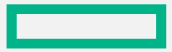


Hewlett Packard
Enterprise

検知できない破壊の話

Noriyoshi Shinoda

July 30, 2020

SPEAKER
篠田典良（しのだのりよし）

- 所属
  - 日本ヒューレット・パッカード株式会社
- 現在の業務
  - PostgreSQL をはじめ、Oracle Database, Microsoft SQL Server, Vertica 等 RDBMS 全般に関するシステムの設計、移行、チューニング、コンサルティング
  - Oracle ACE
  - Oracle Database 関連書籍 15 冊の執筆
  - オープンソース製品に関する調査、検証
- 関連する URL
  - 「PostgreSQL 虎の巻」シリーズ
    - http://h30507.www3.hp.com/t5/user/viewprofilepage/user-id/838802
  - Oracle ACE ってどんな人？
    - http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-ja.html

はじめに
ネガティブなお話しです

– クラッシュ後のデータベースはクラッシュ・リカバリにより復旧する

– クラッシュ時に保持されない情報や、ファイル削除が検知できない場合がある。←今日の話

検知できる破壊
データ破壊の検知は年々進化

- ブロック破壊についてはページ・チェックサム機能が進化
  - PostgreSQL 9.3 ではデータベース・クラスタ作成時にチェックサムの有効化が可能に
  - PostgreSQL 11 では pg_verify_checksums コマンドによるチェック機能提供
  - PostgreSQL 12 では pg_checksums コマンドでチェックサム有効／無効を切り替えられるように
- バックアップ
  - PostgreSQL 13 で整合性のチェック機能、pg_verifybackup コマンドの提供
- Read/Write エラーで PANIC になるか、インスタンス起動不可になるファイル
  - pg_control ファイル
  - カレント WAL ファイル
- 自動再作成されるファイル
  - Free Space Map
  - Visibility Map

ログファイル
OSクラッシュ時の書き込みデータ

- loggerプロセスが実行
- ファイルのオープン時
  - O_DIRECT や O_SYNC 等のフラグは指定されていない
- ログ書き込み時にはフラッシュしていない
  - OS クラッシュ時には書き込まれていない可能性
- ログローテーション時のシステムコール

```
open("log/postgresql-2020-07-28_060526.log", O_WRONLY|O_CREAT|O_APPEND, 0666) = 4
lseek(4, 0, SEEK_END)                       = 0
write(4, "2020-07-28 06:05:33.241 JST [265"..., 91) = 91
write(4, "2020-07-28 06:05:33.241 JST [265"..., 55) = 55
lseek(4, 0, SEEK_CUR)                       = 146
```

ログファイル
OSクラッシュ時の書き込みデータ

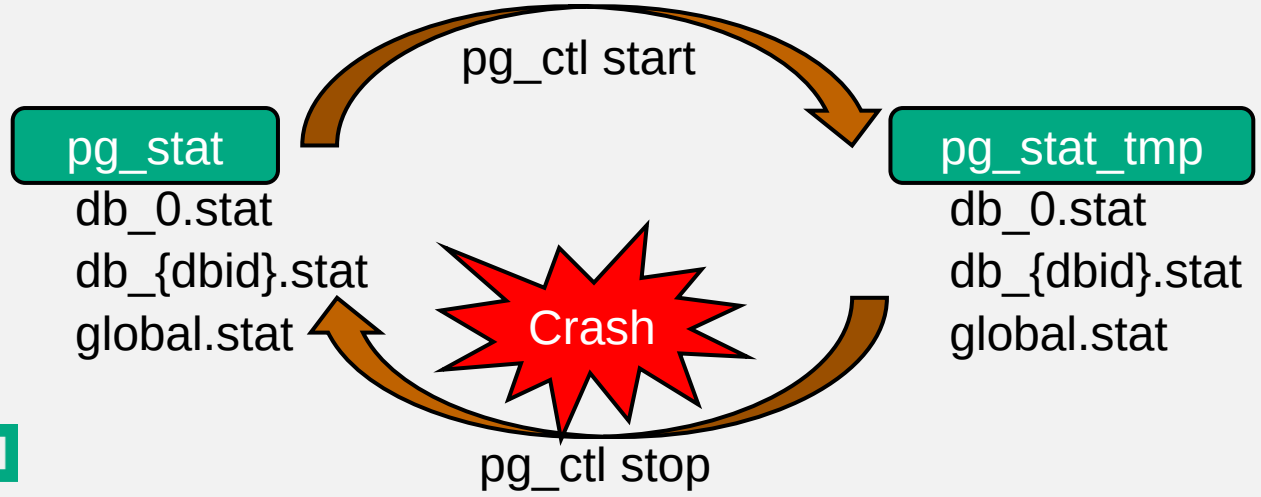
- PostgreSQL 13 の Extension adminpack
  - pg_file_sync 関数が追加された
- 実行例

```
postgres=# CREATE EXTENSION adminpack;
CREATE EXTENSION
postgres=# SELECT pg_file_sync('log/postgresql-2020-07-28_120622.log');
 pg_file_sync
--------------


(1 row)
```

統計情報
インスタンス・クラッシュ後の統計情報

- pg_stat_{all|user|sys}_{tables|indexes} ビュー
- stats collector プロセスが実行
- インスタンス起動時にファイル移動
  - ファイルを pg_stat ディレクトリから pg_stat_tmp ディレクトリ（GUC: stats_temp_directory）へ
- シャットダウン時に戻す
  - OS がクラッシュするとデータ移動が行われない（インスタンス・クラッシュでは問題ない場合も）
  - 次回起動時に pg_stat_tmp ディレクトリのファイルは削除されて初期化



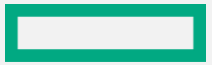
統計情報
インスタンス・クラッシュ後の統計情報

- マニュアルの記述（stats_temp_directory）
  - 「これをRAMベースのファイルシステムを指し示すようにすることで物理I/O要求が減り、性能を向上させることができます。」 ⇒ 元々永続化にはこだわっていない？
- Extension pg_stat_statements は同じ仕組みを使っている
  - pg_stat/pg_stat_statements.stat ファイル ⇔ pg_stat_tmp/pgss_query_texts.stat ファイル
  - インスタンス・クラッシュ時には pg_stat_statements ビューの内容が消える

セグメントファイル
1 GB超のファイル

- レコード数 = 1億件／テーブルサイズ = 約 4 GB のテーブルを準備
  - 1 GB を超えるテーブルはファイル分割される

```
postgres=> SELECT COUNT(*), pg_relation_size('data1'), pg_relation_filepath('data1') FROM data1;
   count    | pg_relation_size | pg_relation_filepath
------------+------------------+----------------------
 100000000 |       4428111872 | base/13578/16557
(1 row)
postgres=> ¥! ls -l base/13578/16557*
-rw-------. 1 postgres postgres 1073741824 Jul 28 01:41 base/13578/16557
-rw-------. 1 postgres postgres 1073741824 Jul 28 01:42 base/13578/16557.1
-rw-------. 1 postgres postgres 1073741824 Jul 28 01:43 base/13578/16557.2
-rw-------. 1 postgres postgres 1073741824 Jul 28 01:44 base/13578/16557.3
-rw-------. 1 postgres postgres  133144576 Jul 28 01:48 base/13578/16557.4
-rw-------. 1 postgres postgres    1105920 Jul 28 01:48 base/13578/16557_fsm
-rw-------. 1 postgres postgres     139264 Jul 28 01:48 base/13578/16557_vm
```



セグメントファイル
1 GB超のファイル

– インスタンス停止後に 16557.1 ファイルを削除し、インスタンス再起動

Index Only Scan

```
postgres=> EXPLAIN SELECT COUNT(c1) FROM data1;
                        QUERY PLAN
-----------------------------------------------------------
Aggregate  (cost=1521122.18..1521122.19 rows= …
   ->  Index Only Scan using data1_pkey on …
(2 rows)
postgres=> SELECT COUNT(c1) FROM data1;
  count
-----------
 100000000
(1 row)
```

Seq Scan

```
postgres=> EXPLAIN SELECT COUNT(c1) FROM data1;
                        QUERY PLAN
-----------------------------------------------------------
Aggregate  (cost=434176.01..434176.02 rows=1 …
   ->  Seq Scan on data1  (cost=0.00..343555 …
(2 rows)
postgres=> SELECT COUNT(c1) FROM data1;
  count
----------
 24248320
(1 row)
```

– 実行計画によって SQL 文の結果が異なる

セグメントファイル
1 GB 超のファイル

- Index Scan ではエラーになる可能性がある

```
postgres=> EXPLAIN SELECT * FROM data1 WHERE c1=30000000;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using idx1_data1 on data1  (cost=0.56..8.58 rows=1 width=12)
   Index Cond: (c1 = '30000000'::numeric)
(2 rows)
postgres=> SELECT * FROM data1 WHERE c1=30000000;
ERROR:  could not open file "base/13578/16557.1" (target block 162162): No such file or directory
```

- pg_relation_size 関数は 1GB を返す

```
postgres=> SELECT pg_relation_size('data1');
pg_relation_size
------------------
       1073741824
(1 row)
```

テーブルファイル
更新があったテーブル

- CHECKPOINT 後に更新（WAL）があったテーブル
- データの準備

```
postgres=> SELECT COUNT(*), pg_relation_size('data2'), pg_relation_filepath('data2') FROM data2;
  count   | pg_relation_size | pg_relation_filepath
----------+------------------+----------------------
 10000000 |        442818560 | base/13578/16561
(1 row)
```

- テーブル更新直後にクラッシュし、ファイルが消えた

```
postgres=> INSERT INTO data2 VALUES (generate_series(1, 10000), 'data2');
INSERT 0 10000
```

テーブルファイル
更新があったテーブル

- 再起動後
  - Seq Scan ではエラーにならない

```
postgres=> SELECT COUNT(*) FROM data2;
 count
-------
 10010
(1 row)
```

  - Index Scan でもエラーにならない

```
postgres=> EXPLAIN ANALYZE SELECT * FROM data1 WHERE c1=100000;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------…
 Index Scan using idx1_data1 on data1  (cost=0.57..8.59 rows=1 width=12) (actual time=46.701..46…
   Index Cond: (c1 = '100000'::numeric)
 Planning Time: 20.413 ms
 Execution Time: 46.784 ms
(4 rows)
```

テーブルファイル
更新があったテーブル

– ログにもエラーは出ない

```
$ tail postgresql-2020-07-28_023916.log
LOG:  starting PostgreSQL 14devel on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-28), 64-bit
LOG:  listening on IPv6 address "::1", port 5432
LOG:  listening on IPv4 address "127.0.0.1", port 5432
LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
LOG:  database system was interrupted; last known up at 2020-07-28 11:13:38 JST
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:  redo starts at 0/542521A8
LOG:  invalid record length at 0/543CDAD0: wanted 24, got 0
LOG:  redo done at 0/543CDA98
LOG:  database system is ready to accept connections
```

テーブルファイル
更新があったテーブル

– インデックスや TOAST ファイルが削除された場合はエラーになる

```
postgres=> SELECT COUNT(*) FROM data1 WHERE c1=100;
ERROR:  index "idx1_data1" contains unexpected zero page at block 0
HINT:  Please REINDEX it.

postgres=> SELECT c2 FROM data1 WHERE c1=1;
ERROR:  missing chunk number 0 for toast value 16822 in pg_toast_16774
```

WAL ファイル
最新の WAL ファイル削除

– 全タプルを削除
  – タプル削除後に OS クラッシュし、最新の WAL ファイルのみが削除された

```
postgres=> SELECT COUNT(*) FROM data2;
  count
----------
 10000000
(1 row)
postgres=> DELETE FROM data2;
DELETE 10000000
postgres=> SELECT pg_walfile_name(pg_current_wal_lsn());
 pg_walfile_name
--------------------------
 000000010000000010000003E
(1 row)
```

WAL ファイル
最新の WAL ファイル削除

- クラッシュ後の再起動ログ
  - クラッシュリカバリが行われたことだけ出力される

```
LOG:  database system was interrupted; last known up at 2020-07-28 11:33:40 JST
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:  redo starts at 0/5C8D8458
LOG:  redo done at 0/67FFFFA8
LOG:  database system is ready to accept connections
```

- 検索

```
postgres=> SELECT COUNT(*) FROM data2;
  count
----------
 10000000
(1 row)
```

THANK YOU

Mail: noriyoshi.shinoda@hpe.com
Twitter: @nori_shinoda