PGECons
PostgreSQL Enterprise Consortium

2013年活動報告書
Appendix 2 バックアップ検証（シングルサーバ編）

PostgreSQLエンタープライズ・コンソーシアム
WG3（設計運用WG）

改訂履歴

版	改訂日	変更内容
1.0	2014/4/25	新規作成

はじめに

- 本検証はシングル構成でPostgreSQLを運用中に障害が発生し、それを復旧する手順等の実機検証です

- 検証するシナリオは以下の3つです
  - データが壊れたとき、論理バックアップからリストアするシナリオ
  - オペレーションミスでデータを削除した場合、削除されたデータをPoint In Time Recoveryでリカバリするシナリオ
  - データ更新中のサーバの電源が落ちたとき、再起動しWALから自動ロールフォワードするシナリオ

目次

- クラッシュリカバリ
- 論理バックアップ／リストア
- オンラインバックアップ／リカバリ
- 付録:ストレージローカルコピー

検証環境

- 環境
  - インストールディレクトリ：/usr/local/pgsql/
  - データディレクトリ：/disk1/data
  - WALディレクトリ：/disk2/pg_xlog
  - ARCHIVEファイル配置ディレクトリ：/disk3/pg_xlog
  - ユーザ名：postgres
  - PostgreSQL：PostgreSQL 9.3.0
  - OS：Red Hat Enterprise Linux Server 6.2 (64-bit x86)
  - 環境変数
    - $PATH：/usr/local/pgsql/bin を追加
    - $PGDATA：/disk1/data

クラッシュリカバリ

シナリオ（1／2）

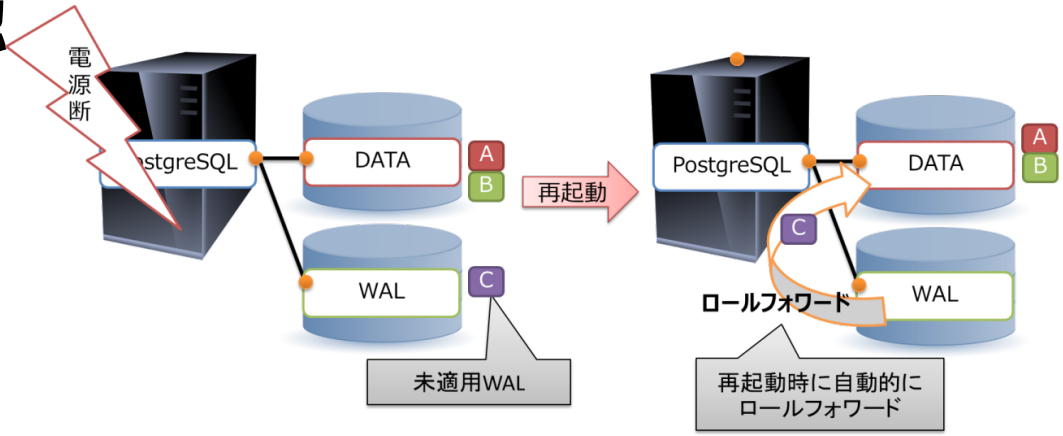
- システム構成
  - シングル構成

- システム概要
  - 更新処理中のシステム

- バックアップ
  - 週に一度、休日にバックアップ

シナリオ(2/2)

- 検証の流れ
  - CHECKPOINTの間隔を調整
    - checkpoint_segments = 1000
    - checkpoint_timeout = 1h
  - データ更新中に電源断を実施
    - 後で確認できるように決まったデータを更新
  - サーバ再起動
  - PostgreSQL再起動
  - 更新したデータを確認



検証結果

- 結果
  - 再起動時ロールフォワードして最終トランザクションまでのデータが反映されている。
    ⇒ 電源断のような障害のときデータが復旧される

```
 [postgres@pgecons1 pg_log]$ cat postgresql-2013-10-10_142041.log
[2013-10-10 14:20:41 JST]LOG:  database system was interrupted; last known up at 2013-10-10 12:36:17 JST
[2013-10-10 14:20:41 JST]LOG:  database system was not properly shut down; automatic recovery in progress
[2013-10-10 14:20:41 JST]LOG:  redo starts at 135/7B000090
[2013-10-10 14:20:52 JST]LOG:  redo done at 135/D7EED2C8
[2013-10-10 14:20:52 JST]LOG:  last completed transaction was at log time 2013-10-10 14:11:59.994082+09

testdb=# select * from pgbench_history order by mtime desc;
  tid   |  bid  |    aid    | delta |           mtime            | filler
--------+-------+-----------+-------+----------------------------+--------
  58045 |  7647 | 521189773 |  1435 | 2013-10-10 14:11:59.993444 |
  93451 |  9583 | 409026996 | -4937 | 2013-10-10 14:11:59.992281 |
  63657 |  9706 | 494699262 |  2752 | 2013-10-10 14:11:59.990973 |
  43897 |   461 | 239405160 |  2752 | 2013-10-10 14:11:59.989828 |
  20564 |  3888 | 764911388 | -3846 | 2013-10-10 14:11:59.989503 |
  46136 |  3762 | 945423791 |  3273 | 2013-10-10 14:11:59.988407 |
  60703 |  4469 | 858075989 | -2745 | 2013-10-10 14:11:59.988401 |
```

論理バックアップ／リストア

シナリオ（1/2）

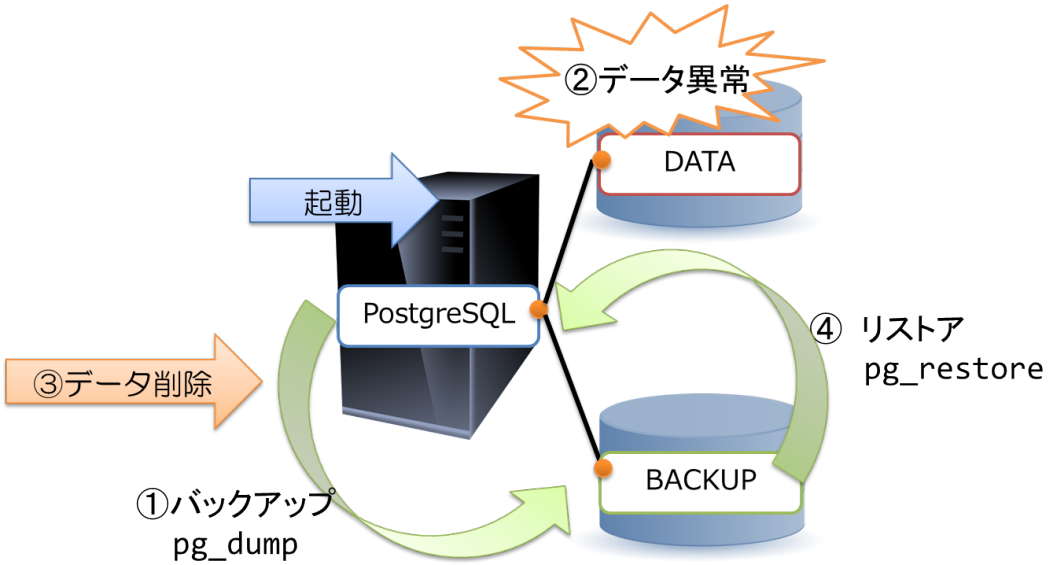
- システム構成
  - シングル構成
- システム概要
  - 参照系システム
  - 更新は土日のバッチ処理のみ
- バックアップ
  - 毎週更新処理完了後pg_dumpを利用して全体の論理バックアップを取得

シナリオ (2/2)

- 検証のイメージ
  - ある週の火曜日から、時々エラーが発生するようになった
  - データが壊れた可能性が高いが壊れたデータの特定ができてない
  - カスタム形式でバックアップを取得している
  - 正しかった時点のデータをリストアしたい

検証手順概要

- pgbenchで初期データ投入
- バックアップ取得
  - pg_dump
- データに異常が発生したと仮定し、データ削除
  - dropdbでデータをデータベースごと削除
  - createdbで再生性
- リストア
  - pg_restore



初期データ

- pgbenchで初期データを投入

```
[postgres@server ~]$ pgbench -is 1000 testdb
creating tables...
100000 of 1000000 tuples (10%) done (elapsed 0.19 s, remaining 1.71 s).
200000 of 1000000 tuples (20%) done (elapsed 0.38 s, remaining 1.54 s).

      ⋮
```

バックアップ取得

- バックアップ

```
[postgres@server ~]$ pg_dump -Fc testdb > /disk4/backup/backup.dump
[postgres@server ~]$ ls /disk4/backup/ -lh
合計 272M
-rw-rw-r-- 1 postgres postgres 272M  3月 10 17:22 backup.dump
```

論理バックアップの特徴

- データは以下のようにデータベースを検索した結果を保存する
  - pg_dump実行時のスナップショットデータ
  - オプションで圧縮形式で取得するのが一般的
- pg_dumpはクライアントとしてデータを取得するので、取得ができればデータが壊れてない可能性が非常に高い
- ただし、バックアップ取得時点以後の更新分はリカバリできない

```
--
-- PostgreSQL database dump
--

SET statement_timeout = 0;
SET client_encoding = 'SQL_ASCII';
SET standard_conforming_strings = on;

・・・
584     1       0
585     1       0
586     1       0
\.
```

例）データが壊れている際にpg_dumpを取得した例
(プレーンテキストで取得した場合)

テーブルのデータファイルが壊れている場合、pg_dump時にエラーになる
もしも、この状態で物理バックアップをとっても、
データファイルの破損に気が付かない

```
pg_dump: Dumping the contents of table "pgbench_accounts" failed: PQgetCopyData() failed.
pg_dump: Error message from server: pg_dump: The command was: COPY public.pgbench_accounts (aid, bid,
abalance, filler) TO stdout;
```

リストア

リストア

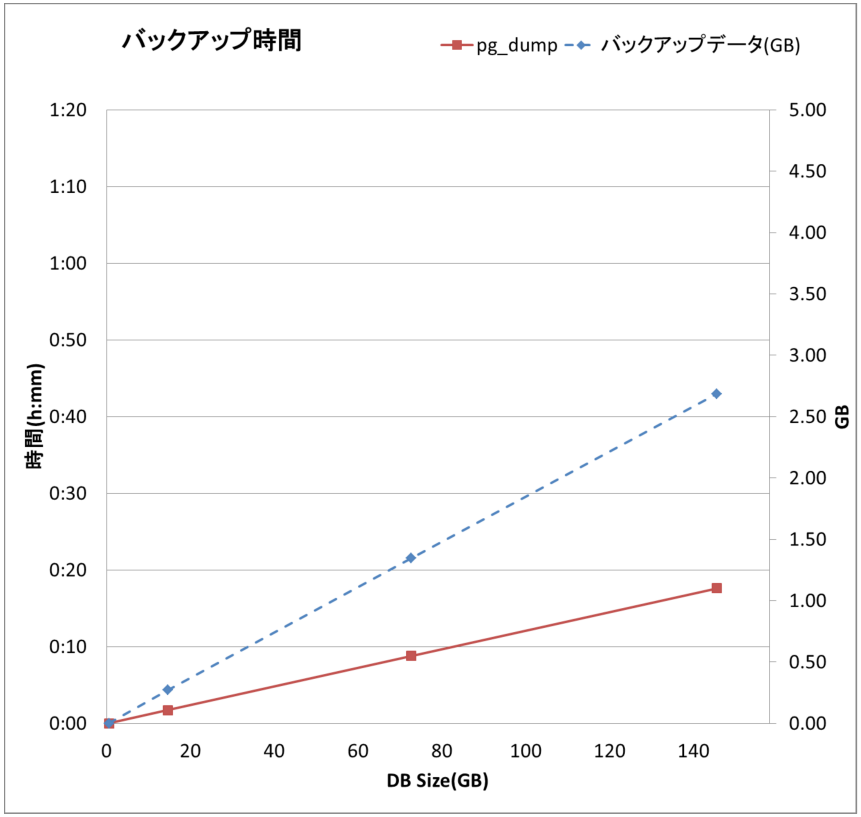
```
[postgres@server ~]$ dropdb testdb
[postgres@server ~]$ createdb testdb
[postgres@server ~]$ pg_restore -d testdb /disk4/backup/backup.dump
```

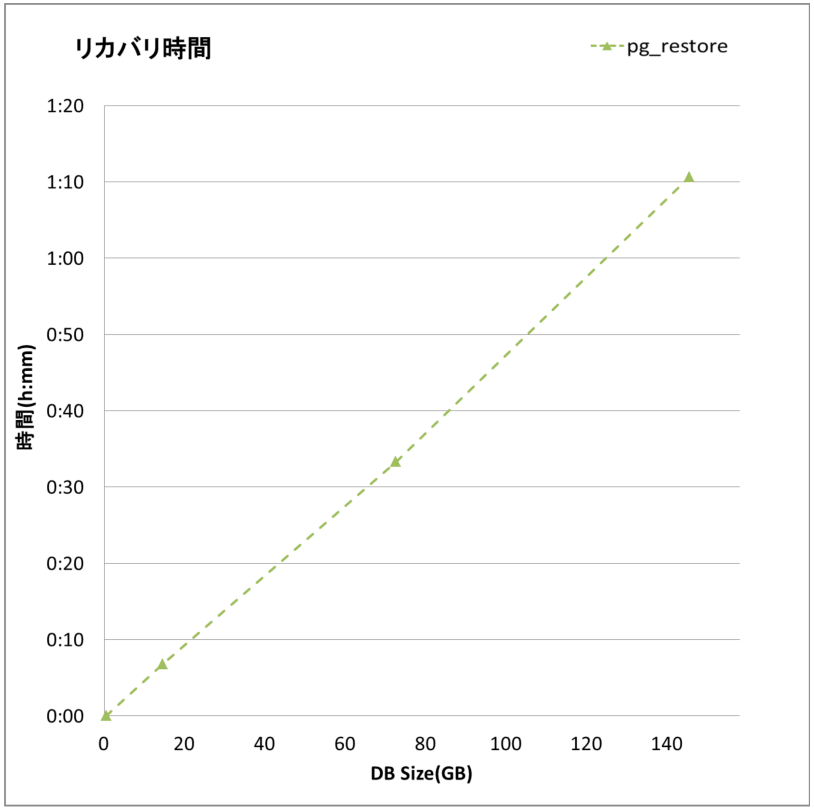
リストアの確認

```
testdb=# \d
              List of relations
 Schema |       Name       | Type  |  Owner
--------+------------------+-------+----------
 public | pgbench_accounts | table | postgres
 public | pgbench_branches | table | postgres
 public | pgbench_history  | table | postgres
 public | pgbench_tellers  | table | postgres
(4 rows)
testdb=# select * from pgbench_accounts;
   aid    | bid | abalance |                                                  filler
----------+-----+----------
+----------------------------------------------------------------------------------
--------
        1 |   1 |        0 |
        2 |   1 |        0 |
          :
```

バックアップとリストア時間の目安について

- データサイズとバックアップ／リストア所要時間の例
  - データ量に比例して線形で増加するため見積もりは簡単
  - バックアップ時間よりもリストア時間の方が長い





オンラインバックアップ・リカバリ

シナリオ（1/2）

- システム構成
  - シングル構成

- システム概要
  - 更新系システム
  - 毎月新しいテーブルを生成しその月の売上を記録
  - 1年前のデータをtruncateで削除

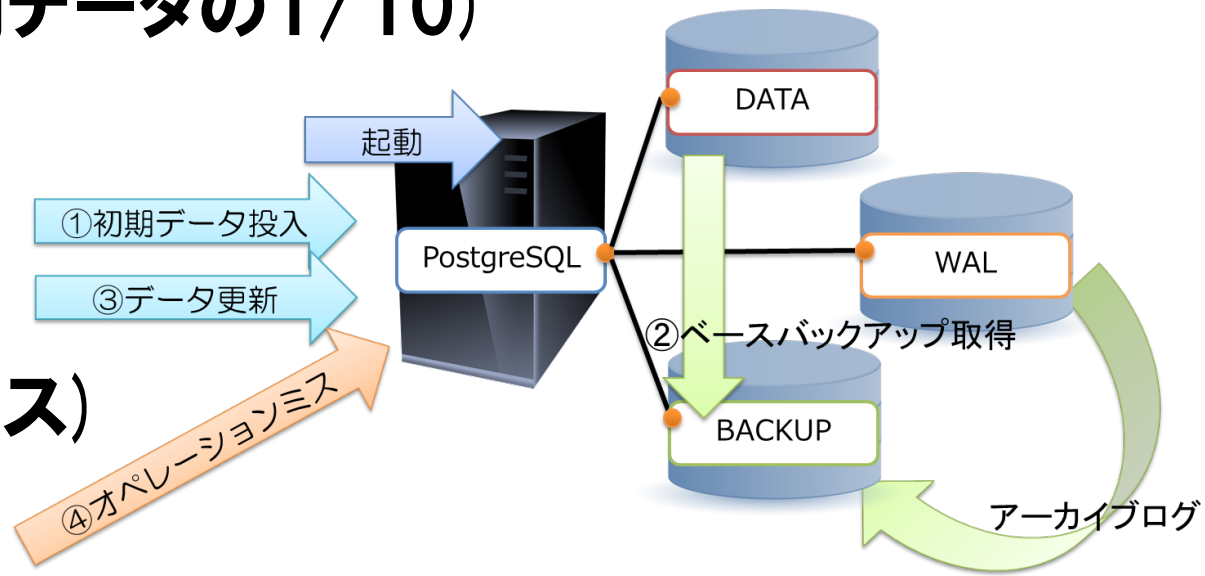
- バックアップ
  - PITRによる物理バックアップ
  - アーカイブモードはONで運用中

シナリオ (2/2)

- 検証のイメージ
  - オペレーションミスで今月の売上データをtruncateしてしまった
  - ベースバックアップ+アーカイブログの取得により運用中
  - truncateした時刻が分かっている、なるべく最新のデータに戻したい

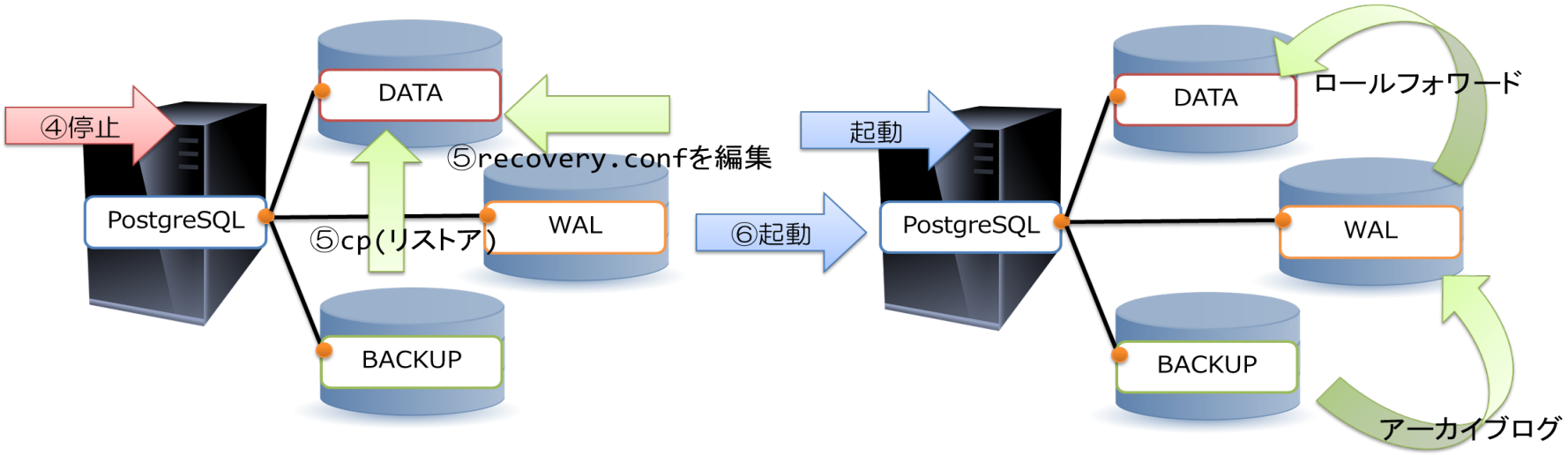
検証手順概要（1/2）

- 初期データ投入
- ベースバックアップ取得
  - pg_start_backup ()
  - tarでデータディレクトリを取得
  - pg_stop_backup ()
- データ更新（初期データの1/10）
- tableをtruncate（オペレーションミス）



検証手順概要(2/2)

- サーバ停止
- ベースバックアップからリストア
- recovery.confを作成してリカバリ時刻を指定
- サーバスタート



archive_mode設定

archive_mode設定

```
#postgresql.confの設定#
wal_level = archive         # minimalは使えない
archive_mode   = on
archive_command = 'test ! -f /disk3/archive/%f && cp %p /disk3/archive/%f'
#archiveはdataと違うディスクに保管したほうがいい
#testでチェックしてからarchiveする(上書きされないように)
#%p : archiveするファイルのパス名
#%f : archiveするファイルのファイル名
archive_timeout = 0       #時間で強制WAL切り替え(必須ではない) 0の場合強制切り替えなし
```

```
[postgres@cyprus data]$ pg_ctl restart
waiting for server to shut down....... done
server stopped
server starting
```

環境変数に$PGDATAを登録してない場合は-Dオプションで
データディレクトリを指定する必要がある

初期データ

- pgbenchで初期データを投入

```
[postgres@server ~]$ pgbench -is 1000 testdb
creating tables...
100000 of 1000000 tuples (10%) done (elapsed 0.19 s, remaining 1.71 s).
200000 of 1000000 tuples (20%) done (elapsed 0.38 s, remaining 1.54 s).

      ⋮
```

バックアップ取得

- pg_start_backupによるベースバックアップ

```
[postgres@server pgsql]$ psql testdb
testdb=# select pg_start_backup('basebackup.tar');
testdb=# \q

[postgres@server pgsql]$ tar cf /disk4/backup/basebackup.tar /disk1/data
[postgres@server pgsql]$ ll -h /disk4/backup/
合計 14.7G
-rw-rw-r-- 1 postgres postgres 14.7G  2月 20 11:27 basebackup.tar

[postgres@server pgsql]$ psql testdb
testdb=# select pg_stop_backup();
testdb=# \q
```

データ更新

データを更新

```
[postgres@server ~]$ pgbench -c 100 -j 10 -t 250 testdb
starting vacuum...end.
transaction type: TPC-B (sort of)
      ⋮
```

データ更新を確認

```
testdb=# select * from pgbench_history ;
 tid | bid |  aid  | delta |           mtime            | filler
-----+-----+-------+-------+----------------------------+--------
   7 |   1 |  3578 |  1839 | 2014-02-20 12:50:39.677176 |
   4 |   1 | 95175 | -4885 | 2014-02-20 12:50:39.690841 |
   6 |   1 | 80955 | -3258 | 2014-02-20 12:50:39.699467 |
   1 |   1 | 98961 |  -929 | 2014-02-20 12:50:39.707503 |
  10 |   1 | 14484 | -1023 | 2014-02-20 12:50:39.715925 |
   9 |   1 | 36463 |  4403 | 2014-02-20 12:50:39.724406 |
  10 |   1 | 72578 | -4197 | 2014-02-20 12:50:39.732842 |

       ⋮
```

障害発生

- オペレーションミスによるデータ削除

```
testdb=# truncate pgbench_history;
TRUNCATE TABLE
testdb=# select * from pgbench_history ;
 tid | bid | aid | delta | mtime | filler
-----+-----+-----+-------+-------+--------
(0 rows)
```

リカバリ

- サーバ停止後recovery.confを作成
- サーバを開始することでリカバリ

```
[postgres@server pgsql]$ pg_ctl stop
[postgres@server pgsql]$ mv /disk1/data /disk1/data_crash_backup
[postgres@server pgsql]$ tar xf /disk4/backup/basebackup.tar -C /disk1
[postgres@server pgsql]$ rm -rf /disk2/pg_xlog/*
[postgres@server pgsql]$ mkdir /disk2/pg_xlog/archive_status
```

環境変数に$PGDATAを登録してない場合は-Dオプションで
データディレクトリを指定する必要がある

```
#recovery.confの設定#
restore_command = 'cp /disk3/archive/base/log/%f %p'
recovery_target_time = 'YYYY-MM-DD hh:mm:ss.ff' #復旧したい時間を書く
```

```
[postgres@server pgsql]$ pg_ctl start
```

環境変数に$PGDATAを登録してない場合は-Dオプションで
データディレクトリを指定する必要がある

リカバリ結果の確認

- データが戻ったことを確認

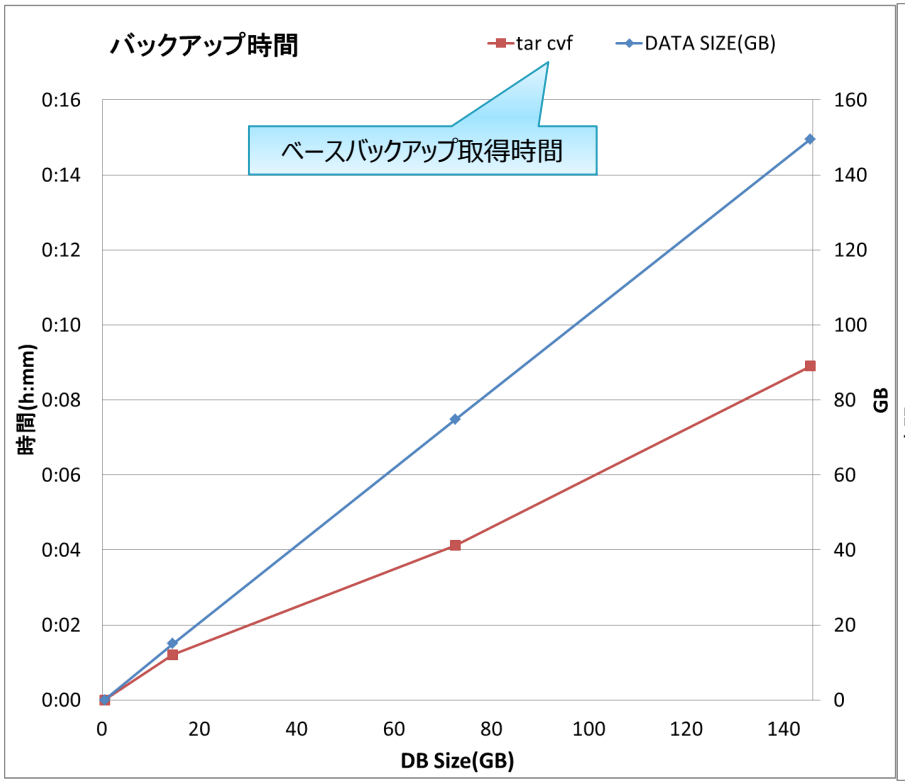
```
testdb=# select * from pgbench_history ;
 tid | bid |  aid  | delta |           mtime            | filler
-----+-----+-------+-------+----------------------------+--------
   7 |   1 |  3578 |  1839 | 2014-02-20 12:50:39.677176 |
   4 |   1 | 95175 | -4885 | 2014-02-20 12:50:39.690841 |
   6 |   1 | 80955 | -3258 | 2014-02-20 12:50:39.699467 |
   1 |   1 | 98961 |  -929 | 2014-02-20 12:50:39.707503 |
  10 |   1 | 14484 | -1023 | 2014-02-20 12:50:39.715925 |
   9 |   1 | 36463 |  4403 | 2014-02-20 12:50:39.724406 |
  10 |   1 | 72578 | -4197 | 2014-02-20 12:50:39.732842 |
       :
       :
```

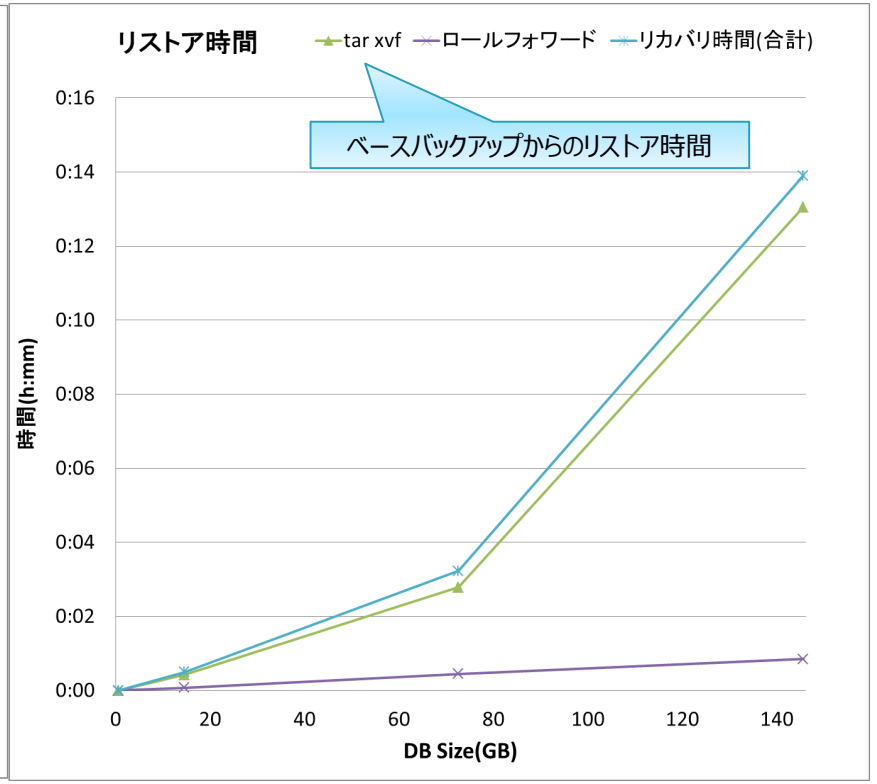
注意点

- postgresql.confファイルはWALで復旧しないので、変更があれば別途バックアップが必要
- 念のため元のデータは削除するより、mvで退避しておくことを推奨

バックアップとリカバリ時間の目安について

- データサイズとバックアップ・リカバリ所要時間の例
  - ロールフォワード時間は更新データ量に比例するため定期的に新しいベースバックアップを取得する必要がある
    - ロールフォワードするログの量はDBサイズの1/10で実施

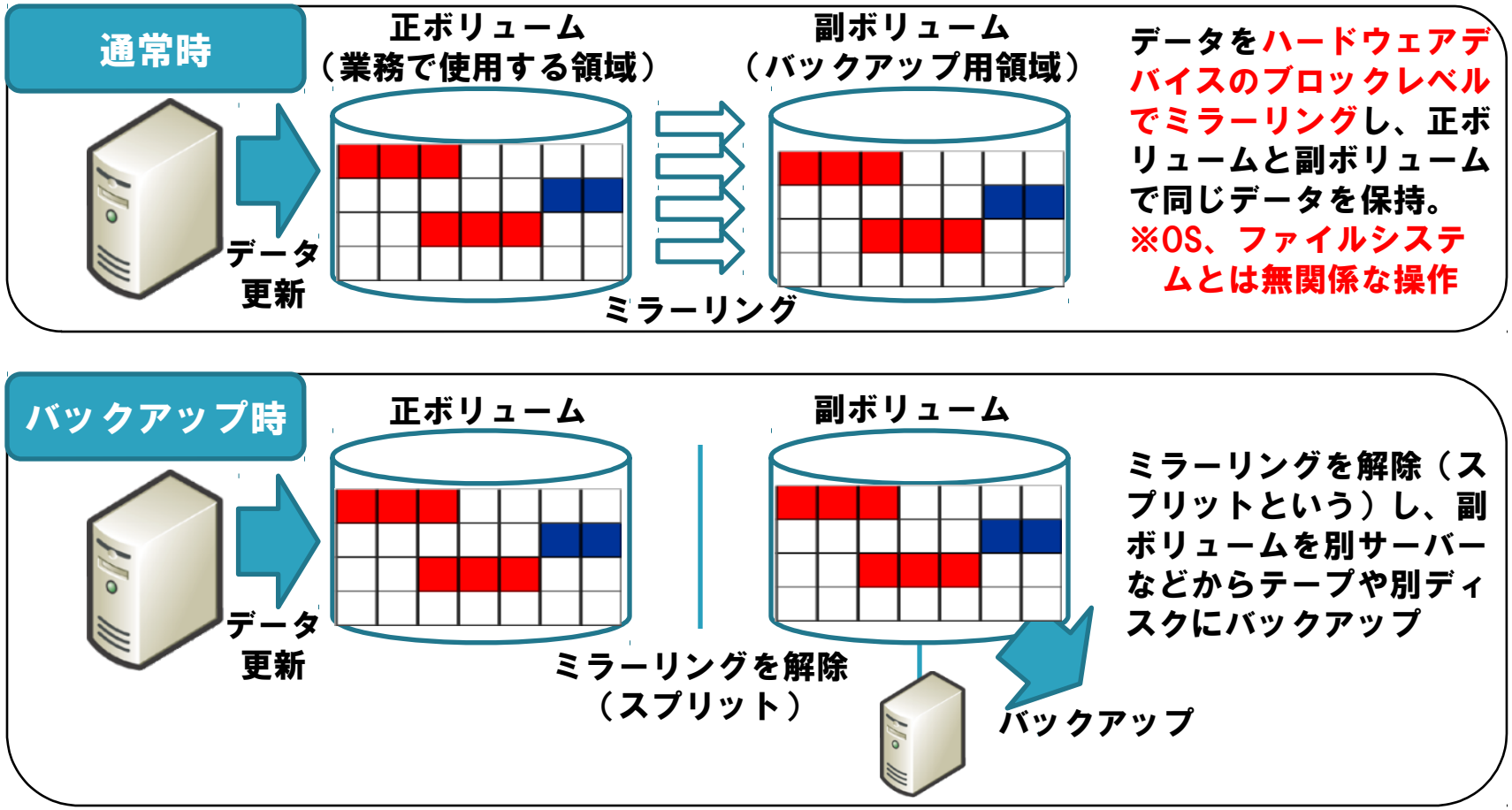




付録

ストレージローカルコピー

ストレージローカルコピーの仕組み



バックアップ完了後は再度ミラーリング状態に

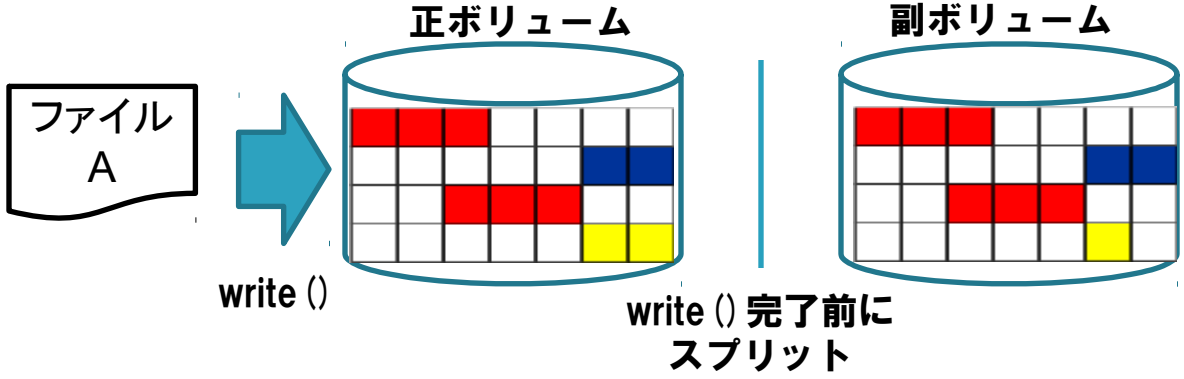
スプリットとリストアの注意点（1/2）

スプリット時にファイルに追記するなど、ファイルサイズが変更される処理中のファイルがあると、タイミングによってはファイルの不整合が発生する可能性があります。

例えばext4ファイルシステムで、ファイル拡張が発生するwrite()の場合、
- ファイルの実データ部分を更新（サイズ拡張）
- ファイルのメタデータ（サイズ情報）を更新

という順序で処理が行われます。

ストレージによるスプリットは、**OSやファイルシステムとは無関係に行われる**ため、非常にレアケースですが、①、②の間でスプリットが行われることがあります。

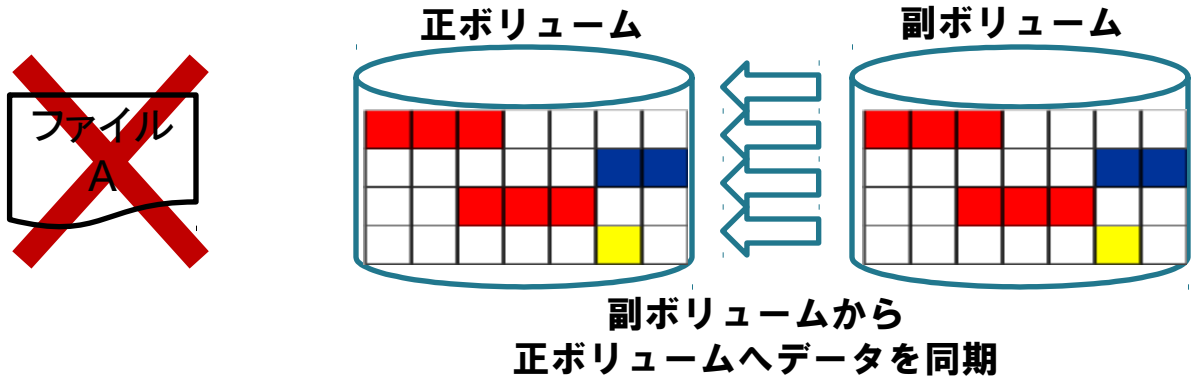


副ボリュームではファイルAのメタデータが更新されていない状態。
正ボリューム側はスプリット後にwrite()が完了し、正常となっている状態。

スプリットとリストアの注意点(2/2)

前頁の副ボリュームをバックアップしている状態で、正ボリュームに障害が発生したため、そのバックアップをリストアするケースを考えてみます。

正ボリュームのデータの内容はストレージの機能により、副ボリュームと同じ状態になります。



このとき、正ボリューム上のファイルAは実データとメタデータが不整合を起こしている状態ですので、正ボリュームをファイルシステムとしてマウントすると、fsckなどにより**ファイルAは削除**されることになります。

PostgreSQLのバックアップとの関連

PostgreSQLではバックアップ取得のために pg_start_backup () を実行しても、データファイルへの書き込みが停止するわけではありませんので、バックアップ中にデータファイルのサイズが拡張することも発生します。（PostgreSQLでは、ブロック（8KB）単位という小さな単位でファイルが拡張するため、拡張の頻度が高い。）
そのため、PostgreSQLのバックアップにストレージローカルコピーを利用する場合、pg_start_backup () を行っても、前述のとおり、副ボリューム中にファイルの実データとメタデータと不整合を起こしたデータファイルが存在する可能性があります。

その副ボリュームを正ボリュームにリストアした場合、不整合を起こしていたデータファイルは削除されてしまいます。データファイル自体が消失した場合はPostgreSQLの機能（たとえばWALなど）ではリカバリすることができません。

ファイルに書き込みがある状態で、ストレージローカルコピーをバックアップとして利用するには、このような危険性があります。この危険性を排除するためには、現在のPostgreSQLではファイルに書き込みが発生しない状態で、ストレージローカルコピーを行う必要があります。

（参考）他のデータベースでは

ストレージローカルコピーによりファイルの不整合が発生する主な原因は、スプリット時にファイルの拡張（あるいは縮小）が起こることにより、ファイルの実データ部分のサイズと、メタデータ上のサイズが一致しない状態になることです。

したがって、スプリット時にファイルの拡張や縮小が起こらなければ、ファイル自体への書き込みがあっても、不整合が発生する危険はほぼないものと考えられます。

そのため、
- 事前にデータファイルを必要なサイズで確保できる
- データファイルが自動的に拡張および縮小しないように設定できる

という特徴を持つデータベースでは、ファイルに書き込みがある状態でも、ストレージローカルコピーをバックアップとして利用することができると考えられます。