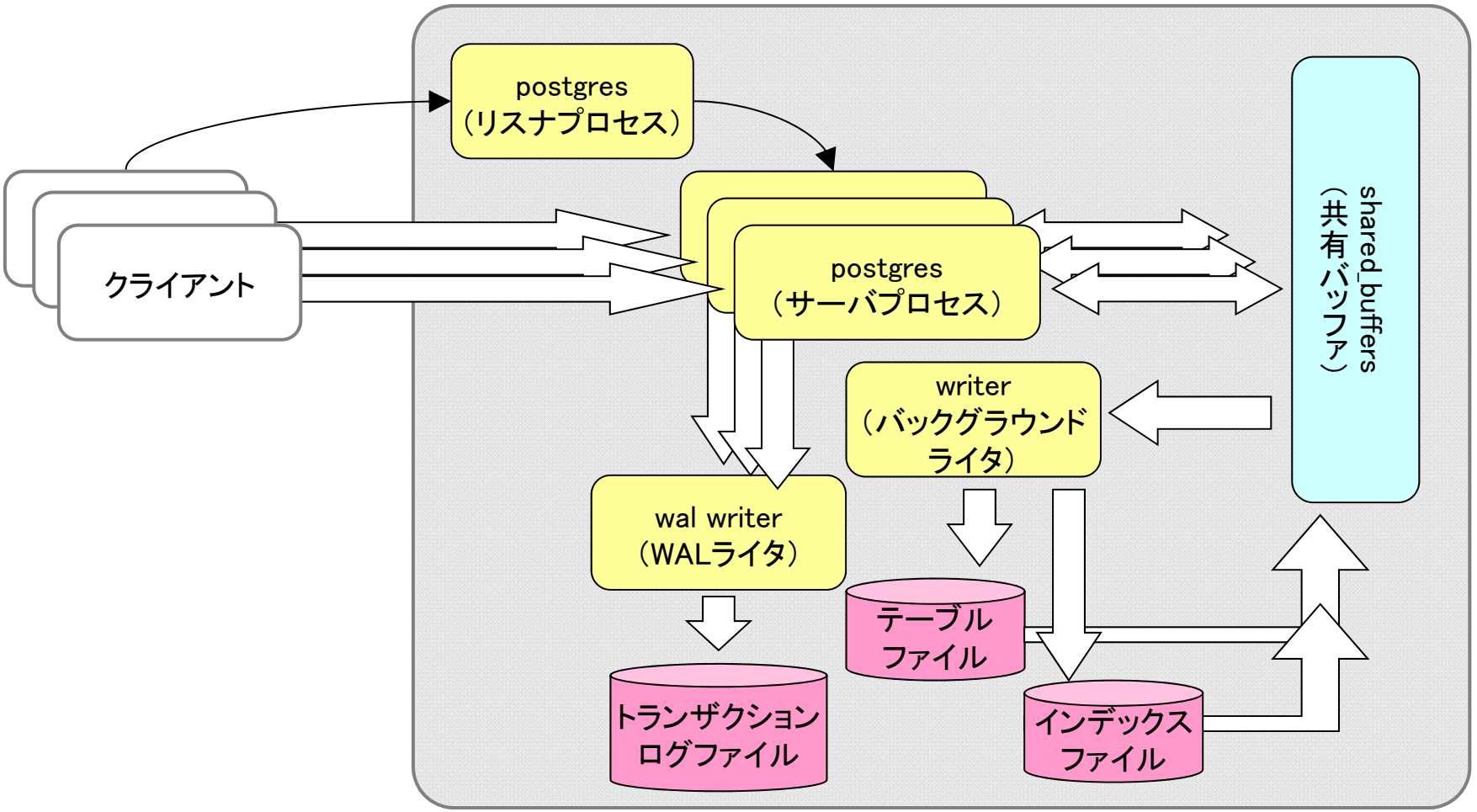
PostgreSQLは、さまざまなプロセス・メモリ領域・ファイルによって構成されている。



PostgreSQLの基本的なアーキテクチャ

OSS-DB

共有バッファを中心として、複数のプロセス間で連携しながら処理を行うマルチプロセス構造。



プロセス

OSS-DB

Postgres(Postmaster)プロセス

- PostgreSQLを起動すると最初に開始されるプロセス。
- クライアントからの接続を受け付け、認証処理を行う。
- 認証されたクライアントに対して、Postgresプロセスを生成(fork)して処理を引き渡す。

Postgresプロセス

- クライアントに対して1対1で存在する。
- クライアントからSQL文を受け付け、構文解析、最適化、実行、結果返却を行う。
- 共有バッファを介してデータを読み書きし、トランザクションログを書く。

Writerプロセス

- 共有バッファの内容をディスク(テーブルファイル、インデックスファイル)に非同期的に書き戻す。バックグラウンドライタ(bgwriter)とも呼ばれる。

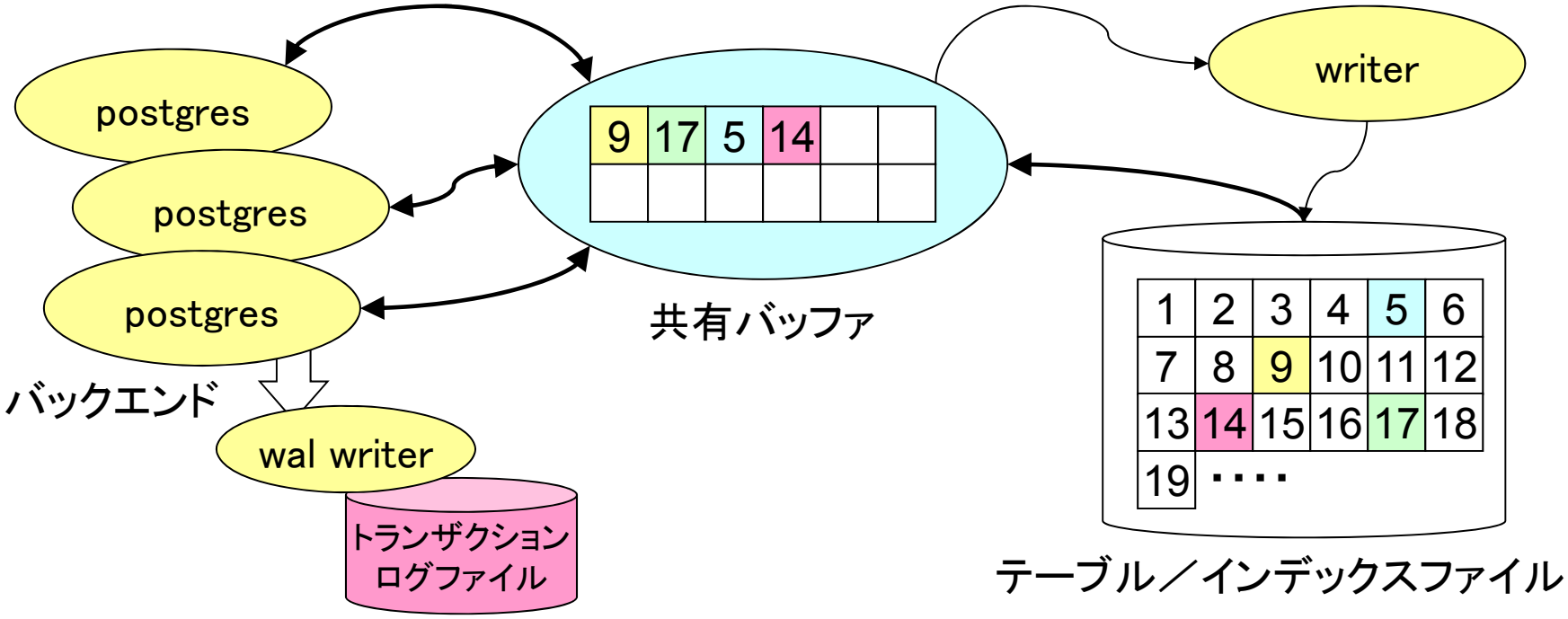
WAL Writerプロセス

- データベースに対する更新情報(WALレコード)をWALファイルに書き込む。

メモリ（共有バッファ）

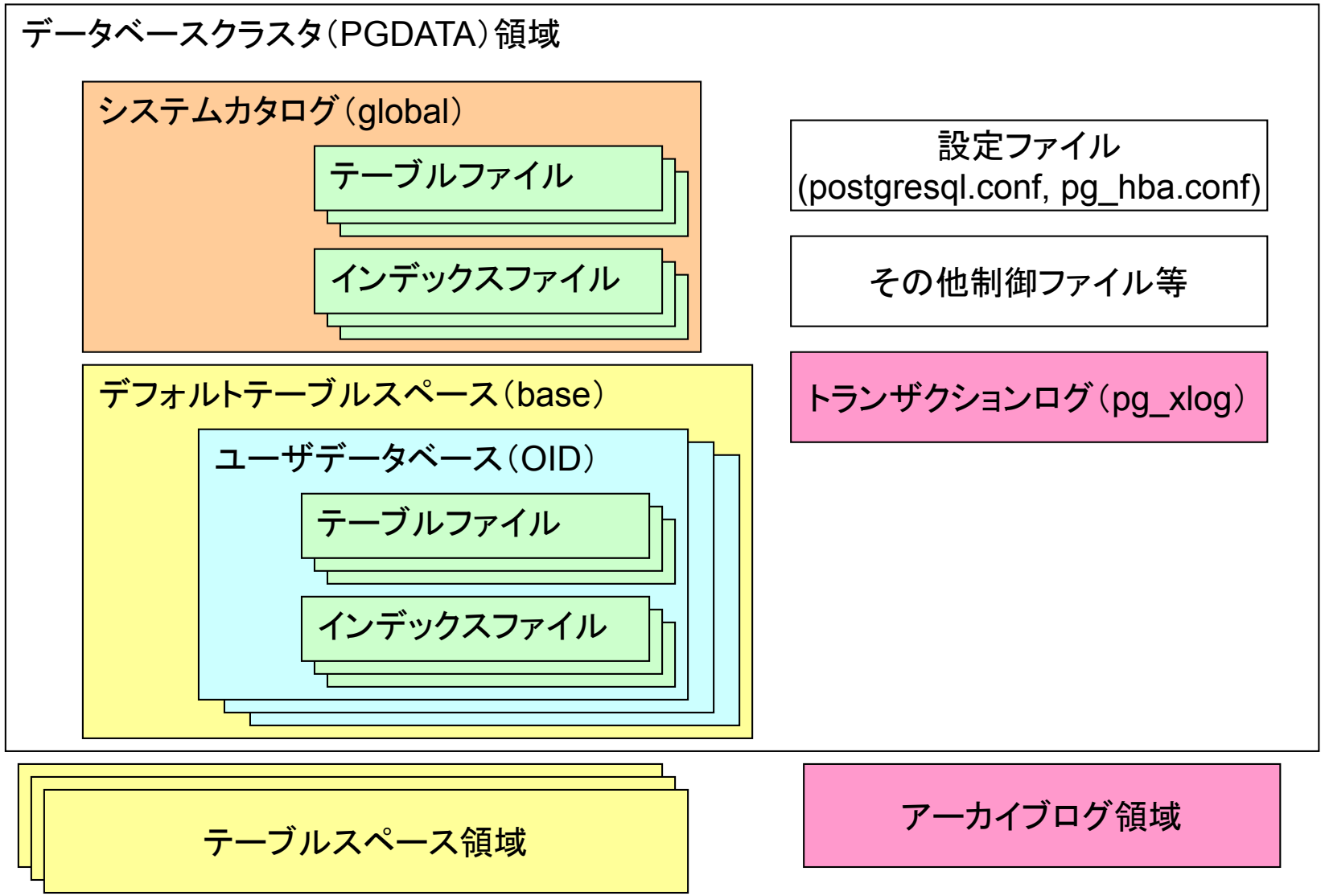
OSS-DB

- ディスク上のブロックをキャッシュするメモリ領域
  - ディスク上のブロックのうち、アクセスするものだけを読み込む
  - すべてのバックエンドプロセスで共有
- キャッシュすることで、ディスクI/Oを抑えて高速化
  - 更新の永続性はトランザクションログで担保する
  - メモリ上で変更されたブロックは、ライタプロセス（非同期）またはチェックポイント（同期）がテーブル／インデックスファイルに書き戻す



データファイルの配置

OSS-DB

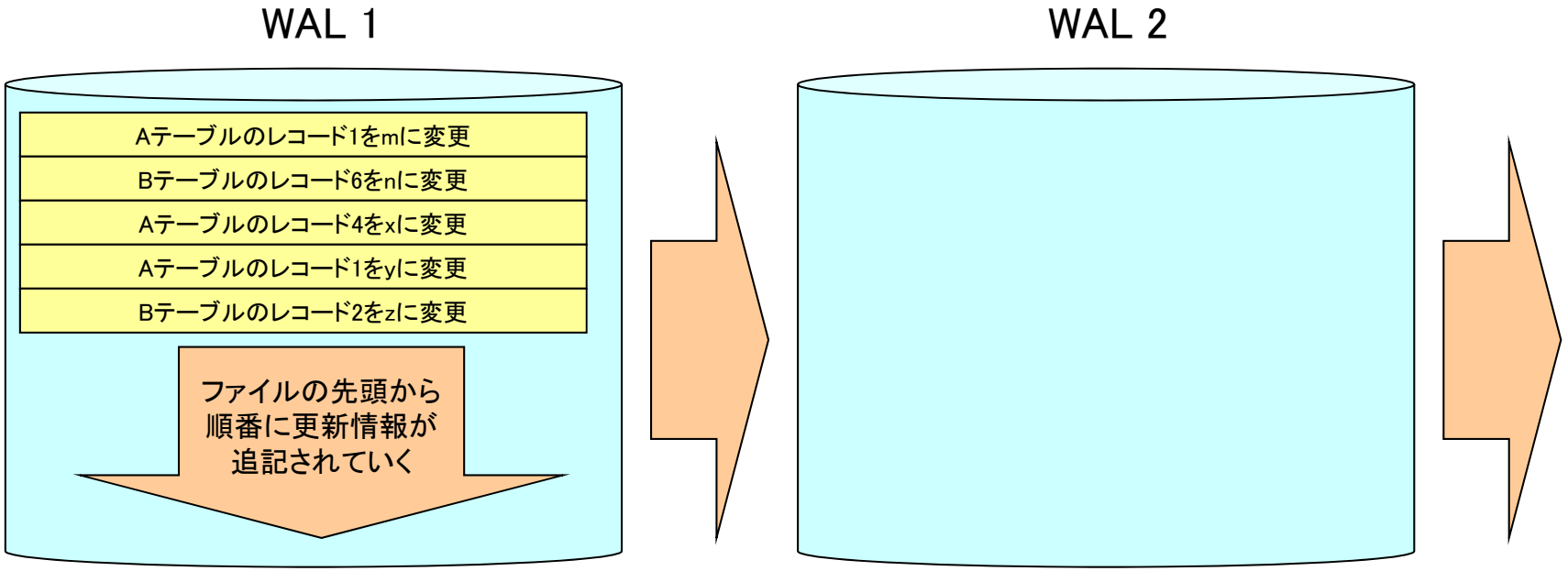


54.1. データベースファイルのレイアウト http://www.postgresql.jp/document/9.0/html/storage-file-layout.html

トランザクションログ(WAL)

OSS-DB

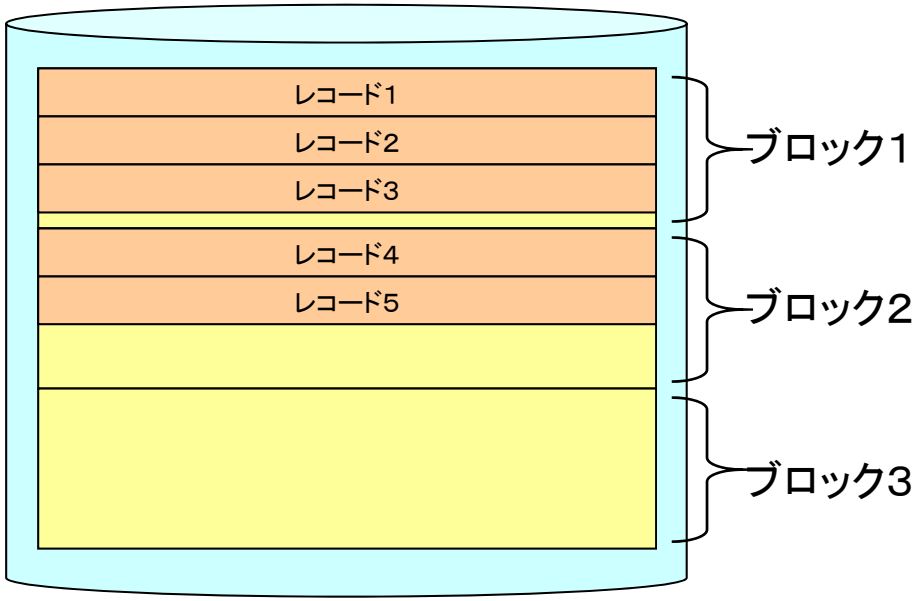
- テーブルやインデックスの更新情報が記録(追記)される
  - 共有バッファのデータを更新する「前」に記録(Write-ahead log)
  - 16MBずつのセグメント(ファイル)に分割されている。
  - クラッシュリカバリの際に読み込まれる(pg_xlog/ 以下に配置)
  - アーカイブされて、PITRのバックアップ/リカバリで使われる(アーカイブログ)



テーブルファイル

OSS-DB

- 8kB単位のブロック単位で構成される
- 各ブロックの中に実データのレコード(タプル)を配置
  - 基本的に追記のみ
  - 削除したら削除マークを付加する(VACUUMで回収)
  - レコード更新時は「削除＋追記」を行う。

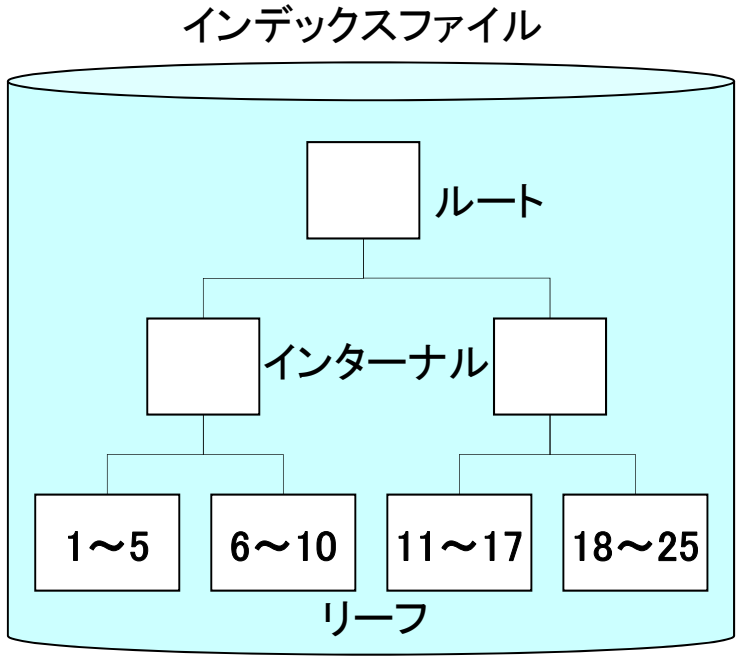



```
DBT1=# SELECT * FROM
pgstattuple('customer');
-[ RECORD 1 ]------+-----------
table_len          | 1754857472
tuple_count        | 3456656
tuple_len          | 1703225491
tuple_percent      | 97.06
dead_tuple_count   | 695
dead_tuple_len     | 350038
dead_tuple_percent | 0.02
free_space         | 31391624
free_percent       | 1.79

DBT1=#
```

インデックス(B-Tree)ファイル

OSS-DB

- 8kB単位のブロック単位で構成される
- ブロック(8kB単位)をノードとする論理的なツリー構造を持つ
  - ルート、インターナル、リーフの各ノードから構成
  - ルートノードから辿っていく
  - リーフノードは、インデックスのキーとレコードへのポインタを持つ




```
DBT1=# SELECT * FROM
pgstatindex('customer_pkey');
-[ RECORD 1 ]------+----------
version            | 2
tree_level         | 2
index_size         | 108953600
root_block_no      | 217
internal_pages     | 66
leaf_pages         | 13233
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 90.2
leaf_fragmentation | 0

DBT1=#
```

PostgreSQLデータファイルの特徴

OSS-DB

■データベースオブジェクト＝ファイル
- 1オブジェクト、1ファイル
- pg_classテーブルのrelfilenodeカラム
  - oid2nameコマンドでも確認可能
- 1GBを超えると1GB単位に分割される
  - XXXX.2, XXXX.3…と連番で作成

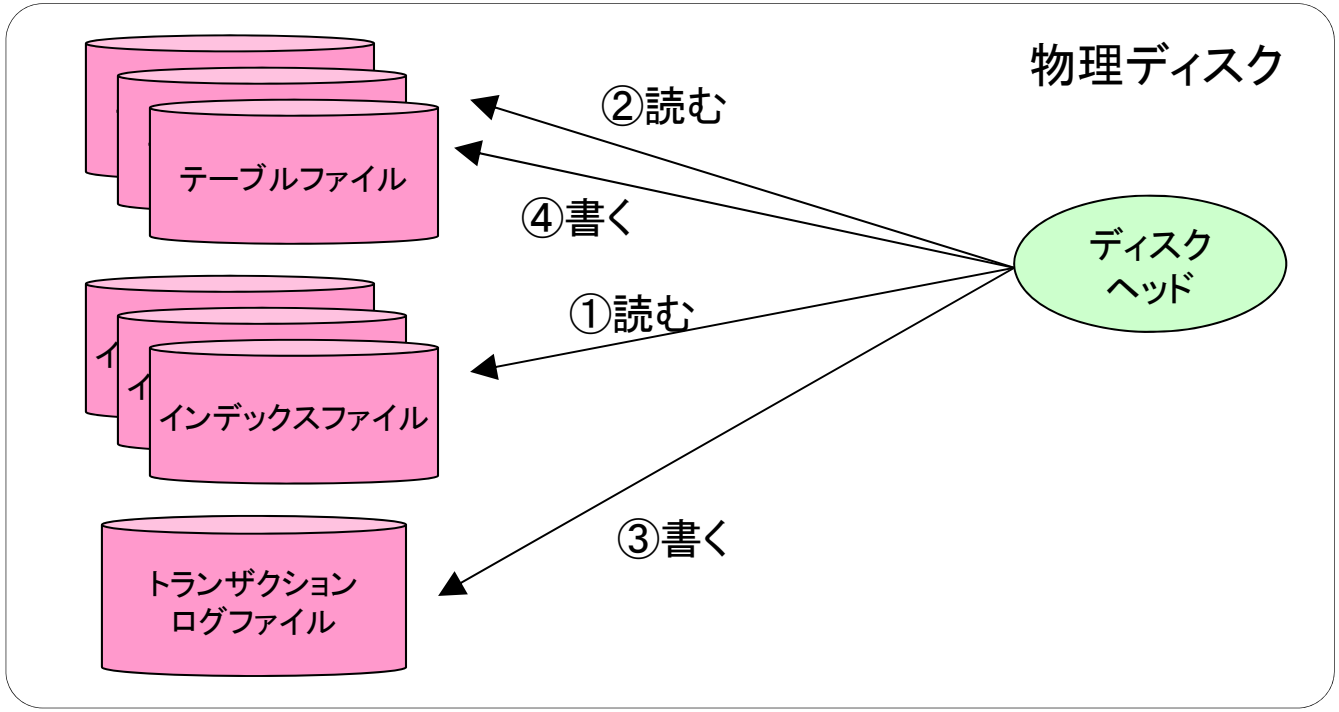
■データが追記されるとファイルが拡張（エクステント）される
- 8kBブロック単位で拡張
- あらかじめ指定したサイズのファイルを作っておくことはできない

発生する3種類のI/O

OSS-DB

■例えば、主キーで検索して該当レコードを更新する場合

- プライマリーキーでインデックスエントリを探す
- インデックスのポインタを元に、テーブル内のレコードを探す
- テーブルレコードを更新する前にトランザクションログに記録する
- テーブルファイルを更新する



（２）クエリの処理

SQL文の処理される流れ

OSS-DB

クエリ受信

構文解析(parse)

・SQL構文の解析、文法エラーの検出
・構文木(parse tree)の生成

書き換え(rewrite)

・VIEW / RULE に基づいた構文木の書き換え

実行計画生成 / 最適化
(plan / optimize)

・最適なクエリプラン(実行計画)の生成
・統計情報などを用いて実行コストを最小化
(コストベース最適化)

実行(execute)

・クエリプランに沿ったデータアクセス、抽出／結合／ソートなどの演算処理
・(更新時)トランザクションログ追記、共有バッファ更新

結果送信

クエリプラン(実行計画)とは

OSS-DB

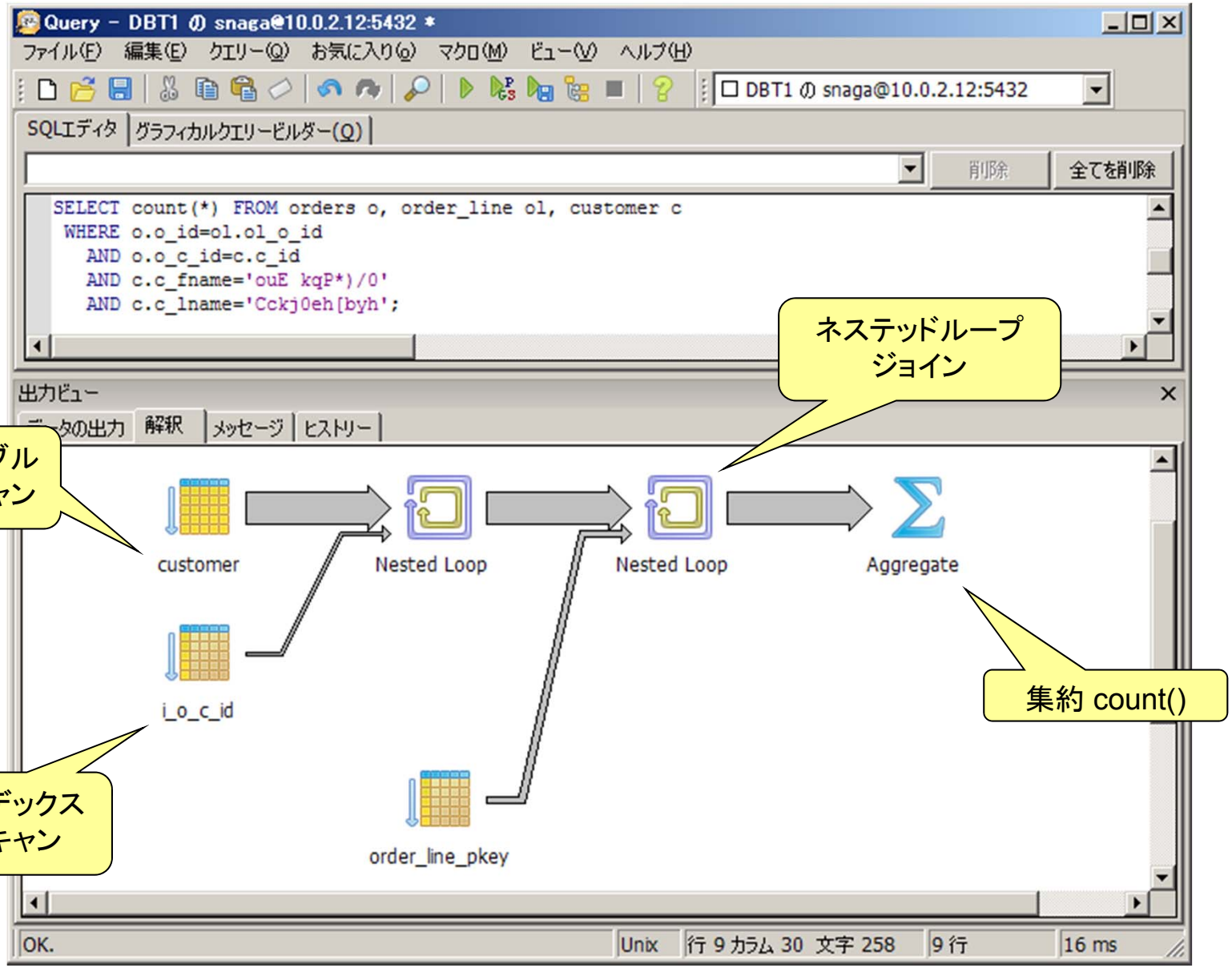
- どのテーブル、インデックスにどのようにアクセスするのか、という「アクセスパス(経路)」の情報

- テーブルやインデックスの統計情報を使って最適化される
  - よって、統計情報が正しいことが前提

- 商用RDBMSで実装されているヒント文はPostgreSQLには存在しない
  - DBAが手動で作るプランよりも、オプティマイザの生成するプランの方が賢い
  - ヒントを使わなければならないような状況なら、データベースやクエリの設計を見直すべき

クエリとクエリプラン

OSS-DB



クエリプランの詳細

OSS-DB

Query - DBT1 の snaga@10.0.2.12:5432 *

ファイル(F) 編集(E) クエリー(Q) お気に入り(o) マクロ(M) ビュー(V) ヘルプ(H)

DBT1 の snaga@10.0.2.12:5432

SQLエディタ | グラフィカルクエリービルダー(Q)

削除 | 全てを削除

```
SELECT count(*) FROM orders o, order_line ol, customer c
 WHERE o.o_id=ol.ol_o_id
   AND o.o_c_id=c.c_id
   AND c.c_fname='ouE kqP*)/0'
   AND c.c_lname='Cckj0eh[byh';
```

出力ビュー

データの出力 | 解釈 | メッセージ | ヒストリー

	QUERY PLAN text
1	Aggregate (cost=266120.26..266120.27 rows=1 width=0)
2	-> Nested Loop (cost=0.00..266120.24 rows=6 width=0)
3	-> Nested Loop (cost=0.00..266056.74 rows=2 width=8)
4	-> Seq Scan on customer c (cost=0.00..266043.96 rows=1 width=8)
5	Filter: (((c_fname)::text = 'ouE kqP*)/0'::text) AND ((c_lname)::text = 'Cckj0eh[byh'::text))
6	-> Index Scan using i_o_c_id on orders o (cost=0.00..12.76 rows=2 width=16)
7	Index Cond: (o.o_c_id = c.c_id)
8	-> Index Scan using order_line_pkey on order_line ol (cost=0.00..31.57 rows=14 width=8)
9	Index Cond: (ol.ol_o_id = o.o_id)

OK. | Unix | 行 9 カラム 30 文字 258 | 9 行 | 16 ms

クエリプランの確認方法

OSS-DB

EXPLAIN

- 最適であると判断された「クエリプラン」を表示。
- 入力されたSQL文を、PostgreSQLがどのように解釈して処理しようとしているのかを表示。

EXPLAIN（ ANALYZE ）

- 「クエリプラン」に加えて、「実行結果」を表示。
- 実際に、どのアクセスにどの程度の時間がかかっているのか、何件のレコードを処理したのか、などを表示。

EXPLAIN（ ANALYZE, BUFFERS ）

- クエリプラン、実行結果に加えて、「バッファアクセス」を表示。

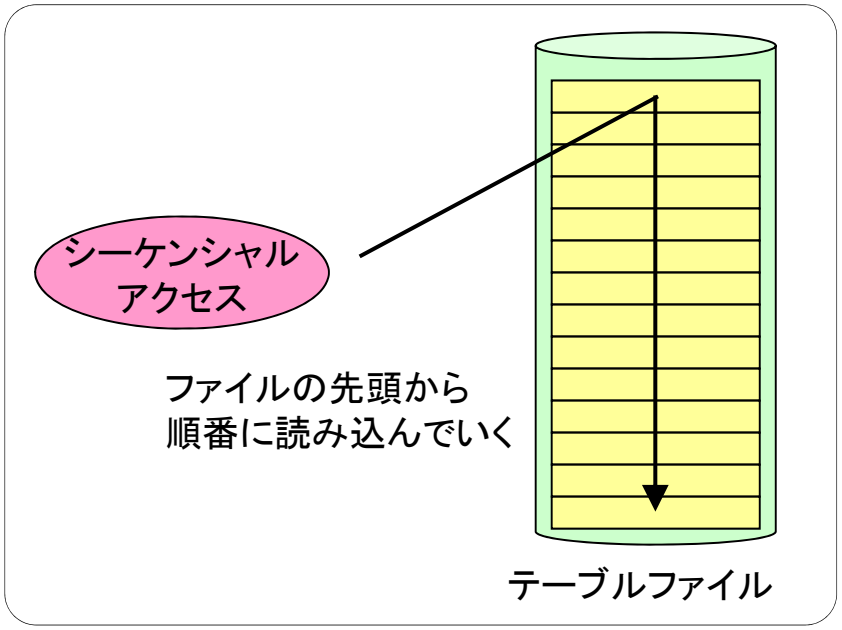
GUIツールで確認する方法（pgAdminIII）

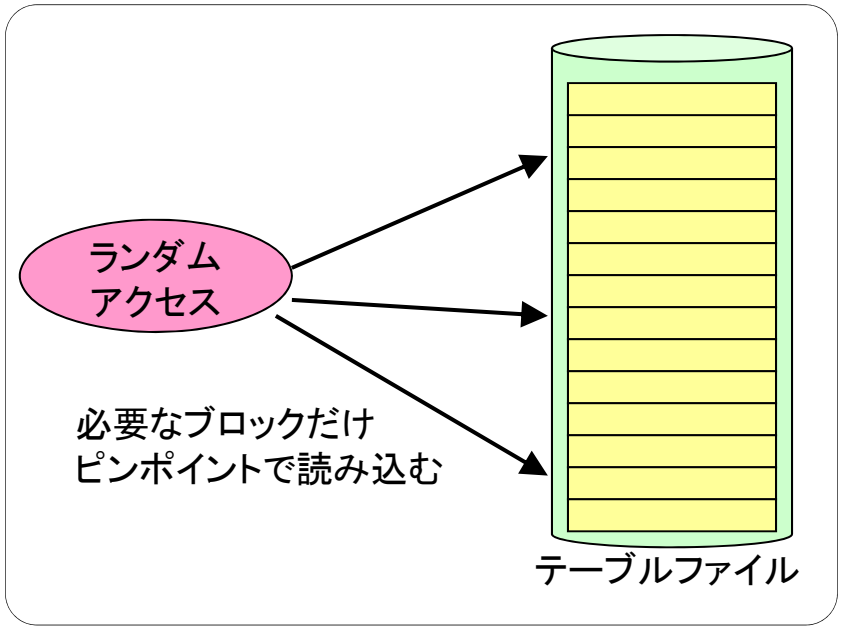
- 「クエリー解釈」＝EXPLAIN
- 「アナライズ解釈」＝EXPLAIN（ ANALYZE ）

データアクセスのパターン

OSS-DB

- シーケンシャルアクセス
  - 全レコード、または多くのレコードを処理する必要がある場合
  - 集約処理、LIKE文の中間一致など
- ランダムアクセス
  - 特定のレコード(を含むブロック)だけにアクセスする必要がある場合
  - 主にインデックスを用いたアクセス





OSS-DB

SELECT count(*) FROM customer;

```
snaga@devsv02:~/pgsql/postgres...
DBT1=# SELECT * FROM pg_statio_user_tables WH
ERE relname='customer'; SELECT * FROM pg_stat
io_user_indexes WHERE relname='customer';
-[ RECORD 1 ]---+---------
relid           | 18062
schemaname      | public
relname         | customer
heap_blks_read  | 0
heap_blks_hit   | 0
idx_blks_read   | 0
idx_blks_hit    | 0
toast_blks_read |
toast_blks_hit  |
tidx_blks_read  |
tidx_blks_hit   |

-[ RECORD 1 ]-+--------------
relid         | 18062
indexrelid    | 18065
schemaname    | public
relname       | customer
indexrelname  | customer_pkey
idx_blks_read | 0
idx_blks_hit  | 0
-[ RECORD 2 ]-+--------------
relid         | 18062
indexrelid    | 18119
schemaname    | public
relname       | customer
indexrelname  | i_c_uname
idx_blks_read | 0
idx_blks_hit  | 0

DBT1=#
```

```
snaga@devsv02:~/pgsql/postg
DBT1=# SELECT * FROM pg_statio_use
ERE relname='customer'; SELECT * F
io_user_indexes WHERE relname='cus
-[ RECORD 1 ]---+---------
relid           | 18062
schemaname      | public
relname         | customer
heap_blks_read  | 0
heap_blks_hit   | 214216
idx_blks_read   | 0
idx_blks_hit    | 0
toast_blks_read |
toast_blks_hit  |
tidx_blks_read  |
tidx_blks_hit   |

-[ RECORD 1 ]-+--------------
relid         | 18062
indexrelid    | 18065
schemaname    | public
relname       | customer
indexrelname  | customer_pkey
idx_blks_read | 0
idx_blks_hit  | 0
-[ RECORD 2 ]-+--------------
relid         | 18062
indexrelid    | 18119
schemaname    | public
relname       | customer
indexrelname  | i_c_uname
idx_blks_read | 0
idx_blks_hit  | 0

DBT1=#
```

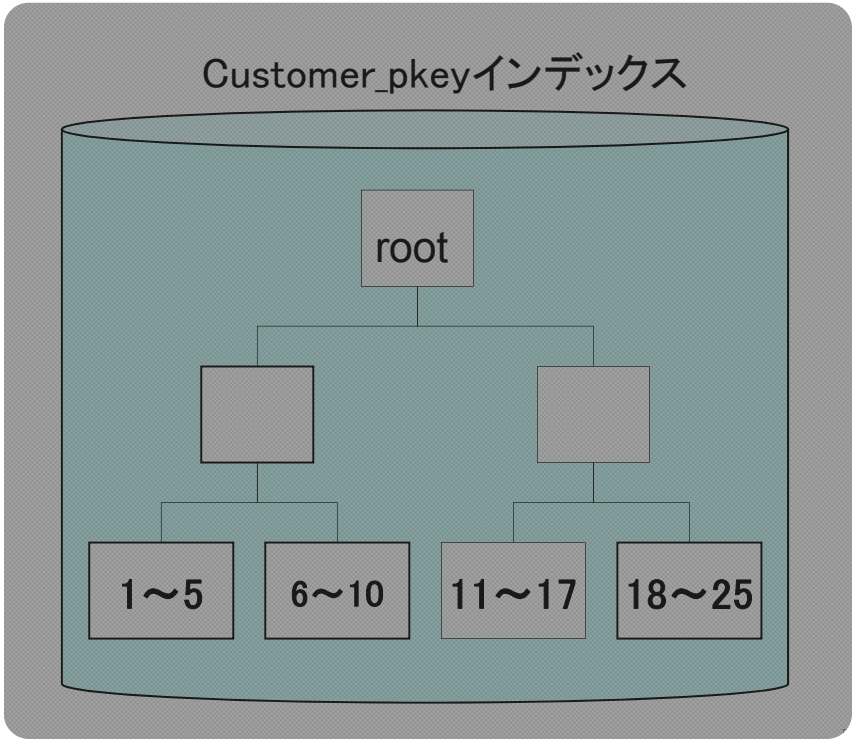
Customer テーブルからのブロック読込 ×214,216

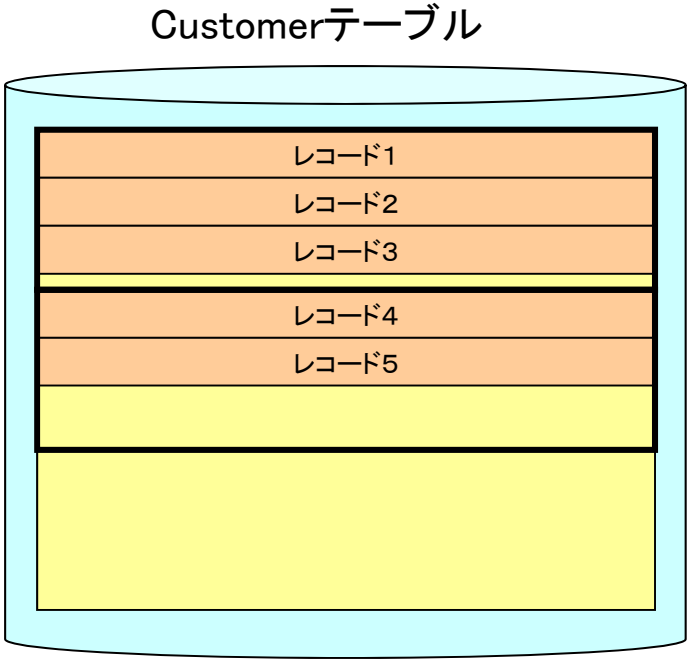
Customer_pkey インデックスのブロック読込 ×0

テーブルスキャン cont'd

OSS-DB

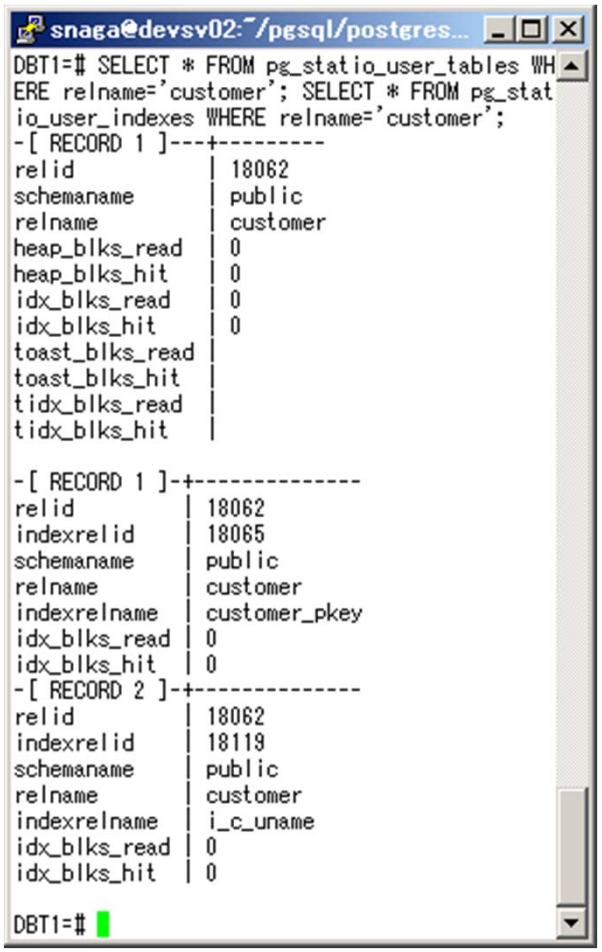
- すべてのデータを確認する必要があるため、customerテーブルファイルを構成するブロックを先頭から読み込む
  - よって、データが増えれば増えるほど時間がかかるようになる。
  - この例では、214,216 ブロック(約1.7GB)を読んでいる。

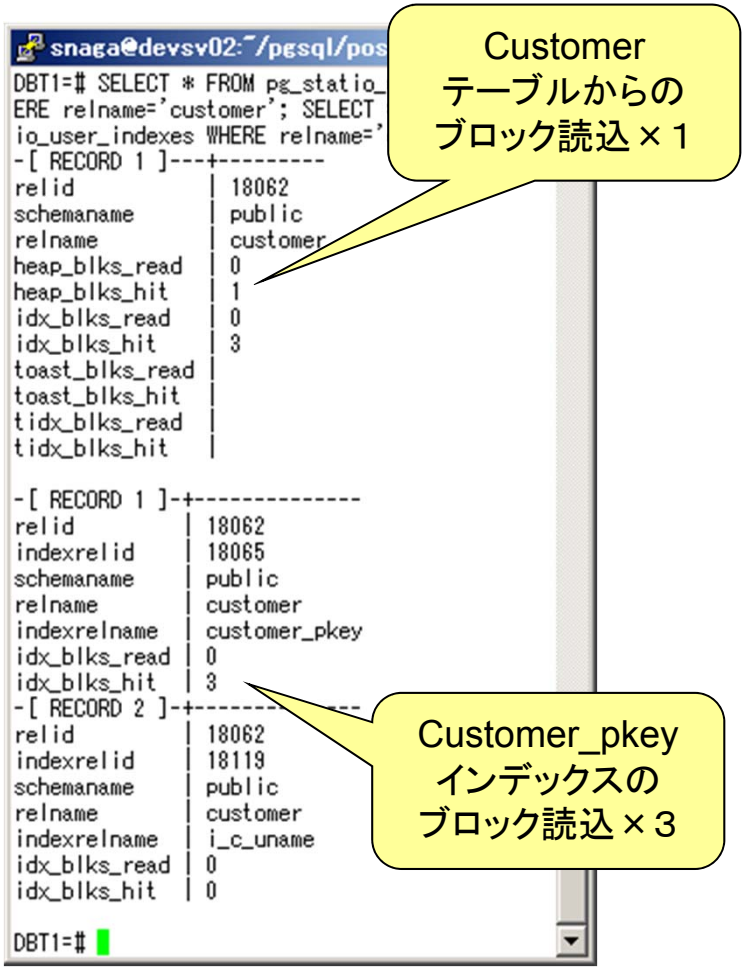




インデックスアクセス

SELECT * FROM customer c WHERE c.c_id=7;

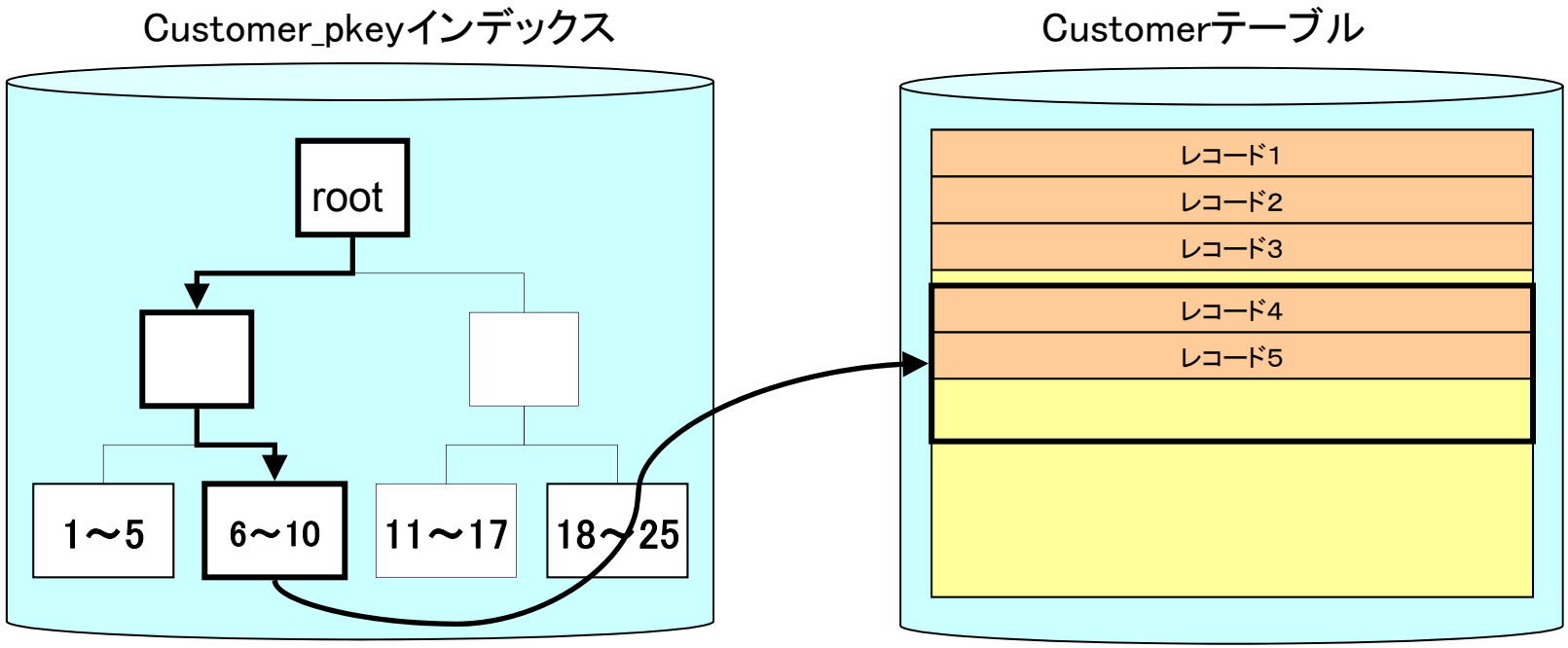




インデックスアクセス cont' d

OSS-DB

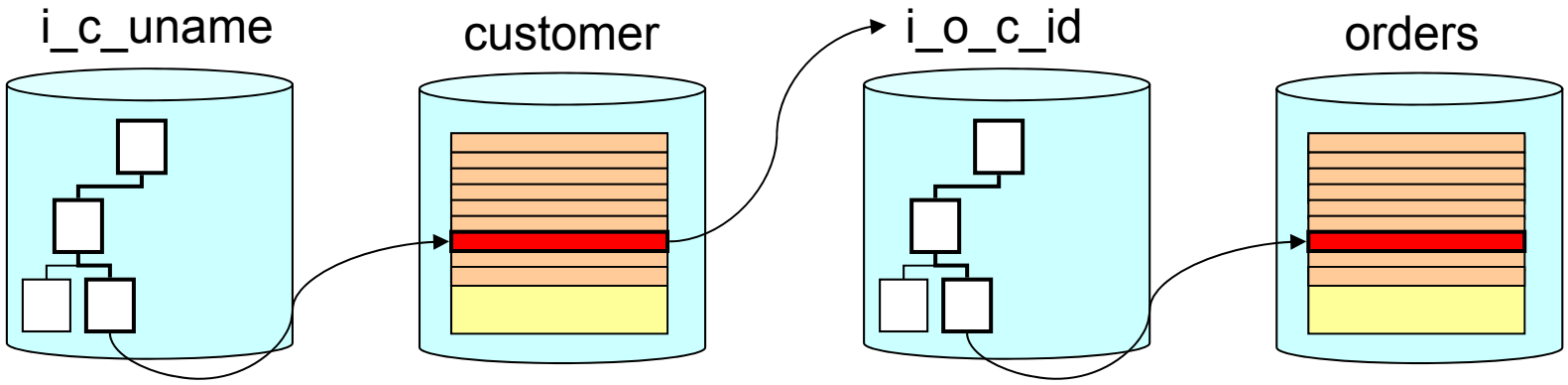
- “c_id=7” レコードの位置を探すため、customer_pkeyを辿ってポインタを見つけ、レコードを含むテーブルファイルのブロックを読み込む。
  - この例では、customer_pkeyインデックスから3ブロック、customerテーブルから1ブロックを読んでいる。
  - レコードの量とディスクアクセス量が比例しない。



結合(Nested Loop Join)

- SELECT count(*) FROM orders o, customer c WHERE o.o_c_id=c.c_id AND c.c_uname='UL';
  - customer を c_uname='UL' でインデックススキャン
  - customer のレコードの c_id を使って orders をインデックススキャン

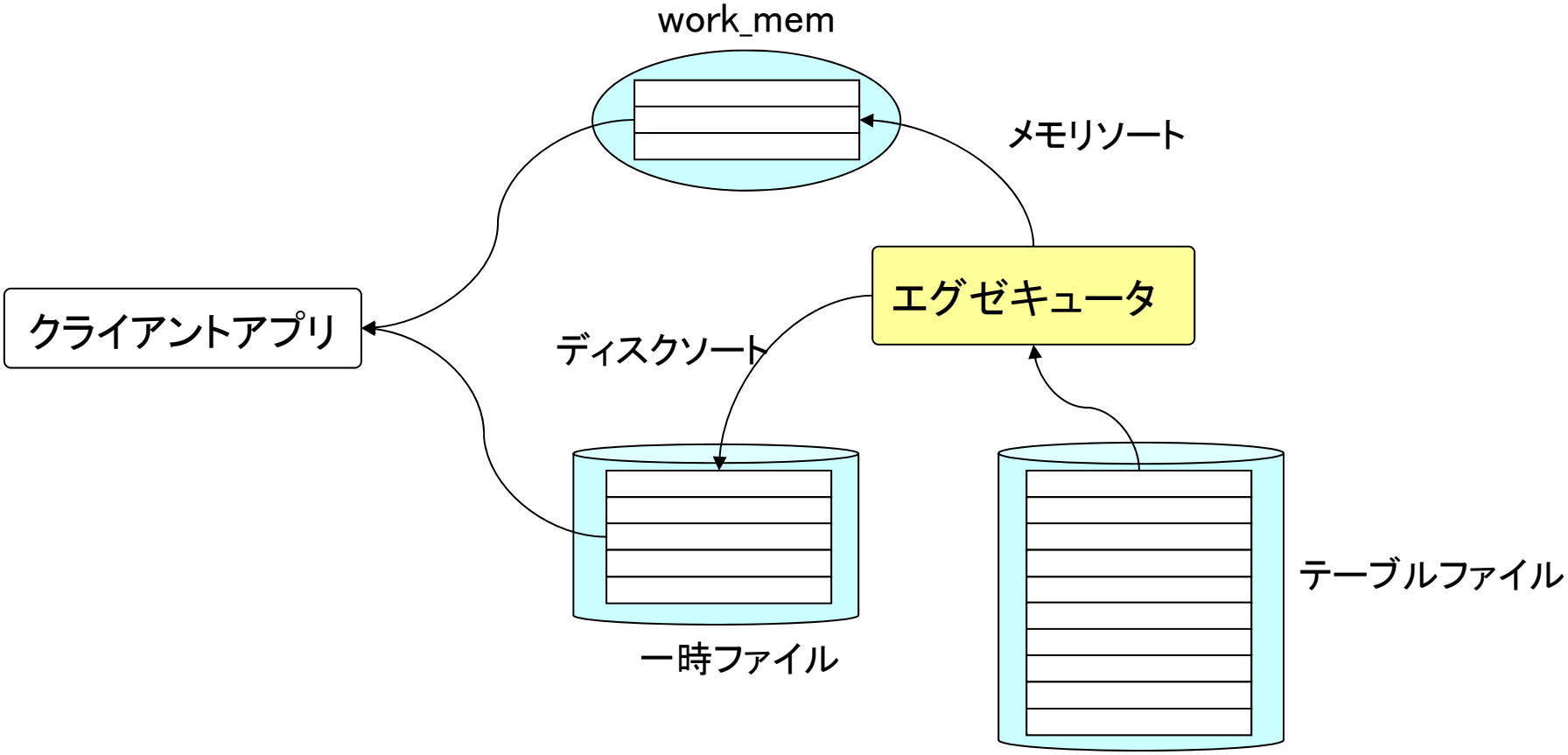
```
snaga@devsv02:~/pgsql/postgresql-9.0.1/contrib/pgstattuple
DBT1=# explain SELECT count(*) FROM orders o, customer c WHERE o.o_c_id=c.c_id AND c.c_uname='UL';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=21.60..21.61 rows=1 width=0)
   ->  Nested Loop  (cost=0.00..21.59 rows=2 width=0)
         ->  Index Scan using i_c_uname on customer c  (cost=0.00..8.80 rows=1 width=8)
               Index Cond: ((c_uname)::text = 'UL'::text)
         ->  Index Scan using i_o_c_id on orders o  (cost=0.00..12.76 rows=2 width=8)
               Index Cond: (o.o_c_id = c.c_id)
(6 rows)

DBT1=#
```



ソート（Sort）

必要なレコードを取り出して並べ替える処理

- 通常はメモリ中で実行するが、メモリが足りなくなるとディスクを使う
- メモリサイズの設定は work_mem パラメータ（デフォルト1MB）

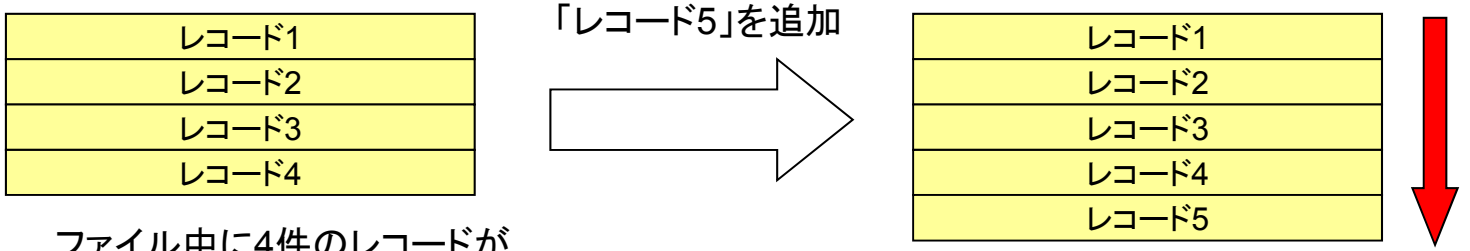


（3）I/O処理詳細

テーブルに対する更新処理

OSS-DB

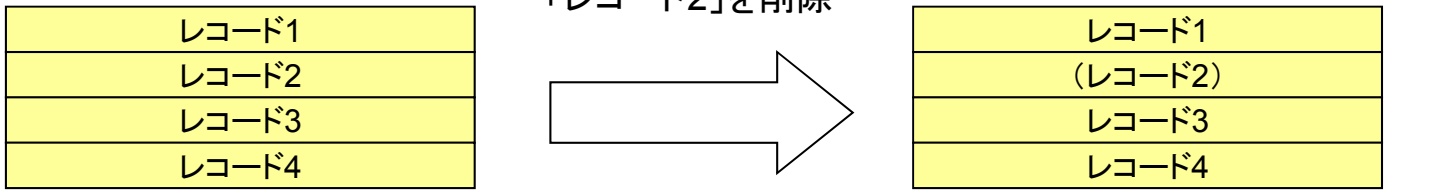
レコード追加処理（INSERT）



ファイル中に4件のレコードが順番に並んでいる

レコード5がファイル末尾に追加され、ファイルサイズが増える

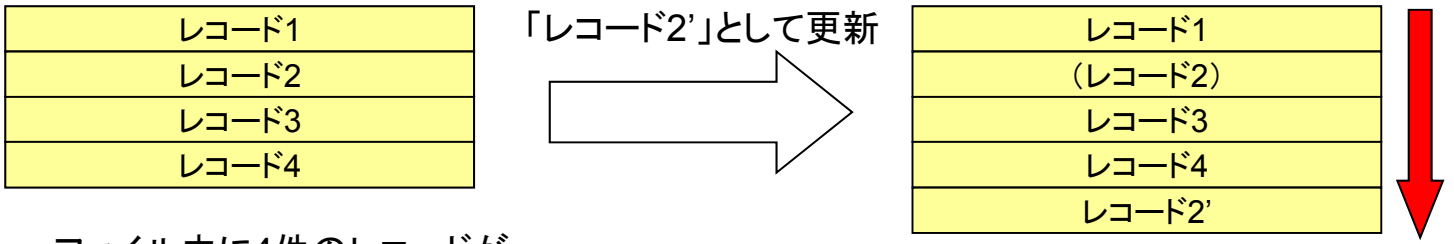
レコード削除処理（DELETE）



ファイル中に4件のレコードが順番に並んでいる

レコード2に削除マークが付けられる

レコード更新処理（UPDATE）



ファイル中に4件のレコードが順番に並んでいる

レコード2に削除マークが付けられ、
レコード2'が新たに追加、ファイルサイズ増加

テーブルに対する参照処理

OSS-DB

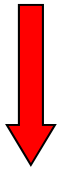
- 各タプル(テーブルのレコード)は、作成したトランザクション、または削除したトランザクションのXIDをヘッダに持つ。
- エグゼキュータは、作成・削除したトランザクションID(XID)を参照しながら、「読み飛ばすレコード」を決める。
- レコードを読んだり、読み飛ばしたりすることで、MVCCを実現する。

作成 XID	削除 XID	レコードデータ

作成XID ・・・ レコードを作成したトランザクションのID
削除XID ・・・ レコードを削除したトランザクションID

動作例(トランザクション分離レベルがRead Committedの場合)

101	-	レコードデータ1
101	103	レコードデータ2
103	-	レコードデータ3
103	-	レコードデータ4



トランザクション101　レコード1とレコード2を作成。コミット。
トランザクション102　トランザクション開始。
トランザクション103　レコード2を削除して、レコード3、レコード4を作成。コミット。
トランザクション102　レコード3、レコード4は参照可、レコード2は参照不可。

※MVCC:Multi-Version Concurrency Control

VACUUM処理

OSS-DB

VACUUM前

VACUUM処理

レコード1
（レコード2）
レコード3
レコード4
レコード2’

レコード2に削除マークが付いている

VACUUM処理

VACUUM後

レコード1
空き領域
レコード3
レコード4
レコード2’

レコード2の領域が「空き領域」として再利用可能になる。

追記前

VACUUMしてあると

レコード1
空き領域
レコード3
レコード4
レコード2’

「空き領域」がある

レコード5を追記

追記後

レコード1
レコード5
レコード3
レコード4
レコード2’

ファイルサイズを変えずに追記できる

VACUUMしてないと

レコード1
(レコード2)
レコード3
レコード4
レコード2’

レコード2の領域が埋まったまま

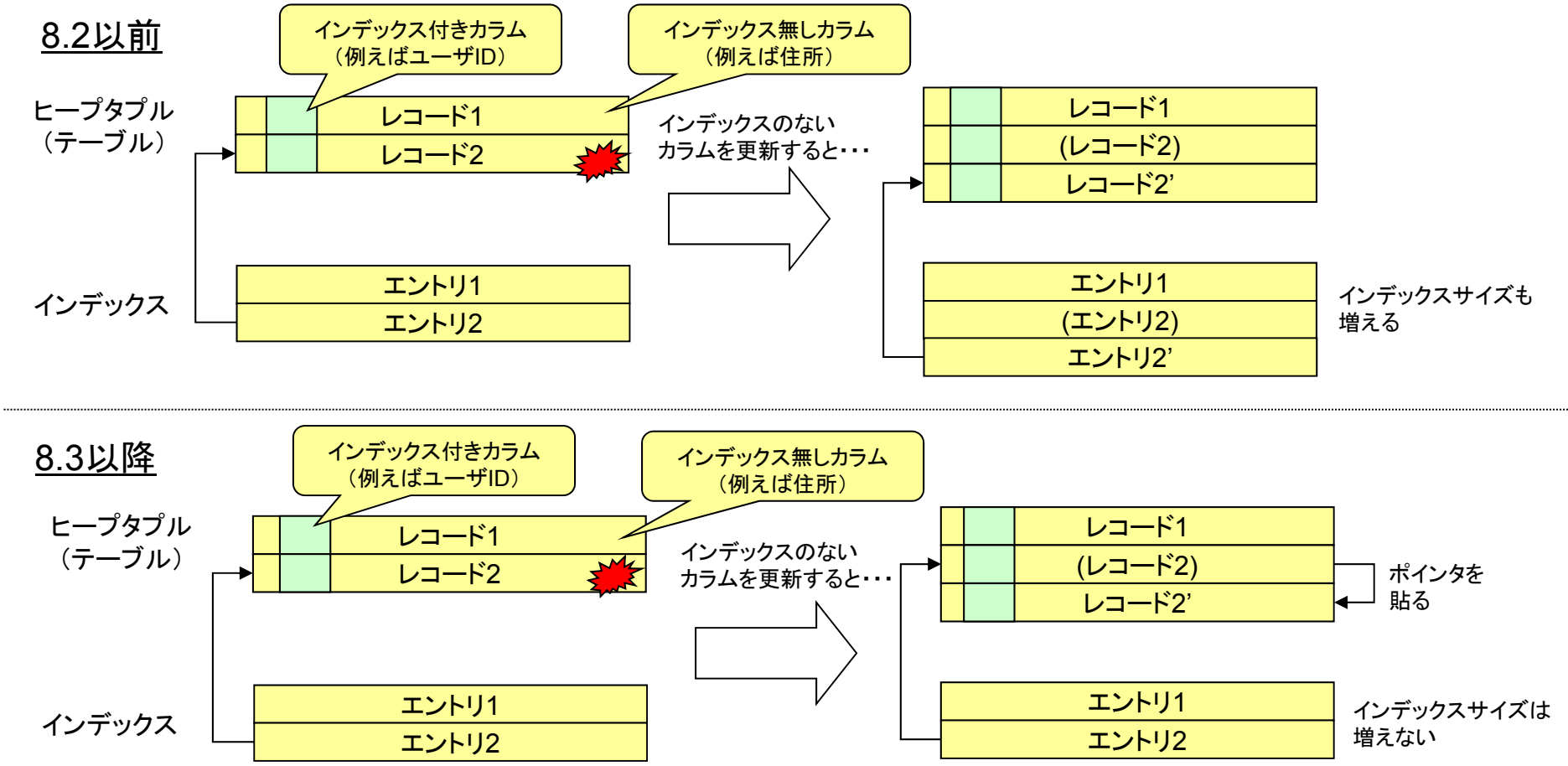
レコード5を追記

レコード1
(レコード2)
レコード3
レコード4
レコード2’
レコード5

ファイルサイズが増加

タプルの更新とインデックスの更新

OSS-DB



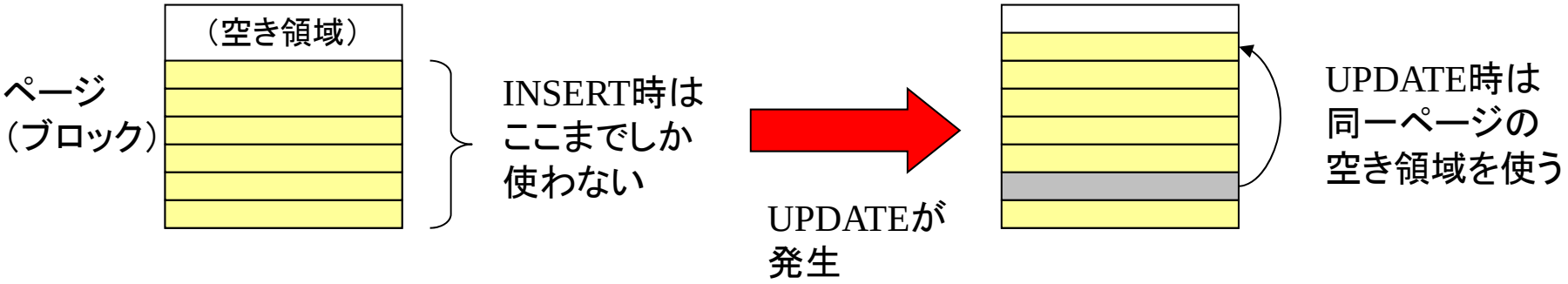
インデックスの張られていないカラムを更新すると、
ヒープのみの（インデックスエントリが無い）カラムができ、
インデックスの増加が抑制される。

これが、HOT（Heap Only Tuple）

FILLFACTORとは

OSS-DB

- データ追加時に予備領域を予約(確保)しておき、更新が発生した場合に、同一ページ(ブロック)を使う
  - これによって更新時にI/Oが削減される
  - 他のブロックを作成する、または読まずに済むため



- テーブルのFILLFACTOR
  - 10から100までのパーセンテージで指定。100(完全に詰め込む)がデフォルト。
  - まったく更新されないテーブルの場合は100でよい。
  - 更新が多いテーブルは90などを指定し、更新に備えて予備領域を確保しておく。

- インデックスのFILLFACTOR
  - インデックスページにどれだけ(データを)詰め込むか、10から100までの任意の値をパーセンテージで指定。
  - B-Treeインデックスでは、デフォルトは90。更新されないインデックスなら100を指定。

CREATE TABLE http://www.postgresql.jp/document/9.0/html/sql-createtable.html
CREATE INDEX http://www.postgresql.jp/document/9.0/html/sql-createindex.html

インデックスのFILLFACTOR

OSS-DB

FILLFACTOR 80
サイズ：2.0MB

```
pgbench=# ALTER INDEX accounts_pkey SET (
  fillfactor = 80 );
ALTER INDEX
pgbench=# REINDEX INDEX accounts_pkey;
REINDEX
pgbench=# SELECT * from
  pgstatindex('accounts_pkey');
-[ RECORD 1 ]------+--------
version            | 2
tree_level         | 1
index_size         | 2031616
root_block_no      | 3
internal_pages     | 0
leaf_pages         | 247
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 79.67
leaf_fragmentation | 0
```

FILLFACTOR 100
サイズ：1.6MB

```
pgbench=# ALTER INDEX accounts_pkey SET (
  fillfactor = 100 );
ALTER INDEX
pgbench=# REINDEX INDEX accounts_pkey;
REINDEX
pgbench=# SELECT * from
  pgstatindex('accounts_pkey');
-[ RECORD 1 ]------+--------
version            | 2
tree_level         | 1
index_size         | 1622016
root_block_no      | 3
internal_pages     | 0
leaf_pages         | 197
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 99.83
leaf_fragmentation | 0
```

（4）領域の見積もり

ユーザデータの見積もり

OSS-DB

テーブルファイル

- テーブルファイルサイズ＝8kB×ブロック数
- ブロック数＝総レコード数÷1ページ格納レコード数
- 1ブロック格納レコード数＝ブロック最大使用可能サイズ×平均充填率÷レコードサイズ

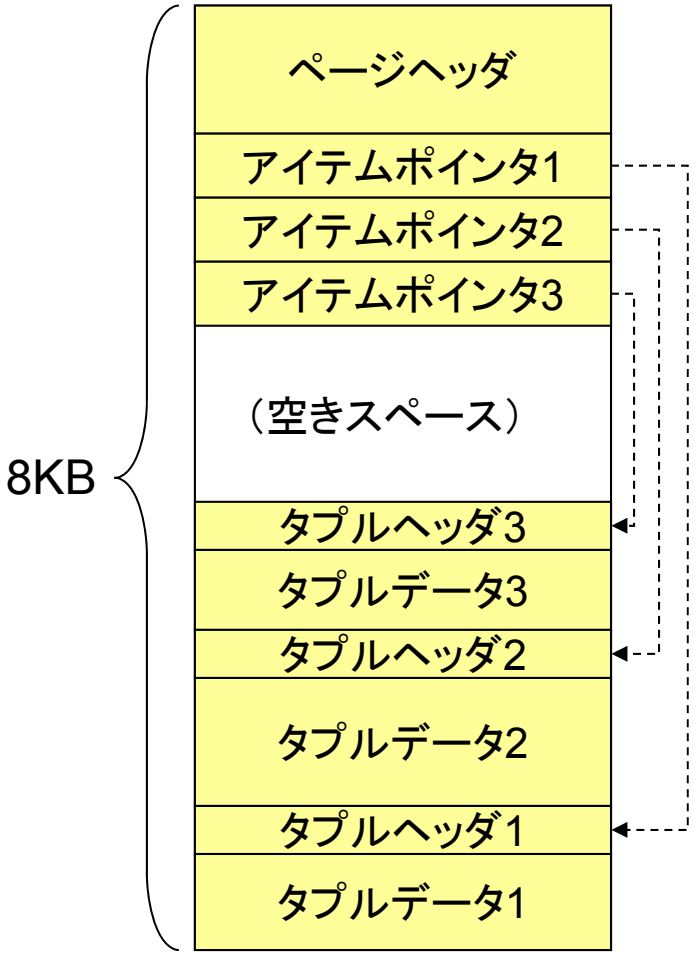
インデックスファイル

- インデックスファイルサイズ≒8kB×リーフノード数
- リーフノード数＝総レコード数÷1ページ格納エントリ数
- 1ブロック格納エントリ数＝ブロック最大利用可能サイズ×平均充填率÷インデックスタプルサイズ

テーブルのページレイアウト

OSS-DB

- テーブルファイルのページブロックは、ページヘッダ、アイテムポインタ、タプルヘッダ、およびタプルデータで構成される。



- ページヘッダを除くスペースを、アイテムポインタが前から、レコードデータが後ろから使う。
- アイテムポインタは、タプルヘッダの開始位置、および長さを保持する。
- タプルヘッダは、そのタプルを作成、削除したトランザクションのXIDを保持する(タプルの可視性の判断に使用)。

- ページヘッダ(PageHeaderData 28バイト)
- アイテムポインタ(ItemIdData 4バイト)
- タプルヘッダ(HeapTupleHeaderData 24バイト)
- タプルデータ(可変、データ型に依存)

データ型とデータサイズ

OSS-DB

型名	格納サイズ	説明	範囲
smallint	2バイト	狭範囲の整数	-32768から+32767
integer	4バイト	典型的に使用する整数	-2147483648から+2147483647
bigint	8バイト	広範囲整数	-9223372036854775808から9223372036854775807
decimal	可変長	ユーザ指定精度、正確	小数点前までは131072桁、小数点以降は16383桁
numeric	可変長	ユーザ指定精度、正確	小数点前までは131072桁、小数点以降は16383桁
real	4バイト	可変精度、不正確	6桁精度
double precision	8バイト	可変精度、不正確	15桁精度
serial	4バイト	自動増分整数	1から2147483647
bigserial	8バイト	広範囲自動増分整数	1から9223372036854775807

型名	格納サイズ	説明	最遠の過去	最遠の未来	精度
timestamp [(*p*)] [without time zone]	8 バイト	日付と時刻両方（時間帯なし）	4713 BC	294276 AD	1μ秒、14桁
timestamp [(*p*)] with time zone	8バイト	日付と時刻両方、時間帯付き	4713 BC	294276 AD	1μ秒、14桁
date	4バイト	日付（時刻なし）	4713 BC	5874897 AD	1日
time [(*p*)] [without time zone]	8バイト	時刻（日付なし）	00:00:00	24:00:00	1μ秒、14桁
time [(*p*)] with time zone	12バイト	その日の時刻のみ、時間帯付き	00:00:00+1459	24:00:00-1459	1μ秒、14桁
interval [*fields*] [(*p*)]	12バイト	時間間隔	-178000000年	178000000年	1μ秒、14桁

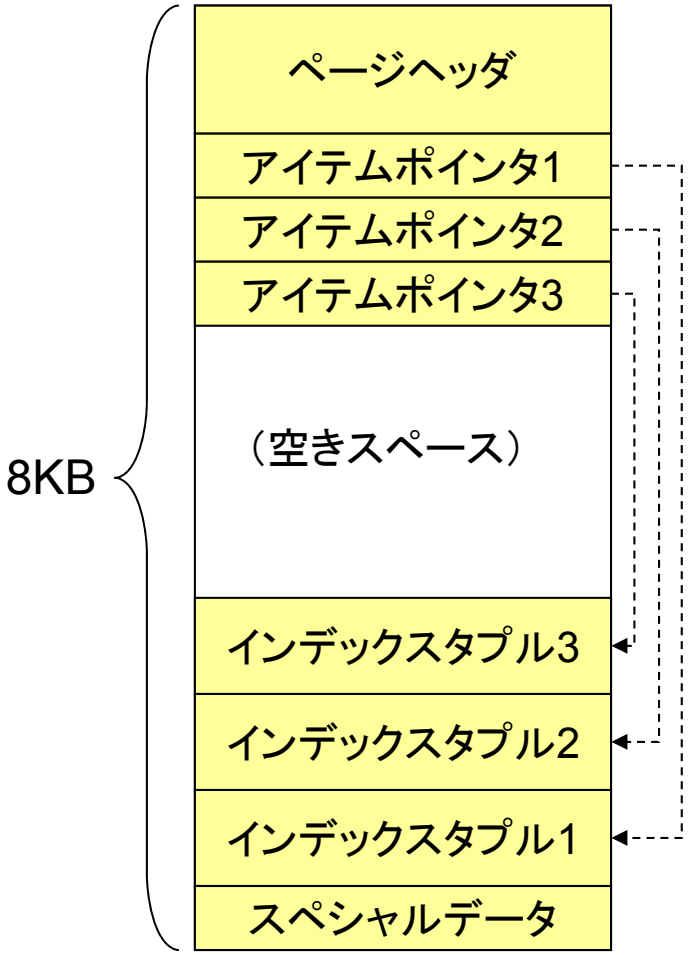
型名	説明
character varying(*n*)、varchar(*n*)	上限付き可変長
character(*n*)、char(*n*)	空白で埋められた固定長
text	制限なし可変長

第8章 データ型
http://www.postgresql.jp/document/9.0/html/datatype.html

B-Tree(リーフ)のページレイアウト

OSS-DB

- B-Treeインデックスのリーフページブロックは、ページヘッダ、アイテムポインタ、インデックスタプルで構成される。



- ページヘッダを除くスペースを、アイテムポインタが前から、インデックスタプルが後ろから使う。
- インデックスタプルは、テーブルファイル内における該当レコードの「ブロック番号」と「アイテム番号」、およびキーの値を保持する。
- スペシャルデータは、B-Treeにおける「隣のノードのブロック番号」や「ツリー中の深さ」を保持する(インデックススキャンで利用)。

- ページヘッダ(PageHeaderData 28バイト)
- インデックスタプル(IndexTupleData 8バイト+可変、キーサイズに依存)
- スペシャルデータ(BTPageOpaqueData 16バイト)

WALファイルの見積もり

OSS-DB

■トランザクションログ領域

- WALファイルは巡回的(cyclic)に使用されるため、最大容量が決まる

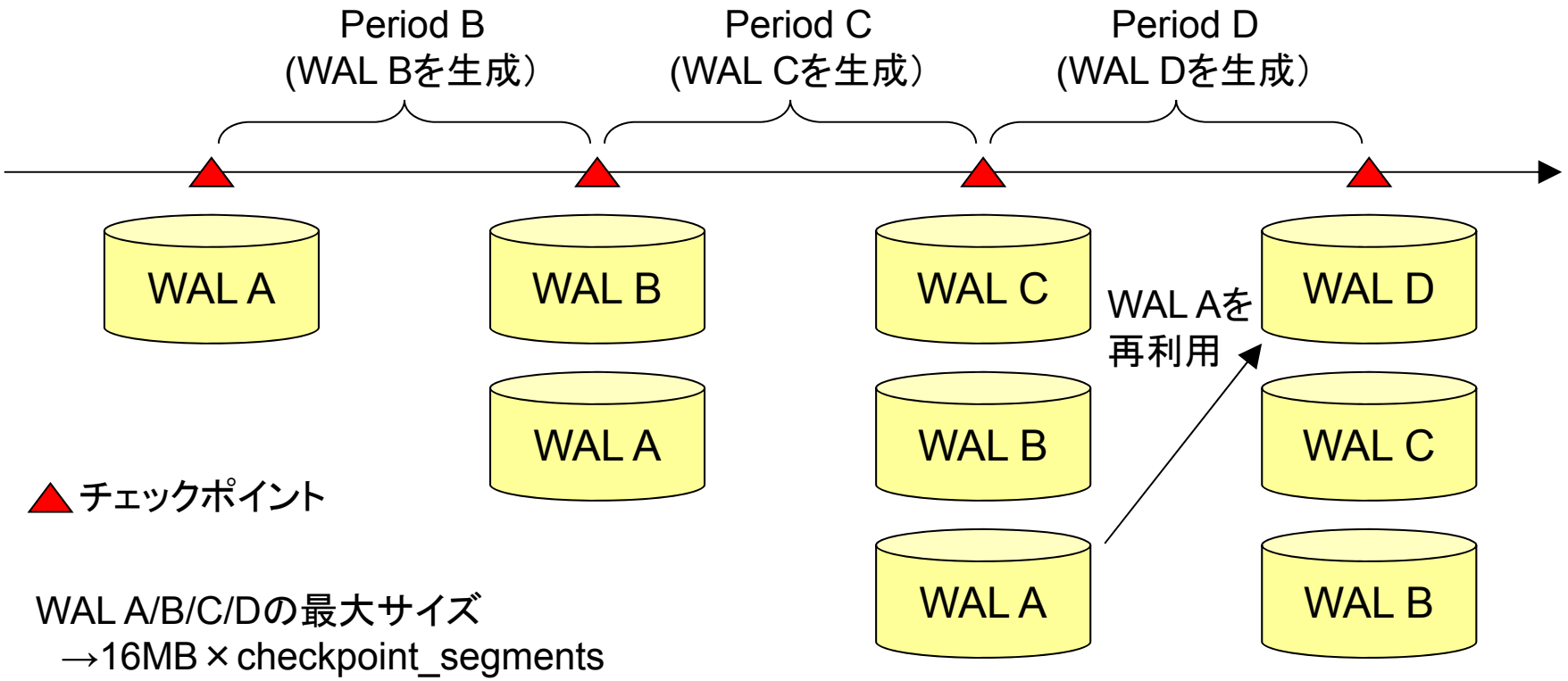
■アーカイブログ領域

- ベースバックアップ(非一貫性バックアップ)の間で生成され、アーカイブされるトランザクションログ
- 机上での見積もりは難しいので、実際にトランザクションを実行して見積もる
- 更新トランザクションの量とベースバックアップの頻度から算出。

トランザクションログ領域の見積もり

OSS-DB

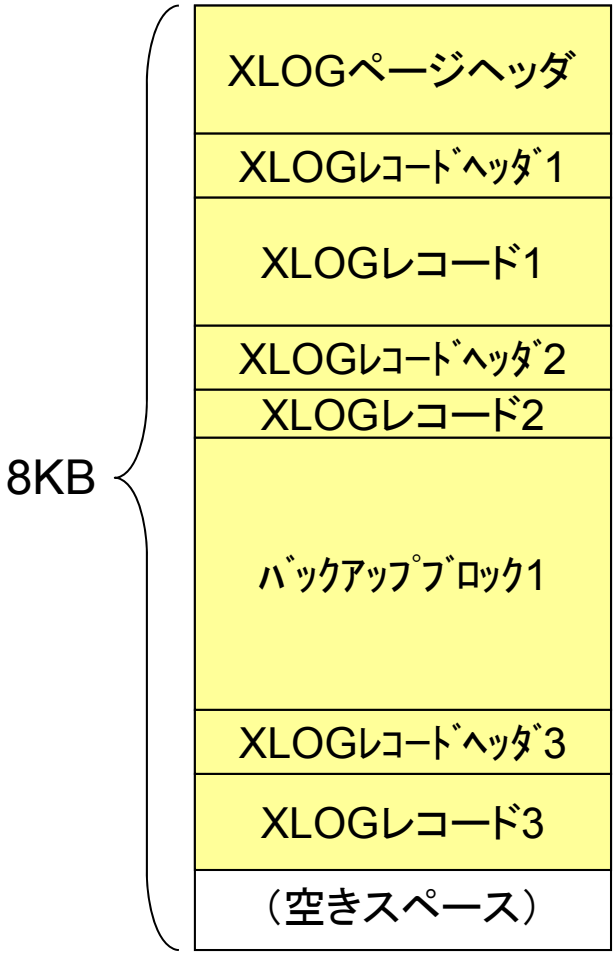
- 最大容量は 16MB×（checkpoint_segments × 3 + 1）
  - WALセグメントファイル（16MB）
  - 各チェックポイント間の最大WALセグメント数（checkpoint_segments）
  - WALを保持しているチェックポイント数（3）



XLOGブロックのレイアウト

OSS-DB

- WALファイルの8kBのXLOGブロックは、XLOGページヘッダ、XLOGレコードヘッダ、XLOGレコード、およびバックアップブロックで構成される。



- XLOGページヘッダを除くスペースを、前から使う。
- XLOGレコードヘッダは、前のXLOGレコードの位置、トランザクションID、更新の種別などを保持する。
- XLOGレコードは、実際の更新レコードを保持する。
- チェックポイント発生直後のページ更新の際、ページ全体をバックアップブロックとして保存する。（full_page_writesオプション）

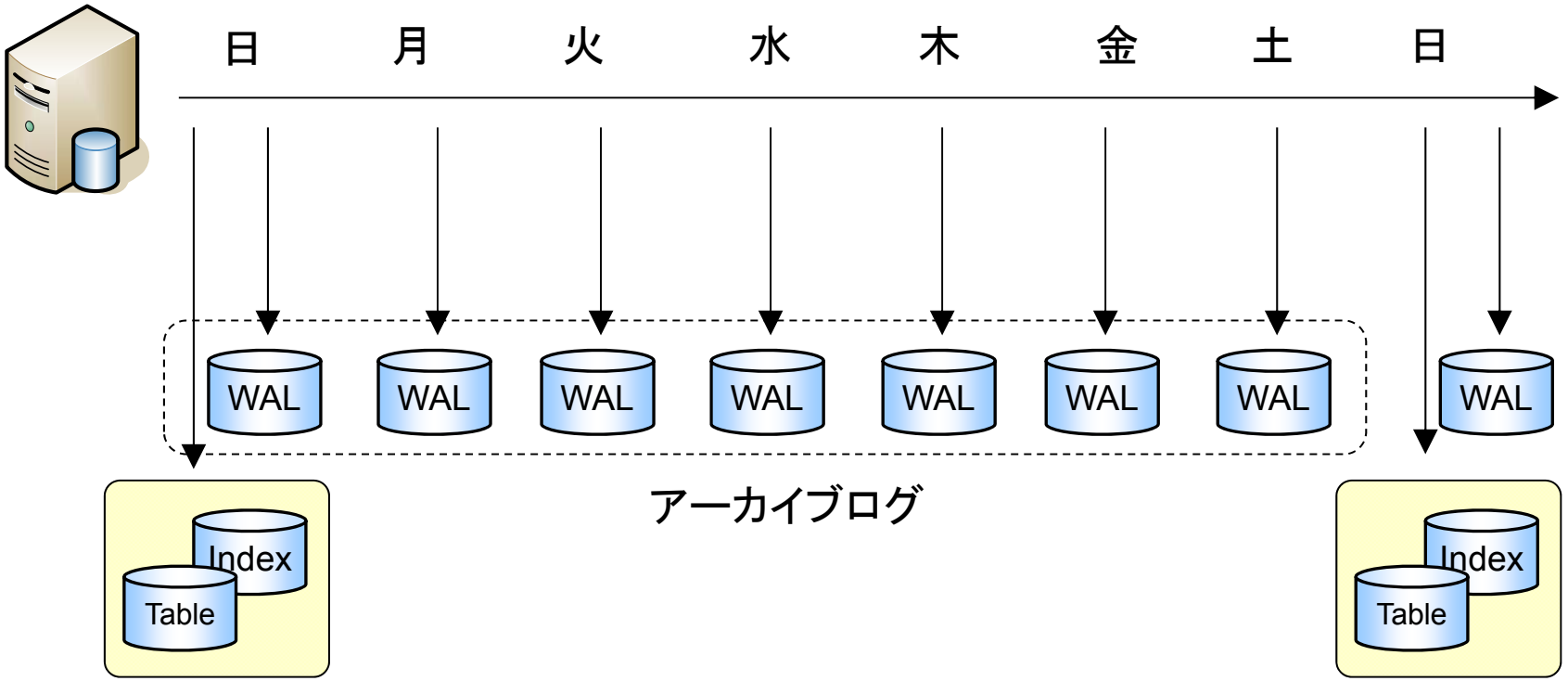
- XLOGページヘッダ（XLogPageHeaderData 16バイト）
- XLOGレコードヘッダ（XLogRecord 26バイト）
- XLOGレコード（可変長）

アーカイブログ領域の見積もり

OSS-DB

ベースバックアップの間に生成されるWALのログ量

- ベースバックアップが週一回であれば、一週間で生成されるWAL量。
- レストア／リカバリ時には一括して配置できるだけのストレージ容量が必要。



WAL生成量の計測

OSS-DB

■計測手順

- 開始時点でのWALファイル名を取得する
- 終了時点でのWALファイル名を取得する
- 「開始・終了地点間で生成されたWALファイル数×16MB」で算出

■WALファイル名の取得方法

- pg_current_xlog_location ()　　現在のWAL位置の取得
- pg_xlogfile_name_offset ()　　WAL位置に該当するWALファイル名の取得

9.24. システム管理関数
http://www.postgresql.jp/document/9.0/html/functions-admin.html

（5）初期設定

PostgreSQLの設定

OSS-DB

カーネルパラメータ

- 共有メモリ、セマフォの設定
- ハードウェアのスペックによっては、デフォルトのままではPostgreSQL起動時にエラーとなる

postgresql.conf

- PostgreSQLパラメータ設定ファイル
- initdbコマンドでデータベースクラスタを作成すると生成される

pg_hba.conf

- ホストベースアクセス認証(HBA)設定ファイル
- 接続元のホスト情報(IP)を使ってアクセス制御を行う

カーネルパラメータ

OSS-DB

- 共有メモリ
  - SHMMAX
    - 共有バッファサイズ＋α（PostgreSQL専用サーバの場合）
  - SHMALL
    - SHMMAX / 4096（ページ数指定）

- セマフォ
  - SEMMNS
    - ceil（（max_connections + autovacuum_max_workers）/ 16）* 17
    - max_connections:100（デフォルト値）
    - autovacuum_max_workers:3（デフォルト値）
    - ceil（（1000 + 10）/ 16）* 17 = 1088
  - SEMMNI
    - ceil（（max_connections + autovacuum_max_workers）/ 16）
    - ceil（（1000 + 10）/ 16）= 64

- sysctl.confの変更
  - kernel.shmmax = <SHMMAX>
  - kernel.shmall = <SHMALL>
  - kernel.sem = <SEMMSL> <SEMMNS> <SEMOPS> <SEMMNI>

17.4. カーネルリソースの管理
http://www.postgresql.jp/document/9.0/html/kernel-resources.html

postgresql.conf

OSS-DB

必ず変更すべき項目

- shared_buffers
- checkpoint_segments
- checkpoint_timeout
- wal_buffers
- archive_mode
- archive_command
- archive_timeout

変更を推奨する項目

- log_line_prefix
- log_filename

確認・変更を推奨する項目

- max_connections
- log_min_duration

第18章 サーバの設定
http://www.postgresql.jp/document/9.0/html/runtime-config.html

pg_hba.conf

OSS-DB

■認証に行うための6項目を1エントリ1行として記述する。

- 接続方法
  - local, host, hostssl, hostnossl
- データベース名
  - all, <データベース名>
- ユーザ名
  - all, +<グループ名>, <ユーザ名>
- 接続元IPアドレス
  - 192.168.0.0/255.255.255.0 など
- 認証方法
  - trust, md5, password, ident, pam, krb5, ldap, 等

```
# "local" is for Unix domain socket connections only
local   all             all                                     ident
# IPv4 local connections:
host    all             all             127.0.0.1/32            ident
# IPv6 local connections:
host    all             all             ::1/128                 ident
```

19.1. pg_hba.confファイル http://www.postgresql.jp/document/9.0/html/auth-pg-hba-conf.html

（6）パフォーマンス管理

パフォーマンスは何で決まるか？

OSS-DB

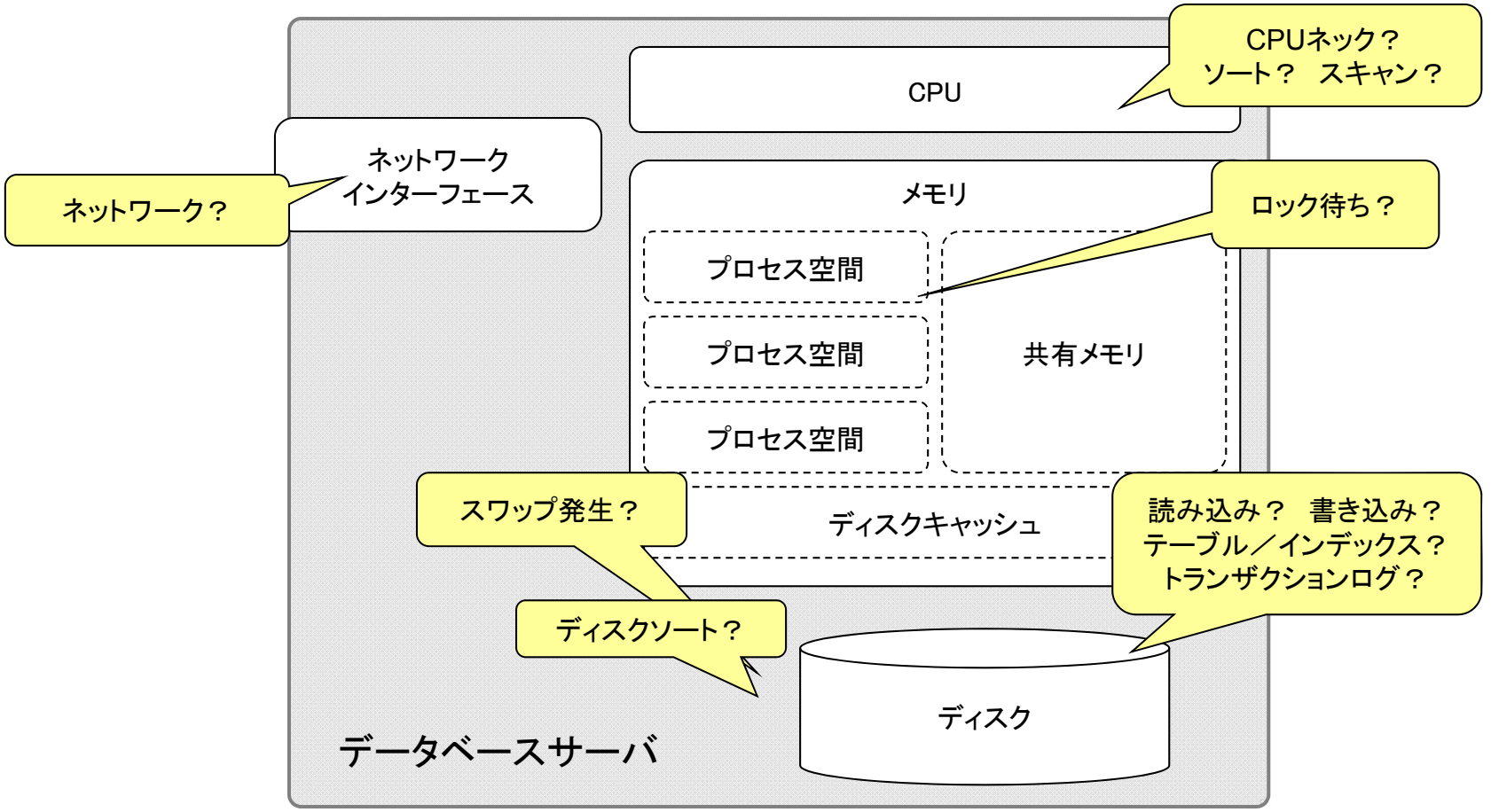
■「単一クエリのレスポンス×クエリの同時実行数」
- 単一クエリのレスポンス
  - サーバ・クライアント間通信（ネットワーク）
  - SQLの構文解析、最適化（CPU処理）
  - ロックの競合（ロック待ち、デッドロックの発生）
  - テーブル、インデックス、ログへのI/O量（ディスクI/O）
  - ソート、結合などの演算処理（CPU処理、ディスクI/O）
- クエリの同時実行数
  - 接続クライアント数（いわゆるWebユーザ数）
  - コネクションプール接続数

■全体としてハードウェアのキャパシティの範囲内であるか？
- ネットワーク、ディスクI/O、メモリ、CPUなどがボトルネックとなり得る。
- ただし、ボトルネック自体は「結果」であり、「原因」ではない。
- 「なぜ、それがボトルネックになっているのか？」が重要。
  - テーブル設計？　SQL文？　同時接続数？　HW？　設定パラメータ？・・・

データベースを構成するハードウェアリソース

OSS-DB

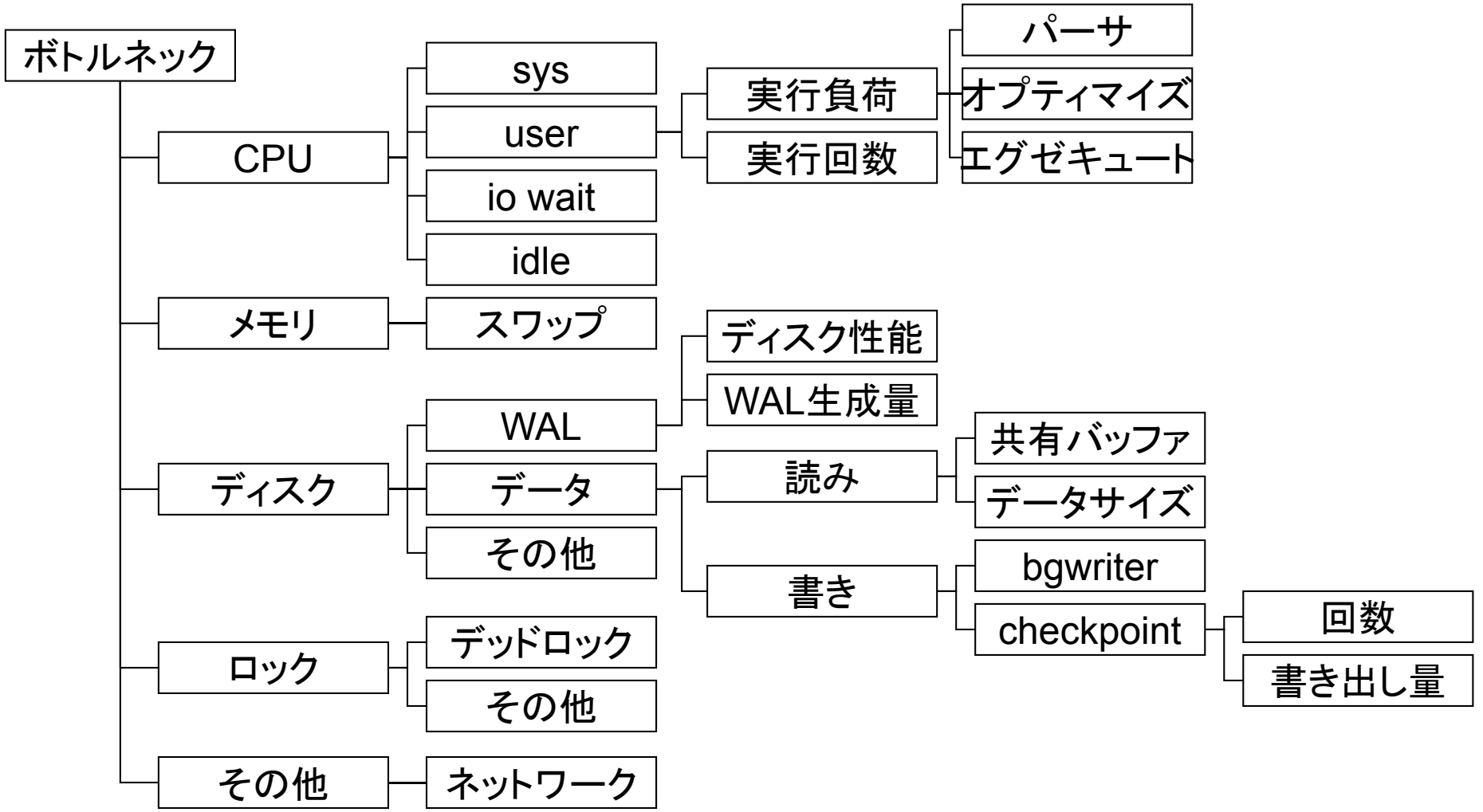
- 複雑な構造を持つRDBMSでは、ボトルネックはいたるところに発生し得るため、まずはきちんと切り分けることが重要。
  - いきなりパラメータチューニングとかを始めない。



パフォーマンス問題の切り分け

OSS-DB

■データベースの構成要素ごとに分解していく



パフォーマンス改善の基本手順

OSS-DB

- 全体のパフォーマンスの傾向をつかむ
  - どのデータベース、テーブルへのアクセスか？　HWの利用状況はどうか？
  - どのメトリックスとどのメトリックスが相関があるか？
- 遅いSQL文を特定する or 実行回数の多いSQLを特定する
  - log_min_durationオプション
  - pgFouine
- 特定のSQLだけが遅い場合・・・
  - SQLのクエリプランおよび実行状況を確認する（EXPLAIN）
- 遅いSQLが特定されない（偏りがない）場合・・・
  - ハードウェアリソースのボトルネックを探す
- 対策を実施する
  - SQL文を書き換える、インデックスを張る、テーブル設計を修正する
  - アプリケーションを修正する
  - ハードウェアを増強する
  - 他・・・

（7）データベースの監視

なぜ「監視」が重要なのか？

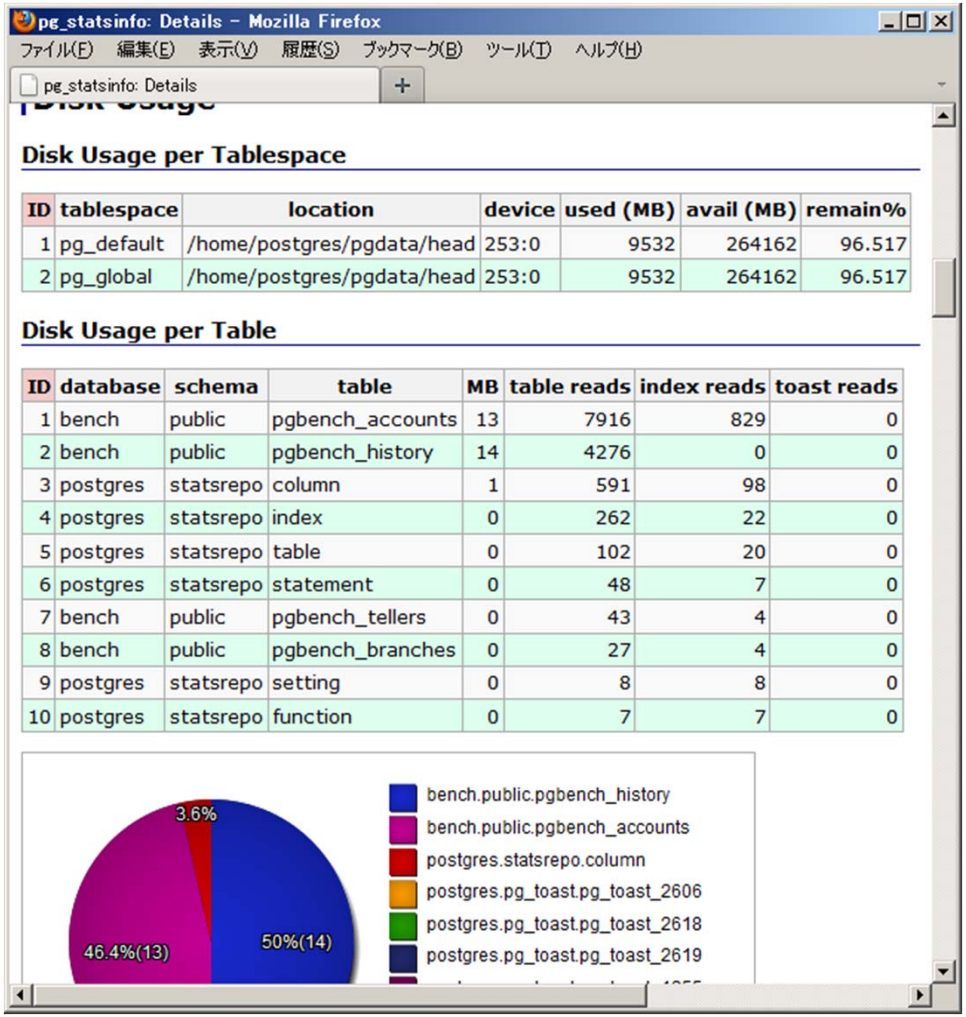
OSS-DB

- PDCA(Plan-Do-Check-Action)を回すため
  - データベースがきちんとサービスを提供しているか？
  - 性能レベルが落ちていないか？
- 監視は「Action」につなげるための「Check」
  - チューニングを行う
  - ハードウェアの増強を行う
  - メンテナンスを行う
- 「何のために、何を監視するのか」
  - あらかじめ決めておくことが重要

全体の傾向を可視化する

OSS-DB

- pg_statinfo/pg_reporterを使って、アクセス統計情報を可視化する。
  - データベース統計情報
  - ディスク使用状況
  - テーブル統計情報
  - チェックポイント情報
  - Autovacuum実行状況
  - SQL文実行状況
  - 等・・・



pg_statsinfo: Project Home Page
http://pgstatsinfo.projects.postgresql.org/

SQLパフォーマンス分析

OSS-DB

- pgFouineによる問題SQL文の抽出、ランキング作成
  - 総実行時間＝レスポンスタイム（実行時間）×実行回数
  - 最長レスポンスタイム
  - 他・・・

pgFouine - a PostgreSQL log analyzer
http://pgfouine.projects.postgresql.org/

pgFouine: PostgreSQL log analysis report – Mozilla Firefox

pgFouine: PostgreSQL log analysis report

Overall statistics | Queries by type | Queries that took up the most time (N) | Slowest queries | Most frequent queries (N) | Slowest queries (N)

Normalized reports are marked with a "(N)".

- Generated on 2011-01-13 16:19
- Parsed /home/snaga/pgsql/pgfouine-1.2/dbt1-001.log (7,000 lines) in 7s
- Log from 2011-01-13 16:14:42 to 2011-01-13 16:16:39
- Executed on devwa02.uptime.jp

Overall statistics

- Number of unique normalized queries: 14
- Number of queries: 3,506 (identified: 3,493)
- Total query duration: 25m27s (identified: 24m49s)
- First query: 2011-01-13 16:14:42
- Last query: 2011-01-13 16:16:39
- Query peak: 49 queries/s at 2011-01-13 16:16:01

Queries by type

Type	Count	Percentage
SELECT	3,493	100.0

Queries that took up the most time (N)

Rank	Total duration	Times executed	Av. duration (s)	Query
1	14m	239	3.52	SELECT * FROM search_results_author('', 0) AS l(i_related1 numeric(0),i_related2 numeric(0),i_related3 numeric(0),i_related4 numeric(0),i_related5 numeric(0),i_thumbnail1 numeric(0),i_thumbnail2 numeric(0),i_thumbnail3 numeric(0),i_thumbnail4 numeric(0),i_thumbnail5 numeric(0),items numeric(0),i_id1 numeric(0),i_title1 VAR Table of contents VARCHAR(0),a_lname1 VARCHAR(0),i_id2 numeric(0),i_title2 VARCHAR(0),a_lname2

完了

OSパフォーマンス監視

OSS-DB

- vmstat
- iostat
- mpstat
- sar
- ps
- free

監視すべき項目とその方法

OSS-DB

■オブジェクトサイズ
- データベースサイズ
  - pg_database_size () 関数
- テーブルサイズ
  - pg_relation_size () 関数、pg_total_relation_size () 関数

■トランザクション量（論理I/O）
- コミット数、ロールバック数（データベース単位）
  - pg_stat_databaseシステムビュー
- INSERT/UPDATE/DELETE数（テーブル/インデックス単位）
  - pg_stat_user_tables/pg_stat_user_indexesシステムビュー

■ディスクI/O量（物理I/O）
- ブロック読み込み、キャッシュ読み込み（データベース単位）
  - pg_statio_databaseシステムビュー
- ブロック読み込み、キャッシュ読み込み（テーブル/インデックス単位）
  - pg_statio_user_tables/ pg_statio_user_indexesシステムビュー

オブジェクトサイズ

OSS-DB

データベース、テーブルサイズ取得用関数

- pg_database_size ()
  - データベースのサイズ
- pg_relation_size ()
  - テーブルのみのサイズ
- pg_total_relation_size ()
  - テーブルとインデックスのサイズ

使い方

- SELECT pg_database_size ('データベース名')
- SELECT pg_relation_size ('テーブル名')

オブジェクトサイズの取得(例)

OSS-DB

```
testdb=# SELECT pg_database_size('testdb');
 pg_database_size
------------------
        154749760
(1 row)

testdb=# SELECT pg_relation_size('pgbench_accounts');
 pg_relation_size
------------------
        130826240
(1 row)

testdb=# SELECT pg_total_relation_size('pgbench_accounts');
 pg_total_relation_size
------------------------
              148914176
(1 row)

testdb=#
```

トランザクション量(論理I/O)

OSS-DB

アクセス統計情報(システムビュー)

- pg_stat_database
- pg_stat_user_tables
- pg_stat_user_indexes

使い方

- SELECT * FROM pg_stat_database
- SELECT * FROM pg_stat_user_tables
- SELECT * FROM pg_stat_user_indexes

トランザクション量の取得（例）

OSS-DB

```
testdb=# SELECT * FROM pg_stat_database WHERE datname='testdb';
-[ RECORD 1 ]-+--------
datid         | 24602
datname       | testdb
numbackends   | 35
xact_commit   | 15196
xact_rollback | 5
blks_read     | 34589
blks_hit      | 461781
tup_returned  | 1128545
tup_fetched   | 64539
tup_inserted  | 1015287
tup_updated   | 45255
tup_deleted   | 0

testdb=# SELECT * FROM pg_stat_user_tables WHERE relname='pgbench_accounts';
-[ RECORD 1 ]----+------------------------------
relid            | 24615
schemaname       | public
relname          | pgbench_accounts
seq_scan         | 1
seq_tup_read     | 1000000
idx_scan         | 43424
idx_tup_fetch    | 43424
n_tup_ins        | 1000000
n_tup_upd        | 21714
n_tup_del        | 0
n_tup_hot_upd    | 9517
n_live_tup       | 1000000
n_dead_tup       | 18393
last_vacuum      | 2012-01-12 09:51:52.548295+09
last_autovacuum  |
last_analyze     | 2012-01-12 09:51:52.858261+09
last_autoanalyze |
```

ディスクI/O量(物理I/O)

OSS-DB

アクセス統計情報(システムビュー)

- pg_statio_user_tables
- pg_statio_user_indexes

使い方

- SELECT * FROM pg_statio_user_tables
- SELECT * FROM pg_statio_user_indexes

ディスクI/O量の取得(例)

OSS-DB

```
testdb=# SELECT * FROM pg_statio_user_tables WHERE relname='pgbench_accounts';
-[ RECORD 1 ]---+-----------------
relid           | 24615
schemaname      | public
relname         | pgbench_accounts
heap_blks_read  | 29946
heap_blks_hit   | 203136
idx_blks_read   | 4363
idx_blks_hit    | 232818
toast_blks_read |
toast_blks_hit  |
tidx_blks_read  |
tidx_blks_hit   |
```

セッション情報

OSS-DB

接続されているセッションの状態を一覧で表示する

- pg_stat_activityシステムビュー

datid	接続しているデータベースのOID
datname	接続しているデータベースのデータベース名
procpid	バックエンド(postgresプロセス)のプロセスID
usesysid	接続しているユーザのOID
usename	接続しているユーザのユーザ名
application_name	接続しているアプリケーション名
client_addr	接続元のクライアントIPアドレス
client_port	接続元のポート番号
backend_start	バックエンドへのセッションが開始された時刻
xact_start	現在のトランザクションが開始された時刻
query_start	現在のクエリの実行が開始された時刻
waiting	ロック待機状態
current_query	現在実行中のクエリ

27.2. 統計情報コレクタ
http://www.postgresql.jp/document/9.0/html/monitoring-stats.html

セッション情報の取得(例)

OSS-DB

```
postgres=# ¥x
Expanded display is on.
postgres=# SELECT * FROM pg_stat_activity;
-[ RECORD 1 ]----+--------------------------------
datid            | 11826
datname          | postgres
procpid          | 4944
usesysid         | 10
usename          | postgres
application_name | psql
client_addr      |
client_port      | -1
backend_start    | 2012-01-13 15:21:23.715083+09
xact_start       | 2012-01-13 15:21:38.583246+09
query_start      | 2012-01-13 15:21:38.583246+09
waiting          | f
current_query    | SELECT * FROM pg_stat_activity;

postgres=#
```

pg_stat_statements

OSS-DB

- 実行したSQL文の情報(SQL文、回数、時間、ブロックアクセス)を表示
  - contribモジュールとしてPostgreSQLソースに同梱

```
snaga@devvm03:~/github/xlogdump
snaga=# SELECT * FROM pg_stat_statements WHERE query LIKE 'SELECT %' LIMIT 1;
-[ RECORD 1 ]-------+------------------------------------------------------------
userid              | 16384
dbid                | 16487
query               | SELECT abalance FROM pgbench_accounts WHERE aid = 49252;
calls               | 1
total_time          | 3.1e-05
rows                | 1
shared_blks_hit     | 4
shared_blks_read    | 0
shared_blks_written | 0
local_blks_hit      | 0
local_blks_read     | 0
local_blks_written  | 0
temp_blks_read      | 0
temp_blks_written   | 0

snaga=#
```

F.29. pg_stat_statements
http://www.postgresql.jp/document/9.0/html/pgstatstatements.html

ロック情報

OSS-DB

ロックの状態を一覧で表示するシステムビュー

- pg_locksシステムビュー

locktype	ロック種別
database	ロック対象のオブジェクトがあるデータベースOID
relation	ロック対象のテーブルのテーブルOID
page	ロック対象のページのページ番号（ブロック番号）
tuple	ロック対象のタプルのページ内タプル番号
virtualxid	ロック対象の仮想トランザクションの仮想トランザクションID
transactionid	ロック対象のトランザクションのトランザクションID
classid	関連するシステムカタログのOID
objid	関連するシステムオブジェクトのOID
objsubid	関連する詳細情報
virtualtransaction	ロックを待機／保持している仮想トランザクションID
pid	プロセスID
mode	ロックモード（共有／排他）
granted	獲得状態

45.50. pg_locks
http://www.postgresql.jp/document/9.0/html/view-pg-locks.html

ロック情報の取得(例)

OSS-DB

```
postgres=# SELECT * FROM pg_locks;
-[ RECORD 1 ]------+---------------
locktype           | relation
database           | 11826
relation           | 10985
page               |
tuple              |
virtualxid         |
transactionid      |
classid            |
objid              |
objsubid           |
virtualtransaction | 2/14892
pid                | 4944
mode               | AccessShareLock
granted            | t
```

```
-[ RECORD 2 ]------+---------------
locktype           | virtualxid
database           |
relation           |
page               |
tuple              |
virtualxid         | 2/14892
transactionid      |
classid            |
objid              |
objsubid           |
virtualtransaction | 2/14892
pid                | 4944
mode               | ExclusiveLock
granted            | t

postgres=#
```

（8）バックアップ・リカバリ

バックアップとレストア／リカバリ

OSS-DB

■バックアップの難しさ

- データはファイルの中にだけあるのではない
- 通常は、共有バッファの内容が最新
- ファイルだけバックアップを取ってもダメ
- ミリ秒単位で処理が進む中、すべてを一貫性を保った状態で

■バックアップの種類

- コールドバックアップ
- ホットバックアップ
- アーカイブログバックアップ

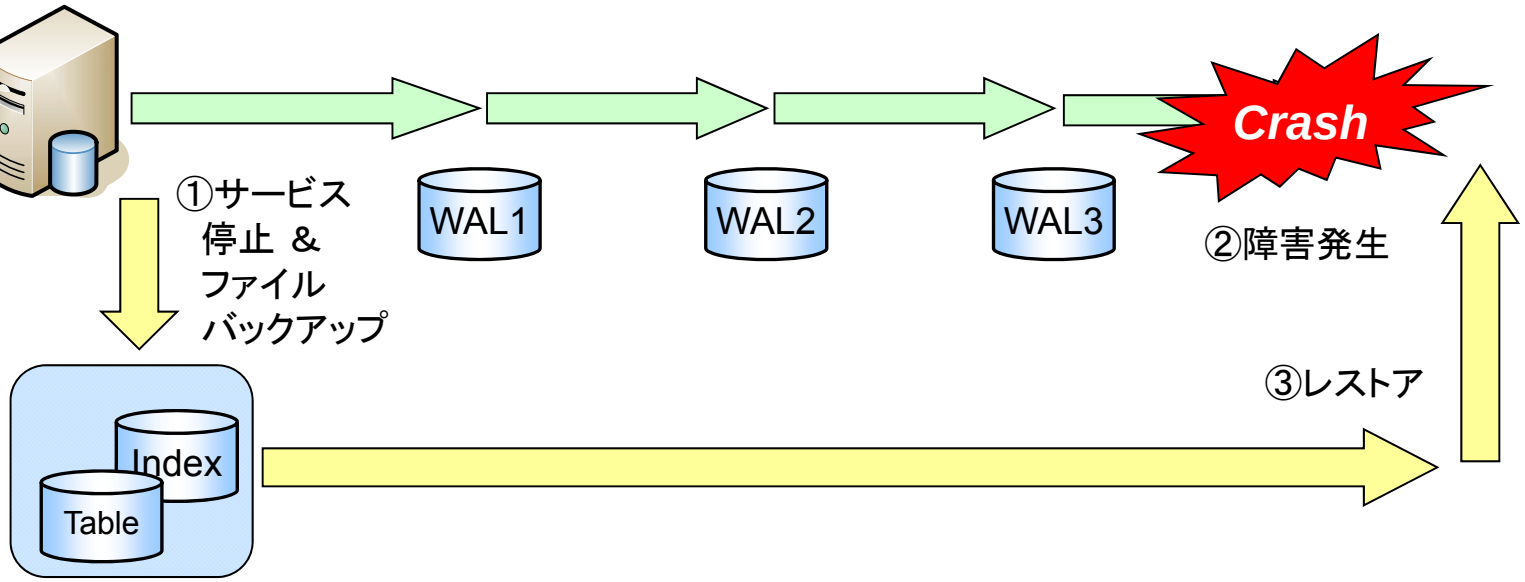
■バックアップ＆レストア／リカバリはリハーサルをしよう！

- 簡単な試験や手順書を作るだけで満足してはいけない・・・

コールドバックアップ

OSS-DB

- サーバプロセスをすべてシャットダウンしてデータファイル全体をバックアップ
  - バックアップの間、サービス停止が発生する。
  - リカバリの際には、バックアップ時のデータに戻る。
  - ファイルバックアップなのでレストアが簡単。
- 向いているケース
  - 前回バックアップ以降の更新データを、アプリログなどから復旧できる場合。
  - ストレージスナップショットが一般化した今、案外現実的。
- 向いていないケース
  - サービスを停止させられない場合。
  - 障害発生の直前までの更新データが必要で、DB以外から復旧できない場合。



コールドバックアップ／レストア手順

OSS-DB

■バックアップ手順

- PostgreSQLシャットダウン
- データベースクラスタ(＋テーブルスペース)のバックアップ
  - データベースクラスタのファイルコピー
  - LVMによるスナップショットバックアップ
  - ストレージ機能によるスナップショット取得
- PostgreSQL再起動
- (スナップショットからのコピー)

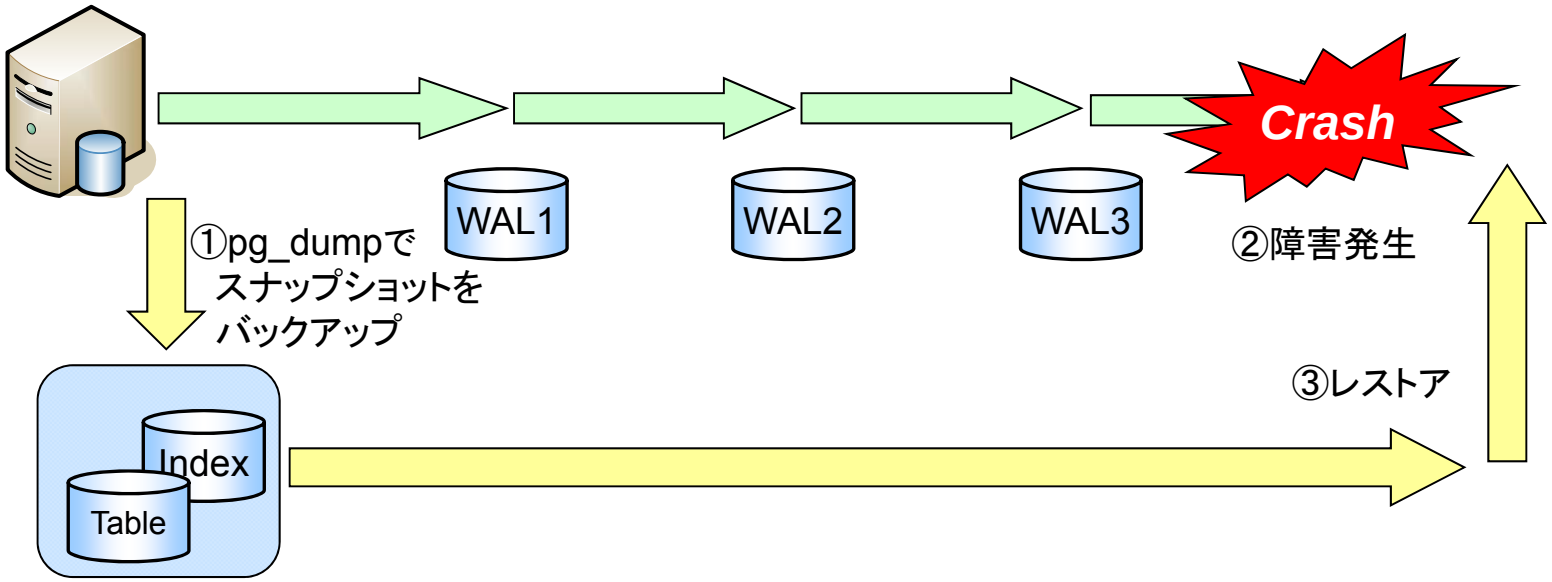
■レストア手順

- PostgreSQLの停止
- 既存データベースクラスタの削除
- データベースクラスタ(＋テーブルスペース)のレストア
- PostgreSQLの再起動

ホットバックアップ（pg_dump/pg_restore）

OSS-DB

- あるタイミングでデータの一貫性を保ちつつバックアップ（export）
  - シンプルかつ柔軟（テーブル単位のバックアップも可）
  - バックアップ時にサービス停止は起こらない。
  - リカバリの際には、バックアップ時のデータに戻る。
- 向いているケース
  - 前回バックアップ以降の更新データを、アプリログなどから復旧できる場合。
  - データベース単位、テーブル単位でバックアップを取りたい場合。
  - 論理バックアップが必要な場合（メジャーバージョンアップなど）
- 向いていないケース
  - 障害発生の直前までの更新データが必要で、DB以外から復旧できない場合。



pg_dumpコマンドによるホットバックアップ

OSS-DB

- pg_dump［*connection-option...*］［*option...*］［*dbname*］
  - PostgreSQLデータベースをスクリプトファイルまたは他のアーカイブファイルへ抽出する

- 実行例：
  - カスタム形式のアーカイブファイルにデータベースをダンプします。
  - $ pg_dump -Fc mydb > db.dump

- 良く使うオプション
  - -F, --format
  - --schema-only, --data-only
  - -t *table*
  - --inserts

pg_dumpによるバックアップファイル

OSS-DB

■プレーンテキスト形式の場合（一部抜粋）

```
--
-- PostgreSQL database dump
--
SET search_path = public, pg_catalog;
SET default_tablespace = '';
SET default_with_oids = false;
--
-- Name: pgbench_accounts; Type: TABLE; Schema: public; Owner: snaga;
  Tablespace:
--
CREATE TABLE pgbench_accounts (
    aid integer NOT NULL,
    bid integer,
    abalance integer,
    filler character(84)
)
WITH (fillfactor=100);

ALTER TABLE public.pgbench_accounts OWNER TO snaga;
```

ホットバックアップからのレストア

OSS-DB

- pg_dumpをオプション指定なしで実行してバックアップを取った場合は、（新規に作成するなどした）空のデータベースに対してpsqlコマンドでレストアを行う。
  - pg_dump testdb > testdb.dmp
  - psql -f testdb.dmp testdb
- pg_dumpをカスタムフォーマットを指定した場合には、（新規に作成するなどした）pg_restoreコマンドを使ってレストアを行う。
  - pg_dump -Fc testdb > testdb.dmp
  - pg_restore -d testdb testdb.dmp

（９）PITRによるバックアップ

用語

OSS-DB

■オンラインWALファイル
- pg_xlogディレクトリに配置されている(まだアーカイブされていない)WALファイル

■アーカイブWALファイル(アーカイブログ)
- アーカイブされたWALファイル

■完全リカバリ
- (オンラインWALファイルを用いて)最新の状態まで戻すことのできるリカバリ

■不完全リカバリ
- オンラインWALファイルを消失したため、最新の状態ではなく、アーカイブWALファイルまでしか戻せないリカバリ

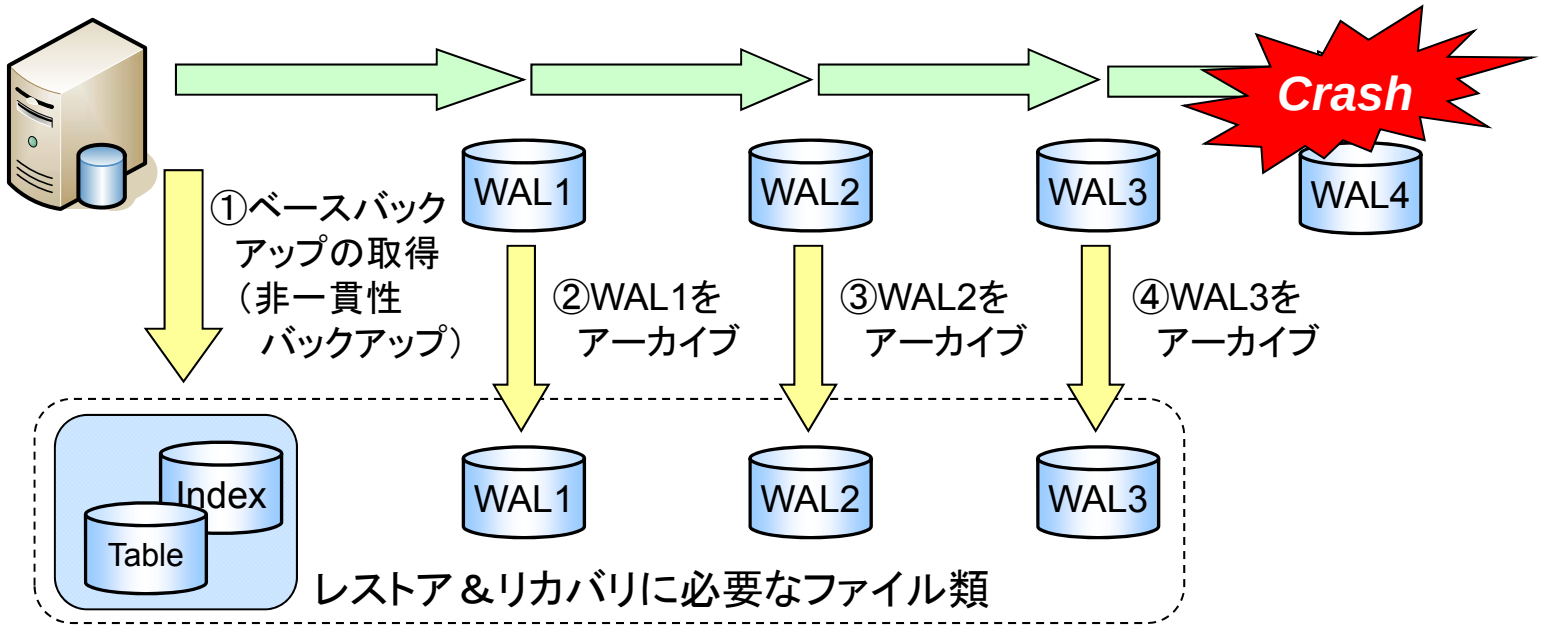
■ベースバックアップ(非一貫性バックアップ)
- 共有バッファなどの状態に関係なく、ファイルシステムレベルで取得するファイルバックアップ。「データベースのファイル」として一貫性の取れた内容である保証は無い。

■タイムライン
- 実施されたリカバリ、およびリカバリ結果を判別するための時間軸

アーカイブログとPITRを用いたバックアップ

OSS-DB

- ベースバックアップ（基準点）＋アーカイブログ（更新差分）
  - サービスを継続したままベースバックアップを取得可能（非一貫性バックアップ）
  - クラッシュ直前のWALの内容まで復旧することが可能
- 向いているケース
  - データベースクラスタ全体の完全なバックアップを取りたい場合。
  - クラッシュ直前の更新まで復旧させる必要がある場合。
- 向いていないケース
  - データベース単位、テーブル単位などでバックアップを取得したい場合。



アーカイブログ関連パラメータ

OSS-DB

- wal_level
  - 生成されるWALレコードの内容を指定する(”minimal”, “archive”, “hot_standby”)
  - アーカイブログを取得する場合には “archive” を指定
- archive_mode
  - アーカイブログ取得モードを設定する(”on” or “off”)
- archive_command
  - オンラインWALファイルをアーカイブするOSコマンド(一般的には cp コマンドなど)
  - 'cp %p /var/lib/pgsql/9.0/backups/archlog/%f '
- archive_timeout
  - 使用中のオンラインWALファイルを強制的にアーカイブする秒数を指定
  - 更新(WALレコード)が少ない場合などでも、確実にアーカイブしたい場合などに設定

18.5. ログ先行書き込み(WAL)
http://www.postgresql.jp/document/9.0/html/runtime-config-wal.html

WALアーカイブ動作詳細

OSS-DB

WALがアーカイブされる契機

- アーカイブタイムアウトの発生
- postmasterの終了
- pg_start_backup () 呼び出し
- pg_stop_backup () 呼び出し
- pg_switch_xlog () 呼び出し

内部でarchive_commandで設定したコマンドが実行される

アーカイブログの動作確認

OSS-DB

■成功している場合

- archive_commandで指定したアーカイブログ領域にファイルがコピーされる

```
$ grep archive_command /var/lib/pgsql/9.0/data/postgresql.conf
archive_command = 'cp %p /var/lib/pgsql/9.0/backups/archlog/%f‘
$ ls -l /var/lib/pgsql/9.0/backups/archlog/
total 147636
-rw------- 1 postgres postgres 16777216 Jan 12 12:41 000000010000000D6000000D7
-rw------- 1 postgres postgres 16777216 Jan 12 12:41 000000010000000D6000000D8
-rw------- 1 postgres postgres 16777216 Jan 12 12:41 000000010000000D6000000D9
-rw------- 1 postgres postgres 16777216 Jan 12 12:41 000000010000000D6000000DA
-rw------- 1 postgres postgres 16777216 Jan 12 12:41 000000010000000D6000000DB
$
```

■失敗している場合

- エラーログを確認

```
2012-01-10 22:45:41 JST 30418 LOG:  archive command failed with exit code 1
2012-01-10 22:45:41 JST 30418 DETAIL:  The failed archive command was: cp
  pg_xlog/000000010000000D600000033
  /var/lib/pgsql/9.0/backups/archlog/000000010000000D600000033
```

ベースバックアップの取得手順と取得対象

OSS-DB

前提条件

- アーカイブログの設定が有効になっていること

取得手順

- pg_start_backup () でバックアップ開始
- データベースクラスタ全体のバックアップを取得
- pg_stop_backup () でバックアップ完了

取得対象

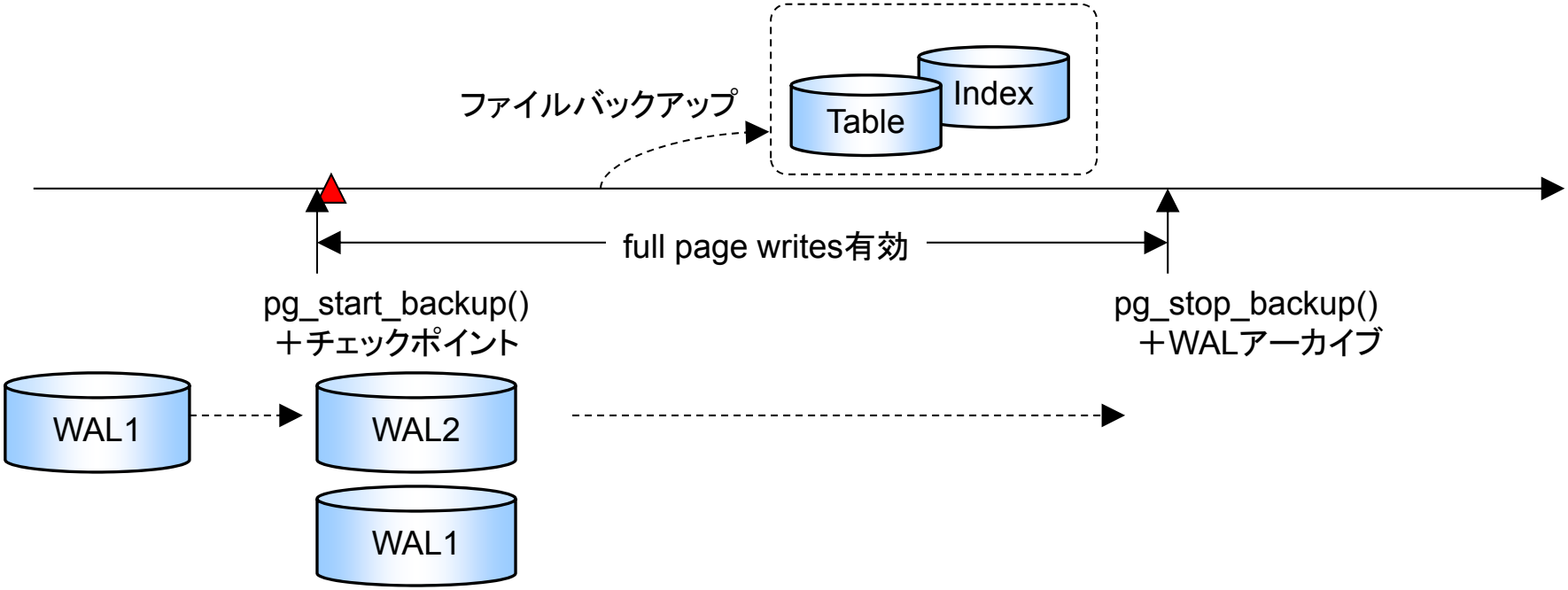
- データベースクラスタ全体
- テーブルスペース（使用している場合）
- XLOGファイル（pg_xlog以下）とpostmaster.pidファイルは除く

ベースバックアップ取得時の動作

OSS-DB

ベースバックアップの動作

- 取得するバックアップは「非一貫性バックアップ」となるため、WAL(アーカイブログ)とセットで一貫性が維持される。
- バックアップ中の更新ブロック、および更新レコードは、WALに記録される(full page writesが強制的に有効化される)。
- リカバリには、ベースバックアップとアーカイブWALが必要。



ベースバックアップの開始処理と終了処理

OSS-DB

- pg_start_backup ('*backuplabel*)
  - WALセグメントの強制スイッチ(アーカイブ)
  - full page writesを有効にする
  - チェックポイントを実行
  - バックアップラベルファイルを作成

- pg_stop_backup ()
  - full page writesの設定を戻す
  - バックアップラベルファイルを読み込み、開始地点を取得
  - バックアップラベルファイルを削除
  - バックアップ開始点をXLOGに記録(バックアップ終了点となる)
  - WALセグメントの強制スイッチ(アーカイブ)
  - バックアップヒストリーファイルを作成

ベースバックアップ取得スクリプト(例)

OSS-DB

手順

- pg_start_backupでベースバックアップを開始
- tarコマンドでベースバックアップを取得
- pg_stop_backupでベースバックアップを終了
- バックアップラベルファイルの内容を表示

```
#!/bin/sh

psql<<__E__
SELECT pg_start_backup('backup test');
__E__

tar cvf /backups/basebackup.tar /var/lib/pgsql/9.0/data

psql<<__E__
SELECT pg_stop_backup();
__E__

cat /var/lib/pgsql/9.0/data/pg_xlog/*.backup
```

ベースバックアップ取得(実行例)

OSS-DB

```
$ sh /backups/basebackup.sh
 pg_start_backup
-----------------
 4/4F00EA14
(1 row)

tar: Removing leading `/' from member names
/var/lib/pgsql/9.0/data/
/var/lib/pgsql/9.0/data/postmaster.pid
/var/lib/pgsql/9.0/data/pg_ident.conf
/var/lib/pgsql/9.0/data/postgresql.conf
/var/lib/pgsql/9.0/data/PG_VERSION
(...snip...)
/var/lib/pgsql/9.0/data/pg_stat_tmp/pgstat.stat
/var/lib/pgsql/9.0/data/pg_tblspc/
/var/lib/pgsql/9.0/data/backup_label
/var/lib/pgsql/9.0/data/postmaster.opts
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
 pg_stop_backup
----------------
 4/516F7068
(1 row)

START WAL LOCATION: 4/4F00EA14 (file 000000090000000040000004F)
STOP WAL LOCATION: 4/516F7068 (file 000000090000000400000051)
CHECKPOINT LOCATION: 4/5086B504
START TIME: 2011-12-12 04:37:20 JST
LABEL: backup test
STOP TIME: 2011-12-12 04:37:32 JST
$
```

バックアップラベルファイル

OSS-DB

- データベースクラスタのディレクトリに作成され、ベースバックアップに含まれて保存される

- ファイル名
  - “backup_label”
- バックアップ開始点
  - START WAL LOCATION: <*xlogid*>/<*xrecoff*> (file <*xlogfile*>)
- チェックポイント点
  - CHECKPOINT LOCATION: <*xlogid*>/ <*xrecoff*>
- バックアップ取得方法
  - BACKUP METHOD: < *‘pg_start_backup’ or ‘streamed’* >
- バックアップ開始時間
  - START TIME: <*YYYY-MM-DD hh:mm:ss zzz*>
- バックアップラベル
  - LABEL: <*backupidstr*>

バックアップヒストリーファイル

OSS-DB

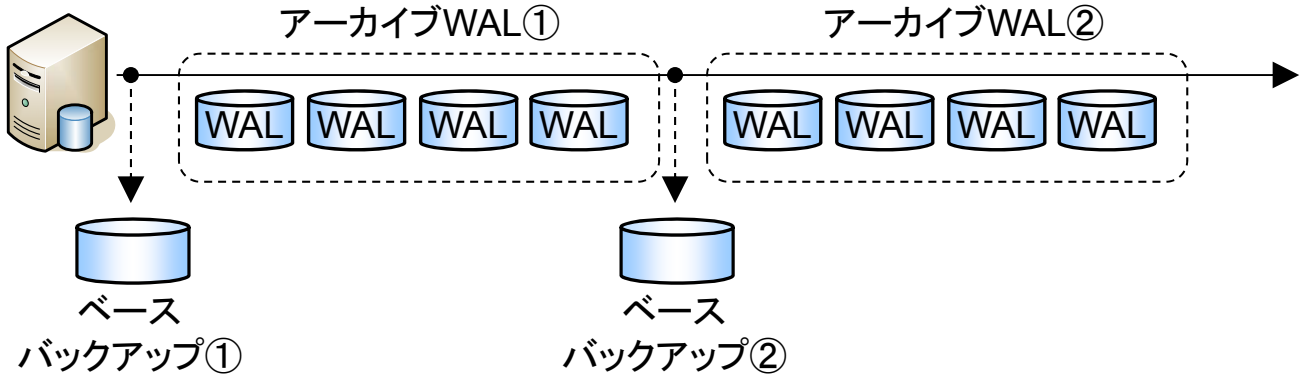
- バックアップラベルにベースバックアップ終了情報が付加され、pg_xlogディレクトリに保存される

- ファイル名（バックアップ開始点でファイル名を生成）
  - *<timelineid><xlogid><xlogseg>.<xlogsegoff>*.backup
- バックアップ開始点
  - START WAL LOCATION: *<xlogid>*/*<xrecoff>* (file *<xlogfile>*)
- バックアップ終了点
  - STOP WAL LOCATION: *<xlogid>*/*<xrecoff>* (file *<xlogfile>*)
- チェックポイント点
  - CHECKPOINT LOCATION: *<xlogid>*/ *<xrecoff>*
- バックアップ取得方法
  - BACKUP METHOD: *< 'pg_start_backup' or 'streamed' >*
- バックアップ開始時間
  - START TIME: *<YYYY-MM-DD hh:mm:ss zzz>*
- バックアップラベル
  - LABEL: *<backupidstr>*
- バックアップ終了時間
  - STOP TIME: *<YYYY-MM-DD hh:mm:ss zzz>*

アーカイブログの消し込み

OSS-DB

- ベースバックアップを取得すると、そのベースバックアップより前のアーカイブログは不要になる
  - 具体的には "START WAL LOCATION" より前のWALファイル
  - ベースバックアップ②を取得したら、アーカイブWAL①は不要



- 消し込みの方法
  - pg_archivecleanupコマンドを使う（contribモジュール）
  - tmpwatchでタイムスタンプで判断（24時間以上経過したら削除、等）

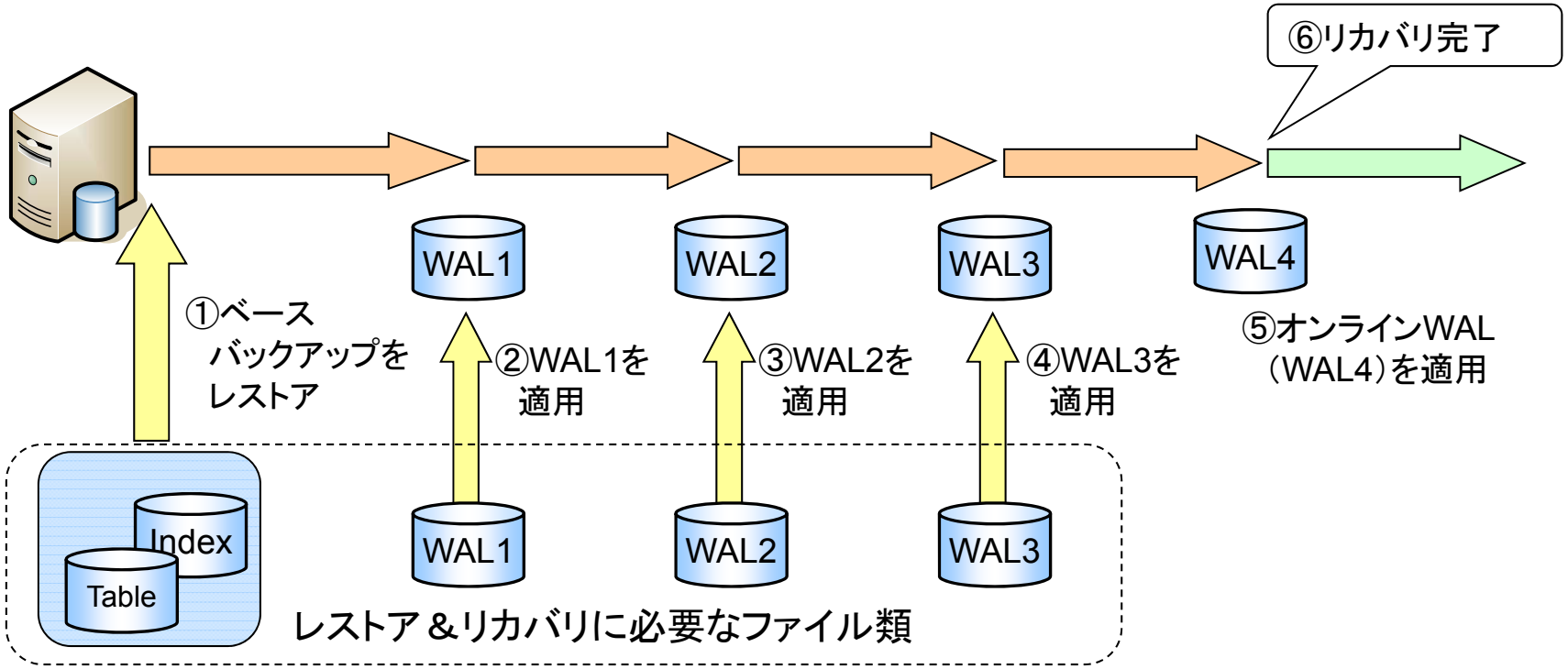
（10）PITRによるリカバリ

アーカイブログとPITRを用いたリカバリ

OSS-DB

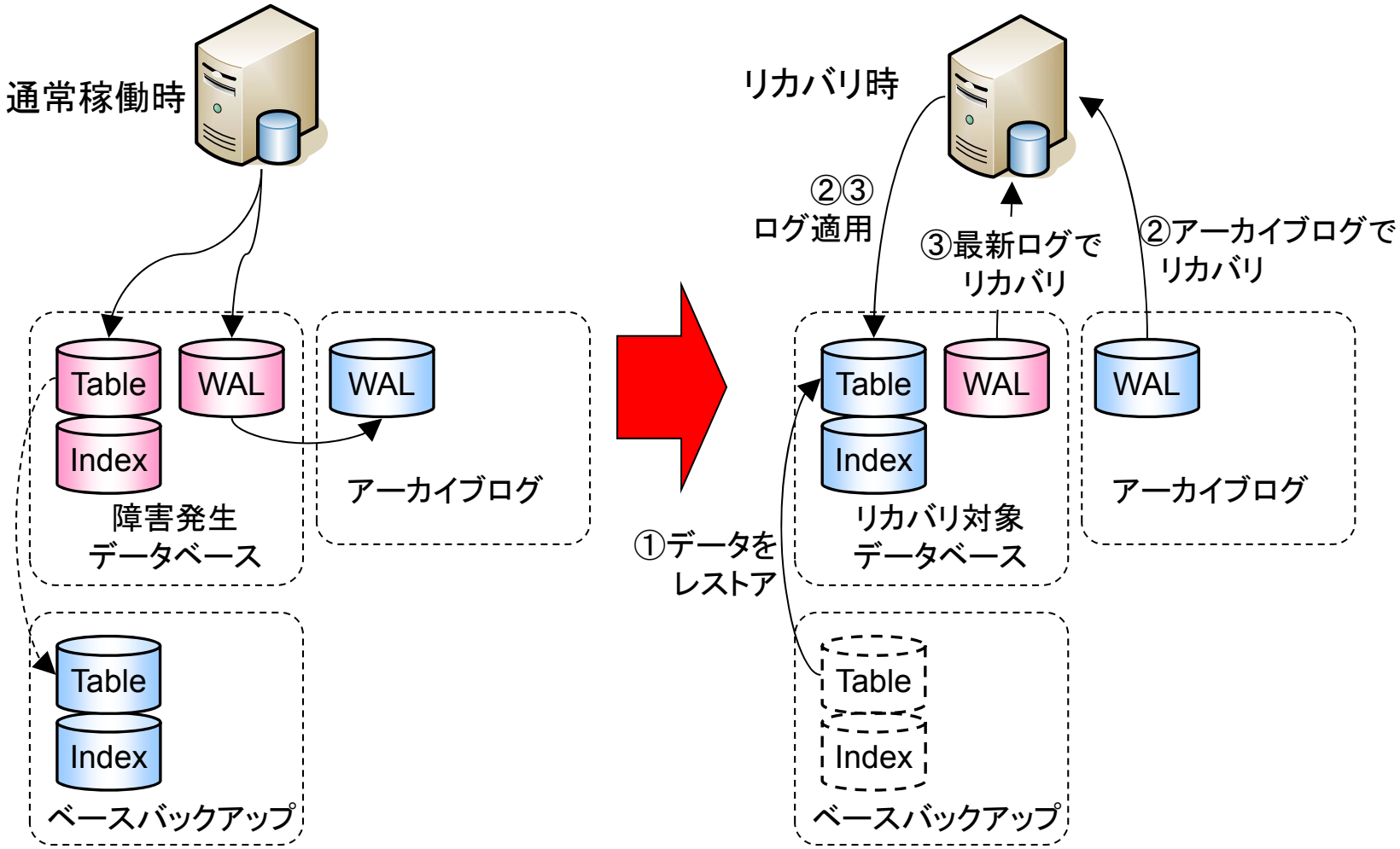
■ベースバックアップ（基準点）＋アーカイブログ（更新差分）

- ベースバックアップをレストア後、アーカイブログをロールフォワードリカバリする。
- 前回のベースバックアップ以降、長期間が経過しているとアーカイブログが多くなり、リカバリの時間が長くなる。
- ベースバックアップレストア時間＋アーカイブログ適用時間×アーカイブログ数



リストア、リカバリ概念図

■ベースバックアップ、アーカイブログ、最新トランザクションログを用いてリカバリを行う。



障害とリカバリのシナリオ

OSS-DB

■データベースクラスタ(テーブルスペース)領域のロスト

- データベースクラスタ(またはテーブルスペース)領域を失った場合には、ベースバックアップからデータベースクラスタをレストアし、アーカイブログを用いてリカバリをする必要があります。
- オンラインWALファイルが残っている場合には「完全リカバリ」が可能です。

■オンラインWAL領域のロスト

- オンラインWALを失うと、PITRによるリカバリは「不完全リカバリ」となります。完全リカバリはできません。
- PostgreSQLが起動しなくなる可能性が高いため、ベースバックアップ＋アーカイブWALからリカバリを実施(不完全リカバリ)。

■アーカイブWAL領域のロスト

- アーカイブWAL領域の障害は、サービスにはすぐには影響しません。
- 但し、アーカイブできなくなると、(再利用できなくなるため)オンラインWAL領域を圧迫し始めるため、アーカイブを再開できるよう、アーカイブWAL領域を復旧させる必要があります。
- また、障害が発生した際には、アーカイブWALがないとリカバリできないため、再度、ベースバックアップを取得する必要があります。

リストア、リカバリ手順

OSS-DB

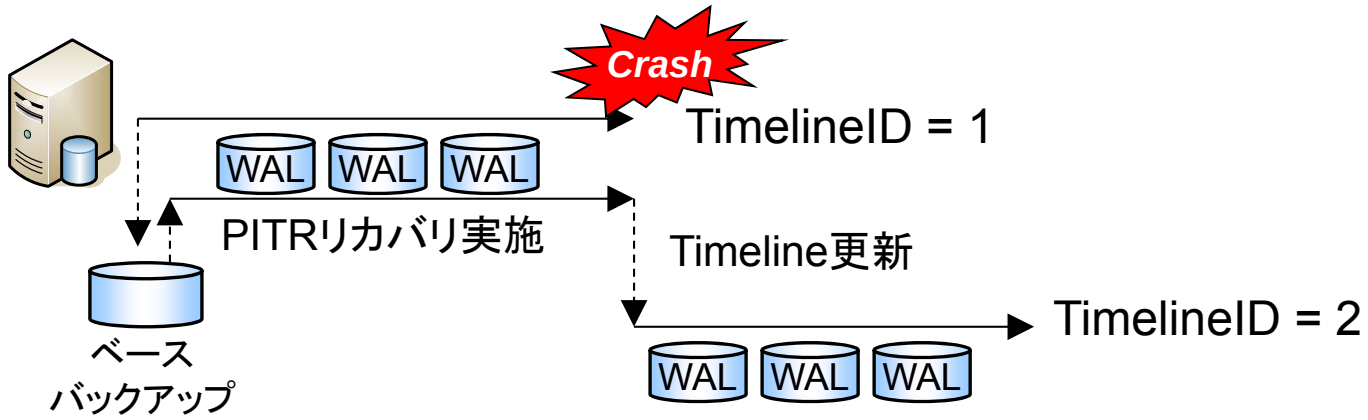
- PostgreSQLサーバを停止する
- 障害の発生したデータベースを保存する(可能であれば)
  - データベースクラスタ
  - トランザクションログ(残っている場合は必ず保護する)
  - テーブルスペース
- ベースバックアップをレストアする
- ベースバックアップ取得以降のアーカイブログをレストアする
- 最新のトランザクションログを配置する
- リカバリ設定ファイル(recovery.conf)を作成する
- PostgreSQLサーバを起動し、リカバリ処理を実行する

タイムライン

OSS-DB

■実施したリカバリを一意に識別するための時間軸

- PITRによるリカバリを行うと、完了した時点でタイムラインIDが繰り上がる
- これによってアーカイブWALファイルが上書きされなくなる



■PITRのリカバリはベースバックアップのタイムラインに沿って実施される

- アーカイブWALファイルの途中でタイムラインが変更されていると、デフォルトではアーカイブWALの適用が行われない。
- 対策1：recovery.confでタイムラインを指定してリカバリを行う。
- 対策2：タイムラインが変わったら即時にベースバックアップを取得し直す。

24.3.4. 時系列
http://www.postgresql.jp/document/9.0/html/continuous-archiving.html

recovery.conf

OSS-DB

restore_command

- アーカイブWALファイルをオンラインWAL用の領域に戻すためのOSコマンド(archive_commandの逆の操作)。
- 通常はcpコマンドなどを指定する。

recovery_target_timeline

- ターゲットとする(到達したい)タイムラインID。
- ベースバックアップからこのタイムラインを目指してアーカイブWALファイルを適用する。

recovery_target_time

- リカバリ(WALファイルのリプレイ)を特定の日時で停止する。
- オペミスや、以前のデータベース状態を復元したい場合に指定。

26.2. リカバリ対象の設定
http://www.postgresql.jp/document/9.0/html/recovery-target-settings.html

PITRの動作状況

OSS-DB

```
[2011-12-12 06:32:52 JST] 31582: LOG:  database system was interrupted; last known up at 2011-
  12-12 06:12:28 JST
[2011-12-12 06:32:52 JST] 31582: LOG:  restored log file "00000002.history" from archive
[2011-12-12 06:32:52 JST] 31582: LOG:  starting archive recovery
[2011-12-12 06:32:52 JST] 31582: LOG:  restored log file "000000010000000000000005" from archive
[2011-12-12 06:32:53 JST] 31582: LOG:  redo starts at 0/5000070
[2011-12-12 06:32:53 JST] 31582: LOG:  consistent recovery state reached at 0/6000000
[2011-12-12 06:32:53 JST] 31582: LOG:  restored log file "000000010000000000000006" from archive
(...snip...)
[2011-12-12 06:33:40 JST] 31582: LOG:  restored log file "00000001000000000000000F" from archive
[2011-12-12 06:33:47 JST] 31582: LOG:  restored log file "000000020000000000000010" from archive
(...snip...)
[2011-12-12 06:34:49 JST] 31582: LOG:  restored log file "00000002000000000000001A" from archive
[2011-12-12 06:34:49 JST] 31582: LOG:  could not open file "pg_xlog/00000002000000000000001B"
  (log file 0, segment 27): No such file or directory
[2011-12-12 06:34:49 JST] 31582: LOG:  redo done at 0/1A00511C
[2011-12-12 06:34:49 JST] 31582: LOG:  last completed transaction was at log time 2011-12-12
  06:23:09.691458+09
[2011-12-12 06:34:49 JST] 31582: LOG:  restored log file "00000002000000000000001A" from archive
[2011-12-12 06:34:49 JST] 31582: LOG:  restored log file "00000003.history" from archive
[2011-12-12 06:34:49 JST] 31582: LOG:  selected new timeline ID: 4
```

リカバリの完了確認

OSS-DB

■完全リカバリの場合

- アーカイブWALファイルからのリカバリを実施、その後、アーカイブWALファイルが無くなってエラーが出るも、その後にはエラーが出ていない。
- これは、アーカイブWALファイルからのリカバリの後、オンラインWALファイルからのリカバリに正常に切り替わったためで、完全リカバリが行われている。
- リカバリが完了したWAL位置は「0/489F8B38」、最終トランザクション時刻は「5時52分01秒」。

```
[2011-12-08 05:59:03 JST] 9003: LOG:  restored log file "000000080000000000000046" from
  archive
[2011-12-08 05:59:03 JST] 9003: LOG:  restored log file "000000080000000000000047" from
  archive
cp: cannot stat `/backups/archlog/000000080000000000000048': No such file or directory
[2011-12-08 05:59:03 JST] 9003: LOG:  record with zero length at 0/489F8B74
[2011-12-08 05:59:03 JST] 9003: LOG:  redo done at 0/489F8B38
[2011-12-08 05:59:03 JST] 9003: LOG:  last completed transaction was at log time 2011-12-
  08 05:52:01.507063+09
```

リカバリの完了確認

OSS-DB

■不完全リカバリの場合

- アーカイブWALファイルからのリカバリの後、オンラインWALファイルからリカバリを行おうとしてエラー。
- これは、オンラインWALファイルが存在しなかったためで、この場合は「不完全リカバリ」となる。
- リカバリが完了したWAL位置は「0/47FFE330」、最終トランザクション時刻は「5時51分54秒」。

```
[2011-12-08 05:56:47 JST] 8849: LOG:  restored log file "000000080000000000000046" from
  archive
[2011-12-08 05:56:48 JST] 8849: LOG:  restored log file "000000080000000000000047" from
  archive
cp: cannot stat `/backups/archlog/000000080000000000000048': No such file or directory
[2011-12-08 05:56:48 JST] 8849: LOG:  could not open file
  "pg_xlog/000000080000000000000048" (log file 0, segment 72): No such file or directory
[2011-12-08 05:56:48 JST] 8849: LOG:  redo done at 0/47FFE330
[2011-12-08 05:56:48 JST] 8849: LOG:  last completed transaction was at log time 2011-12-
  08 05:51:54.085131+09
[2011-12-08 05:56:49 JST] 8849: LOG:  restored log file "000000080000000000000047" from
  archive
```

（11）データベースのメンテナンス

性能劣化の原因

OSS-DB

■データ量の増大

- 実データの増大
  - データが蓄積されていくことによって、実際のデータ量が増大。
- 不要領域の増大
  - データ量は増えていないが、追加・削除・更新を繰り返すことによって、ディスクの利用効率が悪くなり、余計なI/Oが発生。

■問い合わせ処理の増大

- 接続数の増大
- 処理処理の増大

テーブルの不要領域ができる仕組み

OSS-DB

- あるトランザクションによってレコードが(論理的に)削除されると、「削除フラグ」が設定される
  - 物理的にはディスク上に残る
- これは、複数のトランザクションからのレコードの可視性を制御するためである。
  - Multi-Version Concurrency Control (MVCC)
- よって、削除して不要になった領域は、事後的に「再利用可能領域」として回収する必要がある。
  - これが「VACUUM処理」

VACUUMコマンド

OSS-DB

■テーブルの不要領域を回収し、「未使用領域」として記録する。
- 次の更新(追記)の時から、未使用領域を利用できるようになる。

■VACUUMコマンド
- VACUUM ＜テーブル名＞
- VACUUM

23.1. 定常的なバキューム作業
http://www.postgresql.jp/document/9.0/html/routine-vacuuming.html

テーブル/インデックス不要領域の確認

OSS-DB

- contribのpgstattupleモジュールを使用する
- テーブルの不要領域の確認
  - pgstattuple関数
- インデックス(B-Tree)の不要領域の確認
  - pgstatindex関数

F.30. pgstattuple
http://www.postgresql.jp/document/9.0/html/pgstattuple.html

pgstattuple使用方法

OSS-DB

```
pgbench=# ¥x
Expanded display is on.
pgbench=# SELECT * FROM pgstattuple('accounts');
-[ RECORD 1 ]------+----------
table_len          | 138739712
tuple_count        | 1000000
tuple_len          | 128000000
tuple_percent      | 92.26
dead_tuple_count   | 32000
dead_tuple_len     | 4096000
dead_tuple_percent | 2.95
free_space         | 2109248
free_percent       | 1.52
```

自動VACUUM

OSS-DB

■VACUUMをバックグラウンドで自動実行する機能（autovacuum）

- 自動VACUUM起動プロセス（autovacuum launcher）がバックグラウンドに常駐。
- 各データベースに対して一定の周期でVACUUMワーカープロセスを起動。
- レコードの更新、削除が閾値を超えたらVACUUM対象とする
  - autovacuum_vacuum_threshold + autovacuum_vacuum_scale_factor × レコード数
- レコードの追加、更新、削除が閾値を超えたらANALYZE対象とする
  - autovacuum_analyze_threshold + autovacuum_analyze_scale_factor × レコード数

■関連パラメータ（抜粋）

- autovacuum_naptime
  - VACUUMワーカープロセスを起動するインターバル。デフォルトは「1min」。
- autovacuum_max_workers
  - 同時に起動されるVACUUMワーカープロセス数。デフォルトは「3」。
- autovacuum_vacuum_threashold
  - VACUUM処理判定の閾値のベースライン。デフォルトは「50」。
- autovacuum_vacuum_scale_factor
  - VACUUM処理判定の閾値のスケールファクタ。デフォルトは「0.2」。

18.9. 自動Vacuum作業
http://www.postgresql.jp/document/9.0/html/runtime-config-autovacuum.html

遅延VACUUM

OSS-DB

■負荷分散のためVACUUMを「ゆっくり」実行する機能

- VACUUMのI/O処理がパフォーマンスに影響するほど大きな負荷になる場合に使う。
- VACUUMを実行中に「処理コスト」を積算。
- 処理コストが閾値を超えたらスリープする。

■関連パラメータ

- vacuum_cost_page_hit
  - 共有バッファ内のページをVACUUM処理した場合のコスト。デフォルトは「1」。
- Vacuum_cost_page_miss
  - ディスク上のブロックを読み込んでVACUUM処理した場合のコスト。デフォルトは「10」。
- Vacuum_cost_page_dirty
  - VACUUMしたページをディスクに書き戻した場合のコスト。デフォルトは「20」。
- vacuum_cost_limit
  - スリープするコストの閾値。デフォルトは「200」。
- vacuum_cost_delay
  - スリープする時間(ミリ秒)。デフォルトはゼロ(無効)。

18.4.3. コストに基づくVacuum遅延
http://www.postgresql.jp/document/9.0/html/runtime-config-resource.html

VACUUMの動作確認

OSS-DB

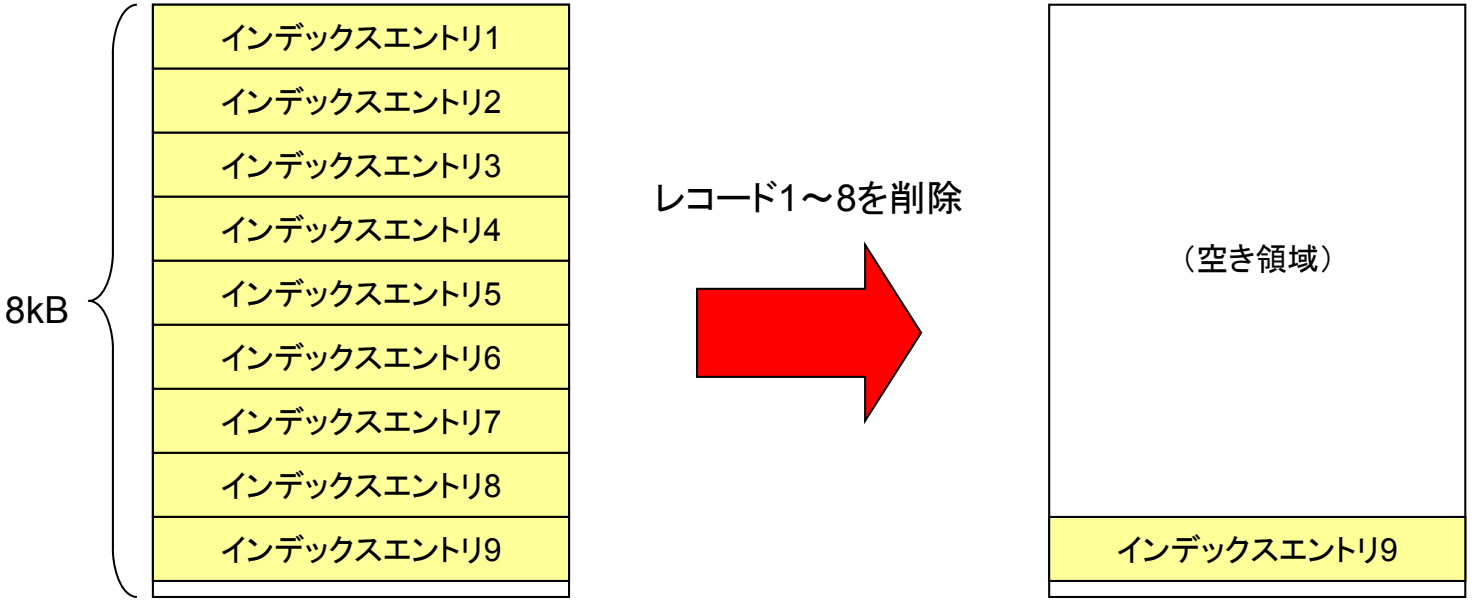
- pg_stat_user_tables システムビュー
  - last_vacuum カラム(Timestamp型)
  - last_autovacuum カラム(Timestamp型)

インデックスの不要領域ができる仕組み

OSS-DB

- インデックス（B-Tree）のリーフノードには、インデックスキーが記録されており、レコードが削除されると、インデックスのキーも不要になる

- VACUUMすると領域は再利用できるようになるが、その領域が使われないとブロックに未使用領域ができ、保持しているインデックスエントリ数に対してブロック利用効率が低下する。（少ないレコードに大きなインデックスファイル）



23.2. 定常的なインデックスの再作成
http://www.postgresql.jp/document/9.0/html/routine-reindex.html

REINDEXコマンド

OSS-DB

■インデックスを再作成する
- 未使用領域のないインデックスが作られる

■REINDEXコマンド
- REINDEX TABLE ＜テーブル名＞
- REINDEX DATABASE ＜データベース名＞

■REINDEX時のロック
- インデックスの元となっているテーブルに対して共有ロックを獲得する(テーブルに書き込みできなくなる)
- 再作成する対象のインデックスに対して排他ロックを獲得する(インデックスを読めなくなる)

REINDEX
http://www.postgresql.jp/document/9.0/html/sql-reindex.html

pgstatindex使用方法

OSS-DB

```
pgbench=# ¥x
Expanded display is on.
pgbench=# SELECT * FROM pgstatindex('accounts_pkey1');
-[ RECORD 1 ]------+---------
version            | 2
tree_level         | 2
index_size         | 17956864
root_block_no      | 361
internal_pages     | 8
leaf_pages         | 2184
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 90.07
leaf_fragmentation | 0
```

データベース統計情報

OSS-DB

■データベース統計情報とは

- オプティマイザが実行計画を作成・最適化する際に利用する情報。
- テーブルのレコード数、NULL値の割合、データの分布や偏りなどをpg_statisticsに保存。
- 統計情報をもとに、オプティマイザが「より効率、パフォーマンスのよい実行計画」を作成する。

■データベース統計情報のメンテナンス

- 通常は自動VACUUMが実施する。
- 大量データの更新(データロード、バッチ更新、消し込み等)の後は、テーブルと統計情報が一致しなくなるため、その場合にはANALYZEコマンドを実行する。
- 一般的には、EXPLAINした際に、「想定している実行プランが出ない」、「表示されるレコード数がおかしい」といった現象として顕在化する。

45.36. pg_statistic
http://www.postgresql.jp/document/9.0/html/catalog-pg-statistic.html
45.57. pg_stats
http://www.postgresql.jp/document/9.0/html/view-pg-stats.html

ANALYZEコマンド

OSS-DB

データベース統計情報を更新する

ANALYZEコマンド

- ANALYZE ＜テーブル名＞
- ANALYZE

ANALYZE
http://www.postgresql.jp/document/9.0/html/sql-analyze.html

（12）パフォーマンスチューニング（GUC）

パフォーマンスチューニングの基本戦略

OSS-DB

- I/Oが出ないようにする
  - キャッシュのヒット率を上げる
    - キャッシュを大きくする
    - データサイズを小さくする(アクセスを局所化する)
  - チェックポイントの間隔を延ばす
- I/Oを平準化する
  - バックグラウンドライタ
  - 遅延VACUUM
- I/Oを分散する
  - オンラインWAL領域を別ディスクにする
  - テーブルスペースを使う

GUCパラメータで設定できる項目

OSS-DB

- 共有バッファ
- WALバッファ
- ワークメモリ(ソートメモリ)
- チェックポイント
- バックグランドライタ
- autovacuum

※GUC(Grand Unified Configuration)
PostgreSQLのパラメータ管理モジュール。postgresql.confで設定・管理する。

共有バッファ、WALバッファ、ワークメモリ

OSS-DB

■共有バッファ

- テーブルやインデックスなどのデータファイルをブロック単位でキャッシュしておく共有メモリ内の領域。
- GUCパラメータの shared_buffers で指定。
- 数GB程度から始め、キャッシュのヒット率を見ながら調整を行う。

■WALバッファ

- WALレコードをディスクに書き出す前にバッファリングされる共有メモリ内の領域。
- GUCパラメータの wal_buffers で指定。
- トランザクションがCOMMITされるとフラッシュされる。
- 同時実行トランザクションが多い場合、または長いトランザクションが多い場合には大きめに設定(16MB、32MBなど)。

■ワークメモリ(ソートメモリ)

- SQLでソート処理を行う際にメモリ内でソートを行える上限値(デフォルトは1MB)。
- GUCパラメータの work_mem で指定。
- EXPLAIN ANALYZEで「Sort Method: external merge Disk: ????kB」が頻発し、パフォーマンスが悪化している場合は増加(同時実行数とメモリ使用量に注意)。

第18章 サーバの設定 http://www.postgresql.jp/document/9.0/html/runtime-config.html

チェックポイント

OSS-DB

チェックポイントとは

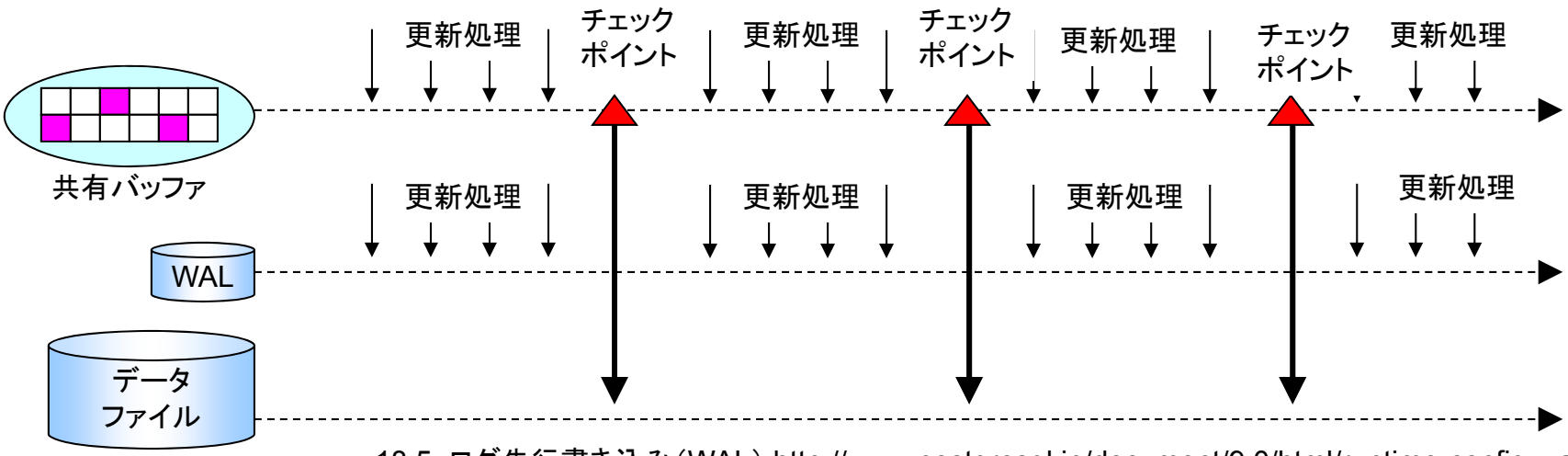
- 共有バッファの内容がディスクに反映されていることを保証する地点。
- クラッシュリカバリの開始点として使われる。

チェックポイントにおける処理

- 共有バッファ内の変更されているページ(dirtyページ)をディスクに一括して書き戻す。

チェックポイントの発生契機

- checkpoint_segments で設定されたWALファイル数の上限に到達。
- checkpoint_timeout で設定されたタイムアウトが発生。
- CHECKPOINTコマンドによる実行。



18.5. ログ先行書き込み（WAL） http://www.postgresql.jp/document/9.0/html/runtime-config-wal.html

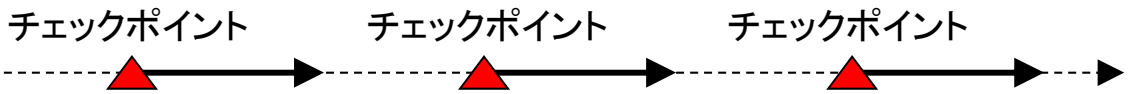
チェックポイントの負荷軽減

OSS-DB

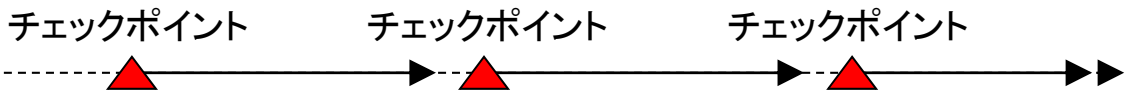
■チェックポイントを実行する時間を延ばして、負荷を低減する。

- checkpoint_completion_targetの指定による遅延チェックポイント。
- 次のチェックポイントが始まるまでの時間に対する割合で指定する。

checkpoint_completion_target = 0.5



checkpoint_completion_target = 0.9



■バックグラウンドライタによるdirtyバッファの抑制（後述）

バックグラウンドライタ

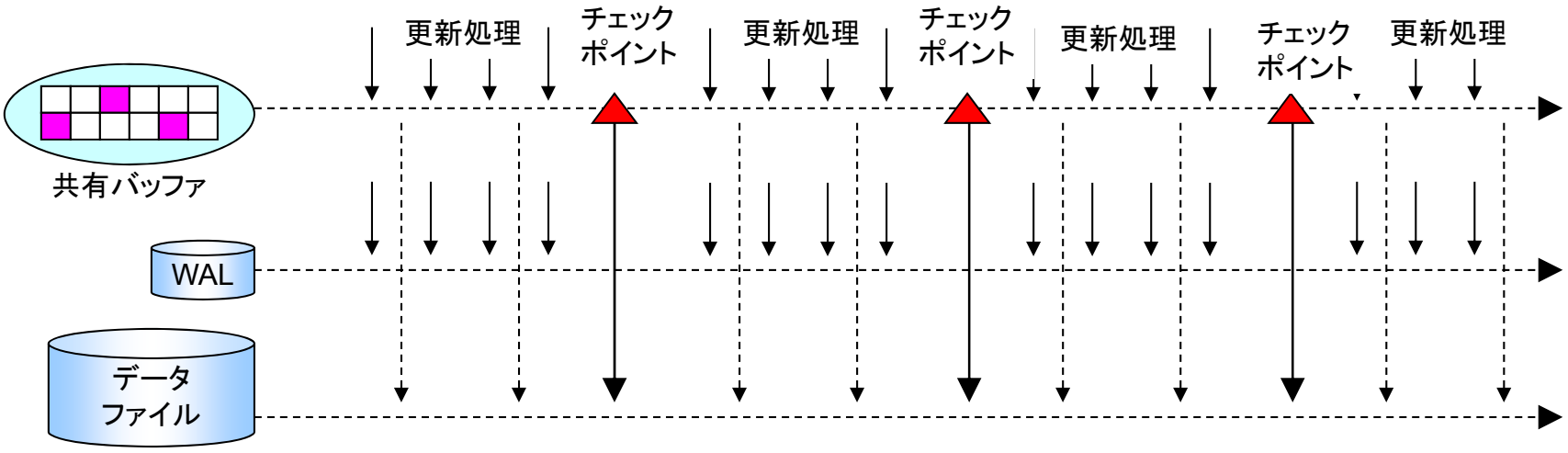
OSS-DB

■バックグラウンドライタとは

- チェックポイントにおけるI/Oの負荷を軽減するため、チェックポイント以外の時間帯に「少しずつ」ディスクに書き戻し、I/O書き出しを平準化するプロセス。

■バックグラウンドライタの処理

- bgwriter_lru_maxpagesで指定した数のdirtyページを書き戻す。
- bgwriter_delayで指定した時間待機する。

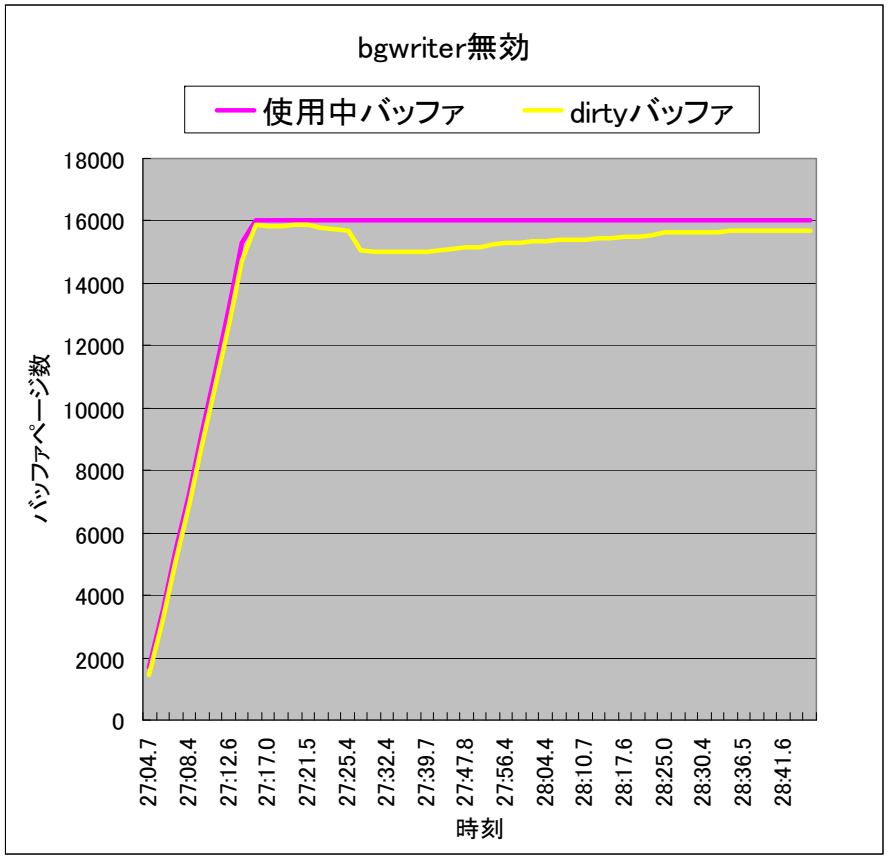


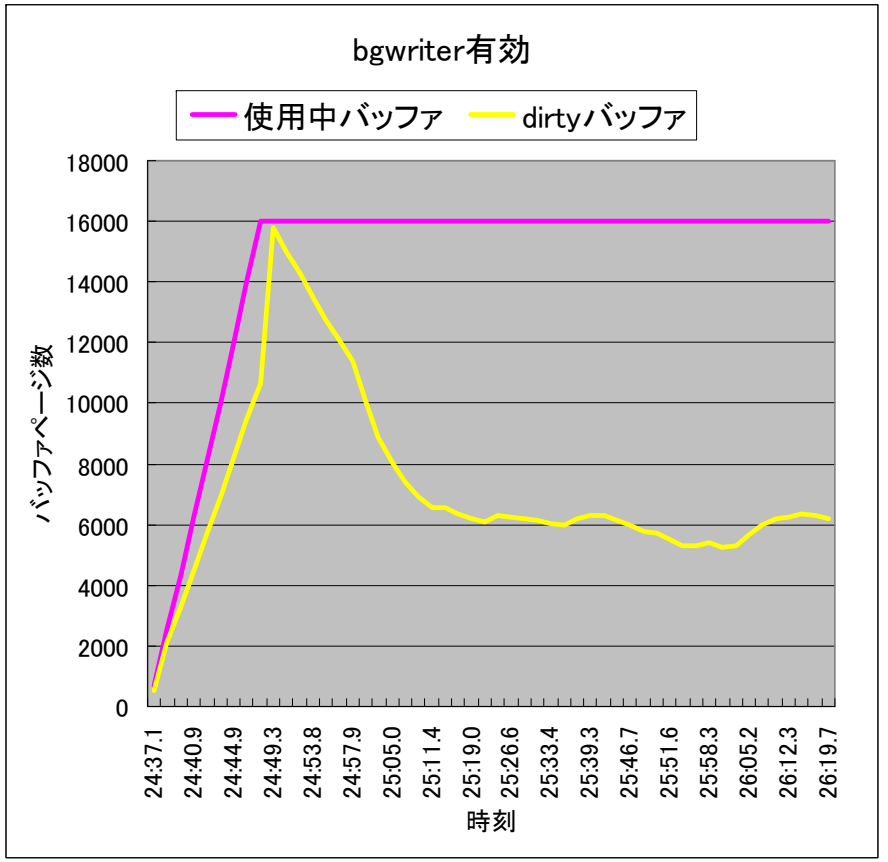
18.4.4. バックグラウンドライタ
http://www.postgresql.jp/document/9.0/html/runtime-config-resource.html

バックグラウンドライタの効果

OSS-DB

- バックグラウンドライタ(bgwriter)は、dirtyバッファを少しずつディスクに書き戻す





F.24. pg_buffercache
http://www.postgresql.jp/document/9.0/html/pgbuffercache.html

バックグラウンドライタ統計情報

OSS-DB

- バックグラウンドライタの活動状況を監視する
  - pg_stat_bgwriterシステムビュー

- checkpoints_timed
  - タイムアウトによって発生したチェックポイントの回数
- checkpoints_req
  - CHECKPOINTコマンドまたは既定のセグメント数に到達したために発生したチェックポイントの回数
- buffers_checkpoint
  - チェックポイント処理においてディスクに書き出されたブロック数
- buffers_clean
  - チェックポイント処理以外においてディスクに書き出されたブロック数
- maxwritten_clean
  - 一回の書き出し最大ブロック数に到達したためbgwriterを途中で停止した回数
- buffers_backend
  - バッファの新規獲得に先立って、ディスクに書き出された回数
- buffers_alloc
  - バッファに読み込まれた回数

パラメータ設定におけるトレードオフ

OSS-DB

■共有バッファを大きくすると・・・
- より多くのディスクブロックを共有バッファに保持できるため、パフォーマンスが向上する。
- 大量のdirtyページが発生するため、チェックポイント時の負荷が高くなる。

■チェックポイントの間隔を大きくすると・・・
- チェックポイントの発生数を抑え、パフォーマンスが向上する。
- チェックポイント時の負荷が高くなる。
- クラッシュリカバリに要する時間が長くなる。

■バックグラウンドライタを頻繁に動かす（多く書き出す）と・・・
- チェックポイントにおける負荷は減るが、書き出しのディスクI/Oが頻発する（書き出しの平準化により）。
- 全体的なパフォーマンスが低下する。（特にディスクが1本の場合）

（13）冗長化

冗長化方式の選定

OSS-DB

■実現方式を評価するに当たって特に重視すべき点

- 負荷分散の必要性の有無。
- 単一障害点(Single Point of Failure、SPoF)の有無。
- 運用が容易であるかどうか(運用の作業負荷、ノウハウの蓄積)。
- データー貫性の厳密性(レプリケーション遅延)の程度。

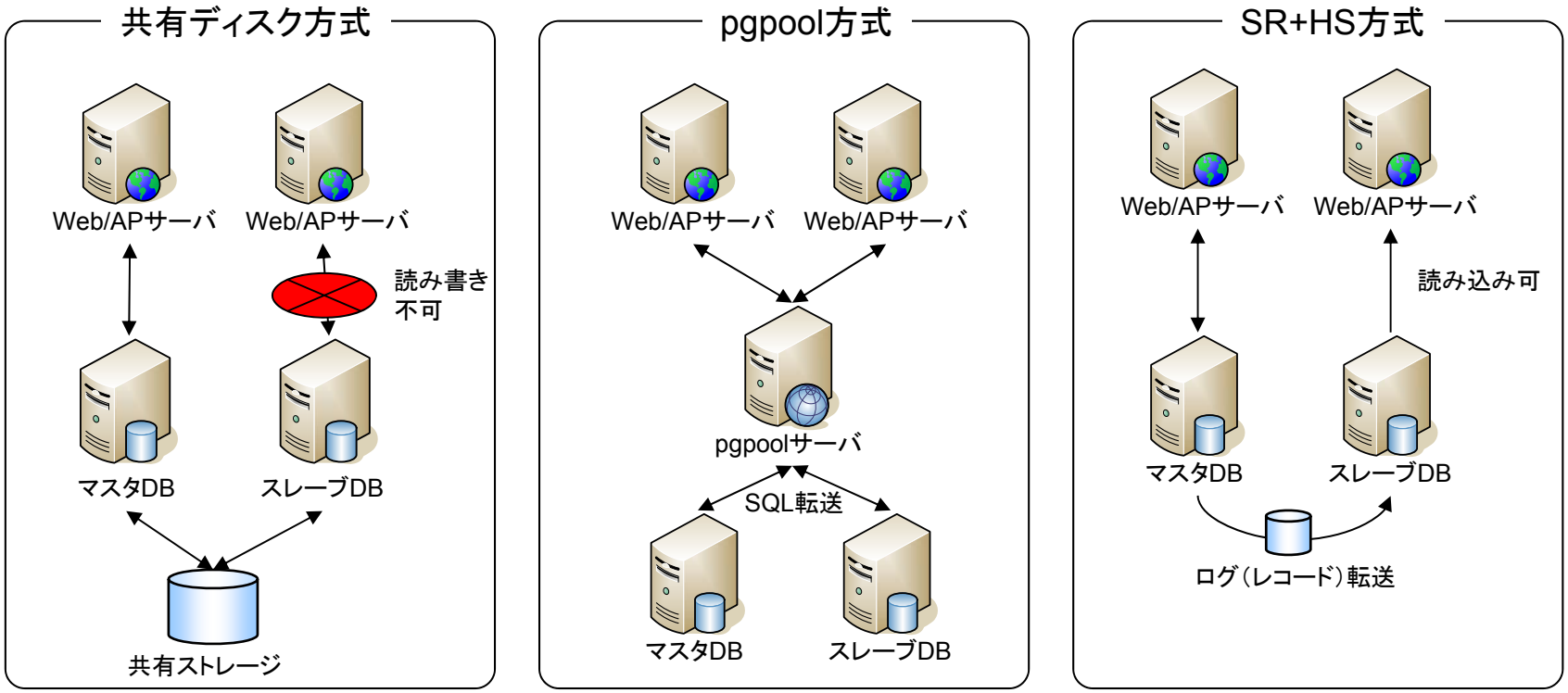
実現方式	アーキテクチャ	負荷分散	同期遅延	運用性	備考
アーカイブログ転送	アクティブ／スタンバイ	×	数十秒 ～数分	◎	ウォームスタンバイ方式。
DRBDディスク同期	アクティブ／スタンバイ	×	なし	△	要DRBD運用ノウハウ。
共有ディスク方式	アクティブ／スタンバイ	×	なし	△	共有ディスクが高価でSPOF。
Slony-Iレプリケーション	アクティブ／アクティブ、 マスター／スレーブ	○	数秒	△	公開されているSlony-Iの運用ノウハウが少ない。バージョン混在可。
pgpool-II	アクティブ／アクティブ、 マスター／スレーブ	○	なし	○	pgpoolサーバがSPOF(冗長化可)。 一部、APへの影響有り(now() 等)。
ストリーミング・レプリケーション(9.0～)	アクティブ／アクティブ、 マスター／スレーブ	○	数百ms～ なし(9.1)	△	公開されている運用ノウハウが少ない。遅延なしは9.1以降。

冗長化方式の選定 cont'd

OSS-DB

■PostgreSQLの代表的な冗長化方式の構成は以下の通り。

- シンプルな冗長化のみで良い場合は共有ディスク方式。
- スケールアウトが必要な場合は pgpool か Slony-I。
- 9.0以降はストリーミングレプリケーション(SR+HS)構成が可能。



ストリーミング・レプリケーション

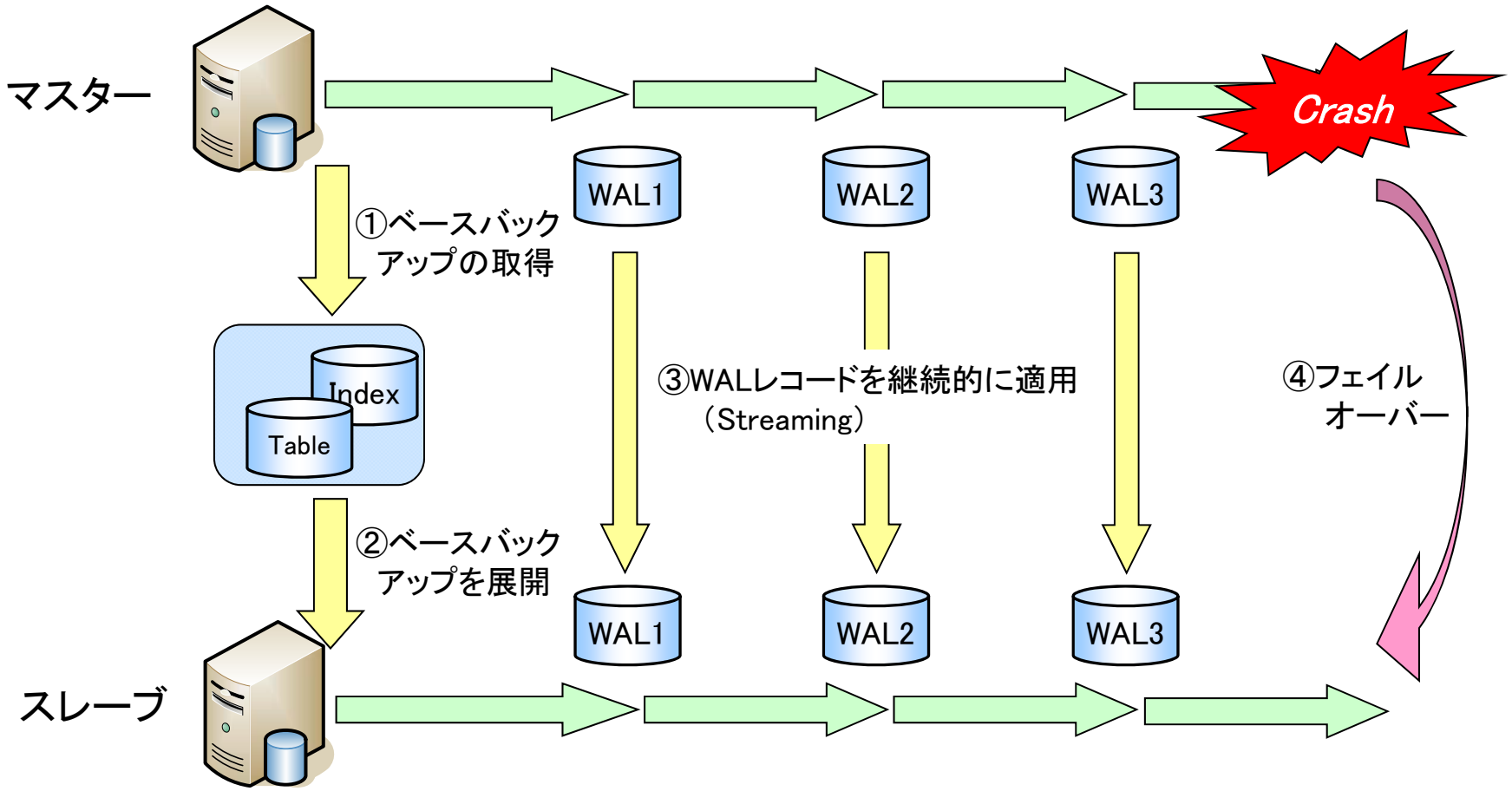
OSS-DB

- PostgreSQL 9.0で標準実装されるレプリケーション機能は、「ストリーミング・レプリケーション（SR）」と呼ばれています。
- ストリーミングレプリケーションは、1つのマスターノード（読み書き可能）と、1つ以上のスレーブノード（読み込みのみ）で構成される、シングルマスタ・マルチスレーブ構成です。
- マスターノードは、マスターノード上で生成された更新情報（トランザクションログ）をスレーブノードに転送します（ストリーミング）。
- スレーブノードは、ログレコードを待ち受け、受信したものを自身のノードのWALファイルに適用していきます。
- スレーブノードでは、リードオンリーのクエリを処理することができます（ホットスタンバイモード）。

25.2. ログシッピングスタンバイサーバ
http://www.postgresql.jp/document/9.0/html/warm-standby.html

ストリーミング・レプリケーション概念図

■ベースバックアップで「基準」を揃え、WALレコードを連続的に転送（Streaming）することで「差分」を埋める。



ストリーミング・レプリケーション導入手順

OSS-DB

- ステップ１：データベースクラスタを初期化&複製
- ステップ２：マスターノードの設定
- ステップ３：スレーブノードの設定
- ステップ４：各ノードの起動&動作確認

ステップ1:データベースクラスタを初期化&複製

OSS-DB

■データベースクラスタを初期化する
- master$ initdb -D $PGDATA --no-locale --encoding=UTF8

■アーカイブログモードを有効にする(postgresql.conf)
- archive_mode = on
- archive_command = 'cp %p /var/lib/pgsql/data/pg_xlogarch/%f'

■ベースバックアップを取得する
- master$ pg_ctl -D $PGDATA start
- master$ psql -c "SELECT pg_start_backup ('initial backup for SR')" template1
- master$ tar cvf pg_base_backup.tar $PGDATA
- master$ psql -c "SELECT pg_stop_backup ()" template1

■スレーブノードにベースバックアップを展開する
- slave$ tar xvf pg_base_backup.tar
- slave$ rm -f $PGDATA/postmaster.pid

ステップ2:マスターノードの設定

OSS-DB

- WALをスレーブに送信できるように設定(postgresql.conf)。
  - listen_addresses = '*'
  - wal_level = hot_standby
  - max_wal_senders = 5
  - wal_keep_segments = 32

- スレーブノードからの接続を受け付けられるように設定(pg_hba.conf)
  - host *replication* all 10.0.2.42/32 trust

ステップ3：スレーブノードの設定

OSS-DB

■スタンバイモードとして設定（postgresql.conf）

- hot_standby = on

■スタンバイ用の設定ファイルの作成（recovery.conf）

- standby_mode = 'on'
- primary_conninfo = 'host=10.0.2.41 port=5432 user=snaga'
- trigger_file = '/var/lib/pgsql/data/pg_failover_trigger'
- restore_command = 'cp /var/lib/pgsql/data/pg_xlogarch/%f "%p"'

ステップ4：各ノードの起動&動作確認

OSS-DB

- マスターサーバを起動し、スレーブサーバを起動。
- マスターノードでは以下のようなログが見られる。
  - LOG: replication connection authorized: user=snaga host=10.0.2.42 port=55811
- スレーブノードでは以下のようなログが見られる。
  - LOG: streaming replication successfully connected to primary
- マスターノード上でレコードを更新し、スレーブノードで参照できれば設定は完了。
- 実際には、これらの手順以外に障害発生時のフェイルオーバーの実装が必要になります。

参考文献

OSS-DB

書籍・雑誌

- WEB+DB PRESS vol.24、25「徒然PostgreSQL散策」（技術評論社）
- WEB+DB PRESS vol.32～37「PostgreSQL安定運用のコツ」（技術評論社）
- WEB+DB PRESS vol.63「Web開発の『べし』『べからず』」（技術評論社）
- PostgreSQL徹底入門 第3版（翔泳社）
- データベースパフォーマンスアップの教科書 基本原理編（翔泳社）

オンラインドキュメント類

- PostgreSQL 9.0.4文書
  http://www.postgresql.jp/document/9.0/html/index.html
- Explaining Explain ～ PostgreSQLの実行計画を読む ～ (PDF版)
  http://lets.postgresql.jp/documents/technical/query_tuning/explaining_explain_ja.pdf
- HOTの仕組み（1） - Let's Postgres
  http://lets.postgresql.jp/documents/tutorial/hot_2/
- PostgreSQLのチューニング技法 - しくみを知って賢く使う-
  http://www.postgresql.jp/events/pgcon09j/doc/b2-3.pdf
- スロークエリの分析 - Let's Postgres
  http://lets.postgresql.jp/documents/technical/query_analysis
- ソーシャルゲームのためのデータベース設計
  http://www.slideshare.net/matsunobu/ss-6584540
- 高信頼システム構築標準教科書 －仮想化と高可用性－
  http://www.lpi.or.jp/linuxtext/system.shtml
- MVCC in PostgreSQL
  http://chesnok.com/talks/mvcc_couchcamp.pdf
- Query Execution Techniques in PostgreSQL
  http://neilconway.org/talks/executor.pdf
- Robert Haas: Index-Only Scans
  http://rhaas.blogspot.com/2010/11/index-only-scans.html

ご清聴ありがとうございました。

■お問い合わせ■
アップタイム・テクノロジーズ合同会社
永安 悟史
snaga@uptime.jp

