PGECons
PostgreSQL Enterprise Consortium

2013年活動報告書
Appendix 3 バックアップ検証（SR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

- 本検証はストリーミングレプリケーション(以下SR)を構成している環境で障害が発生し、それを復旧する手順等の実機検証です。
- 想定する環境については「2. SR環境での障害／復旧」に記述しています。
- 検証するシナリオは以下の2シナリオです。
  - 障害が発生しスレーブサーバをマスタに昇格して業務継続し、後日SRを再構成するシナリオ
  - 障害が発生し、バックアップからマスタをリカバリするシナリオ

※前者は手順中にバックアップを行いますが、リカバリを行わないため、可用性検証という見方もできます。

目次

1. 環境準備

(1) PostgreSQL 9.3環境構築

- インストールプラットフォームは RHEL6.2 x86_64
- ソースコードのダウンロード
  http://www.postgresql.org/ftp/source/v9.3.0/
  からダウンロード
- インストール準備

```
##インストールメディアをサーバの適当なディレクトリに配置
# cp postgresql-9.3.0.tar.bz2 /work/

##メディアを展開
# cd /work
# bzip2 -d postgresql-9.3.0.tar.bz2
# tar xvf postgresql-9.3.0.tar
```

1. 環境準備(続き)

(1) PostgreSQL 9.3環境(続き)

□ PostgreSQL環境

- 作業用ディレクトリ:/work
- PostgreSQL起動ユーザ:postgres
- インストールディレクトリ:/usr/local/pgsql(デフォルト)
- 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/archive
- postgresユーザの
  PATH環境変数に /usr/local/pgsql/bin を追加
  PGDATA環境変数に /disk1/data を設定

1. 環境準備(続き)

(1) PostgreSQL 9.3環境構築(続き)

□ PostgreSQL起動ユーザ作成

```
# # postgresユーザ作成（gid、uidは適宜指定する）
# groupadd postgres -g 501
# useradd postgres -u 501 -g 501
# chown postgres:postgres /work/postgresql-9.3.0
```

□ インストール

```
# # デフォルト設定でのインストール
# su - postgres
$ cd /work/postgresql-9.3.0
$ ./configure
$ gmake world
$ exit
# gmake install-world
```

pgbench等のcontribモジュールやドキュメントもインストールすることを想定し、「world」指定

1. 環境準備(続き)

(1) PostgreSQL 9.3環境構築(続き)

□ データベースクラスタ初期化

```
## データベース用ディレクトリの作成
# mkdir -p /disk1/data
# mkdir -p /disk2/pg_xlog
# mkdir -p /disk3/archive
# chown -R postgres:postgres /disk1
# chown -R postgres:postgres /disk2
# chown -R postgres:postgres /disk3
#
##データベースクラスタ初期化
# su - postgres
$ initdb -D /disk1/data -X /disk2/pg_xlog
```

PGDATA環境変数が設定されている場合、省略可能

1. 環境準備(続き)

(1) PostgreSQL 9.3環境構築(続き)

□ PostgreSQL起動および接続確認

```
$ vi  /disk1/data/postgresql.conf
以下を設定しファイルを保存
logging_collector = on

$ pg_ctl start -D /disk1/data
$ psql
```

PGDATA環境変数が設定されている場合、省略可能

1. 環境準備(続き)

(2) Streaming Replication環境構築

□ レプリケーション構成

マスタ1台-スレーブ1台の非同期レプリケーション構成

□ レプリケーション環境

IPアドレスは環境によって変える

- マスタ側IPアドレス:172.16.3.101
- スレーブ側IPアドレス:172.16.3.102
- レプリケーション用PostgreSQLユーザ:repuser
- ディレクトリ構成は両サーバとも同じとする。(「(1) PostgreSQL 9.3環境 PostgreSQL環境」を参照)

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

- スレーブ用PostgreSQLの準備
  スレーブサーバで(1)の手順の「インストール」までを行う。

- レプリケーションユーザ作成
  マスタサーバで以下を実施

```
# su - postgres
$
$ psql
postgres=#
postgres=# CREATE ROLE repuser LOGIN REPLICATION PASSWORD 'repuser';
```

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

- **レプリケーションユーザ接続設定**
  マスタサーバのpg_hba.confに以下を設定

[/disk1/data/pg_hba.conf]

```
host    replication     repuser         127.0.0.1/32            md5
host    replication     repuser         172.16.3.101/32         md5
host    replication     repuser         172.16.3.102/32         md5
```

pg_basebackup -h 127.0.0.1 に必要

- **初期データ投入**
  マスタサーバで以下を実施

```
$ createdb testdb
$ pgbench -i -s 10000 testdb
$ vacuumdb --all --analyze
```

スケールファクタ(-s)に10000を指定すると、データベースサイズが約150GBとなる。

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

□ マスタサーバパラメータ設定

マスタサーバのpostgresql.confに以下を設定

[/disk1/data/postgresql.conf]

```
#レプリケーションに必要な設定
listen_addresses = '*'
wal_level = hot_standby
archive_mode = on
archive_command = 'test ! -f /disk3/archive/%f && cp %p /disk3/archive/%f'
max_wal_senders = 2       # スレーブDBの数 + 1
```

その他に
shared_buffers,
checkpoint_segmentsなどを適宜変更する。
検証では
shared_buffers = 16GB
checkpoint_segments = 64
に設定

□ マスタサーバの再起動

```
$ pg_ctl restart
```

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

- スレーブサーバへ初期データ移行

  スレーブサーバで以下を実施

```
# mkdir /disk1/data
# mkdir /disk2/pg_xlog
# mkdir /disk3/archive
# chown -R postgres:postgres /disk1
# chown -R postgres:postgres /disk2
# chown -R postgres:postgres /disk3
# su - postgres
$ pg_basebackup -h 172.16.3.101 -U repuser -D /disk1/data --progress
 password:

$ rmdir /disk1/data/pg_xlog
$ cd /disk1/data
$ ln -s /disk2/pg_xlog pg_xlog
```

pg_basebackupでは pg_xlogディレクトリが dataの下に作られるため、ディレクトリを分ける構成の場合左記手順が必要。
pg_xlogディレクトリが dataの下にある構成の場合

```
$ pg_basebackup -h 172.16.3.101 -U repuser
-D /disk1/data --xlog --progress （実際は1行）
```

を実行し、後の処理は不要

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

- スレーブサーバ設定

  recovery.confを作成し、postgresql.confを修正

```
[/disk1/data/recovery.conf]
standby_mode = 'on'
primary_conninfo = 'host=172.16.3.101 port=5432 user=repuser password=repuser'
restore_command = 'scp /disk2/pg_xlog/%f "%p" 2> /dev/null'

[/disk1/data/postgresql.conf]
hot_standby = on
```

postgresユーザでサーバ間のscpがパスワードなしで実行できるように設定が必要。

手順については「5. ssh/scp設定」を参照。

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

- スレーブサーバでPostgreSQLを起動
  スレーブサーバで以下を実施

```
$ chmod 700 /disk1/data
$ pg_ctl start
```

- レプリケーションをテストするためのトランザクションを実行
  マスタサーバで以下を実施

```
$ pgbench -T 180 testdb
```

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

- レプリケーションの確認

  マスタサーバで以下を実施

```
$ psql -x -c "select * from pg_stat_replication"
-[ RECORD 1 ]----+------------------------------
pid              | 12328
usesysid         | 16384
usename          | repuser
application_name | walreceiver
client_addr      | 172.16.1.102
client_hostname  |
client_port      | 53736
backend_start    | 2013-09-20 16:24:32.960011+09
state            | streaming
sent_location    | 0/140000C8
write_location   | 0/140000C8
flush_location   | 0/140000C8
replay_location  | 0/140000C8
sync_priority    | 0
sync_state       | async
```

スレーブのアドレス

"streaming"であればOK

1. 環境準備(続き)

(2) Streaming Replication環境構築(続き)

- レプリケーションの確認(続き)

  スレーブサーバで以下を実施

```
$ psql -c "SELECT pg_last_xact_replay_timestamp()"
 pg_last_xact_replay_timestamp
-------------------------------
 2013-09-20 16:34:13.988285+09
(1 row)
```

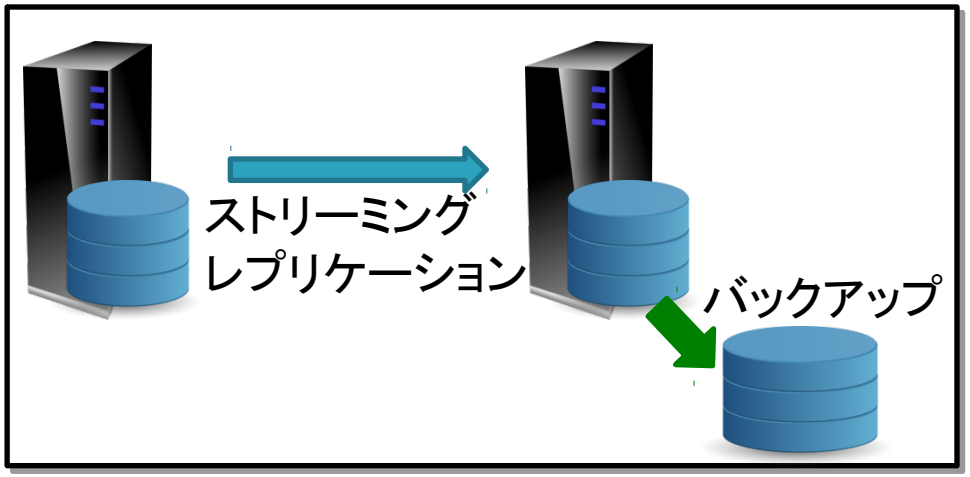
マスタからレプリケーションされ、
最後に適用されたWALの時刻

2. SR環境での障害／復旧

(1) 想定するシステム概要

- □ 非同期ストリーミングレプリケーション構成。
- □ 参照更新ともにあり。更新頻度は頻繁で、夜間もあり
- □ 通常時業務で使用するのはマスタのみ
- □ 毎日スレーブ側DBからバックアップ取得
- □ バックアップはオンラインで取得
- □ ユーザは100人程度
- □ 蓄積データは大量
- □ 監視あり
- □ 障害復旧は手動

同期レプリケーションにも対応できるように



2. SR環境での障害／復旧(続き)

(2) マスタサーバ障害の種類と復旧パターン

	障害状況	障害発生場所・種類	復旧方法
1	マスタサーバのOSが起動しない	・マスタサーバのハードウェア障害 ・OSのバグ etc	・スレーブをマスタに昇格させ業務継続。マスタは後日復旧
2	マスタサーバのPostgreSQLが起動しない(OSは起動)	・PostgreSQLバイナリ破損 ・WAL領域が物理的に破損 ・WAL領域FULL	・スレーブをマスタに昇格させ業務継続。マスタは後日復旧
3	マスタサーバのPostgreSQLでFatalエラー	・データ領域が物理的に破損 ・WALファイル破損 ・データファイル破損	・スレーブをマスタに昇格させ業務継続。マスタは後日復旧
4	データが論理的に破壊	オペレーションミスでデータを破壊	・バックアップからリカバリ

エラー原因、状況により
対応は変わることもある。

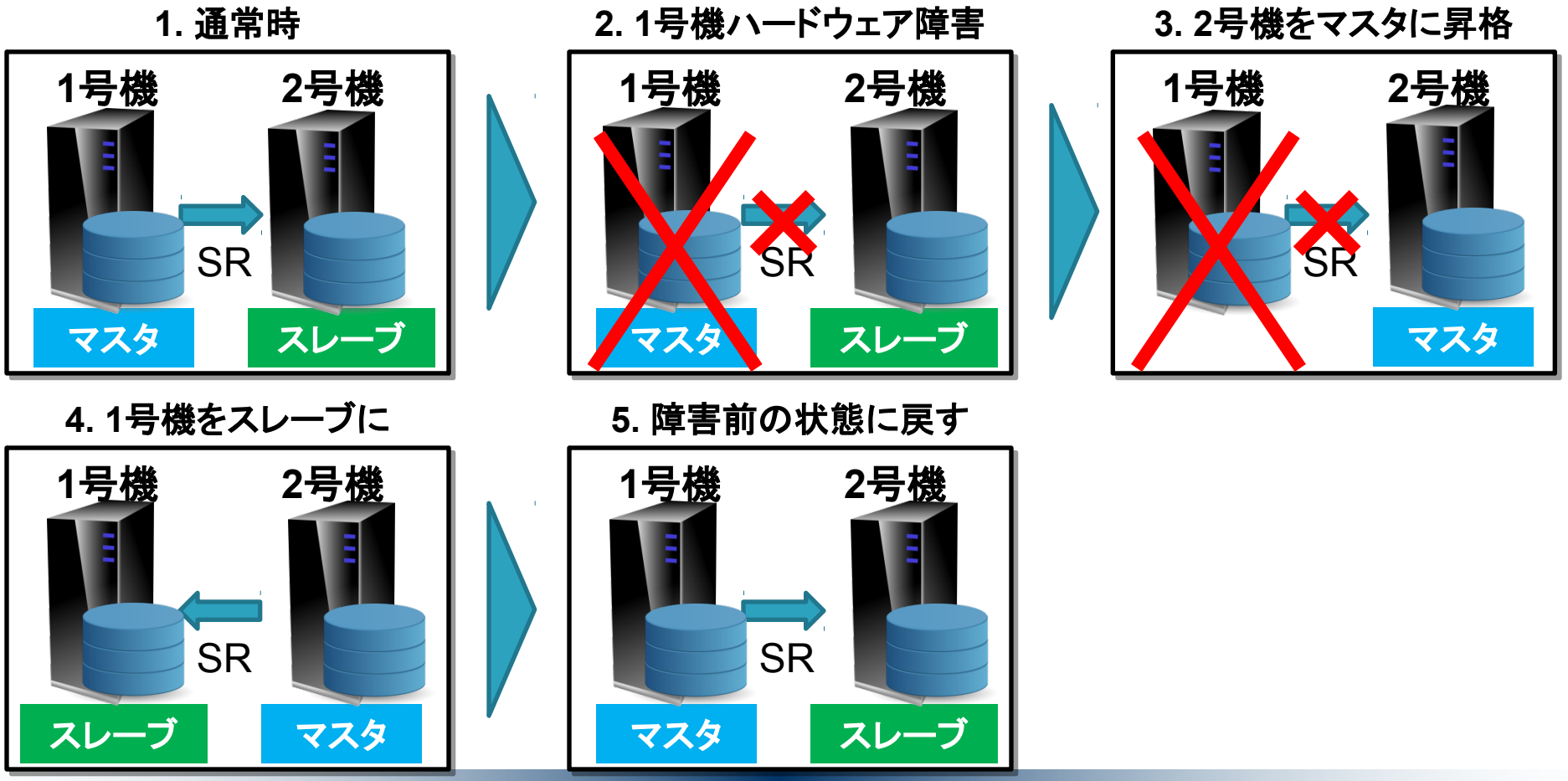
3. SR環境での障害／復旧シナリオ(１)
～スレーブサーバの昇格による業務継続～

(1) シナリオ概要

- マスタサーバのハードウェア障害が発生し、マスタダウン。
- 監視スクリプトにより障害検知。
- 障害状況を調査・分析の結果、マスタの復旧は困難と判断し、スレーブをマスタに昇格(フェイルオーバー)させ業務継続する。
- 新マスタ(旧スレーブ)のバックアップ取得
- バックアップから旧マスタを新スレーブとするSR構成を構築
- 新スレーブをマスタに昇格(スイッチバック)させ、障害前の構成に戻す。

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(1) シナリオ概要(続き) 状態遷移イメージ



3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順

□ 環境準備

9.3新機能のpg_isreadyを使用

- **監視スクリプト**
  本来であれば別途監視サーバから監視を行うが、サーバが2台のため、以下のように自己監視および相互監視を行う。
  ・マスタ側 (postgresユーザで実行)

```
$ while true; do pg_isready  -q ; echo $?,`date`; sleep 10; done;
（別セッションで以下を実行）
$ while true; do pg_isready -h 172.16.3.102 -q ; echo $?,`date`; sleep 10; done;
```

  ・スレーブ側 (postgresユーザで実行)

```
$ while true; do pg_isready  -q ; echo $?,`date`; sleep 10; done;
（別セッションで以下を実行）
$ while true; do pg_isready -h 172.16.3.101 -q ; echo $?,`date`; sleep 10; done;
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

□ 環境準備(続き)

- **監視スクリプト(続き)**

・出力例

リターンコード
0:正常
1:接続拒否
2:応答なし

```
0, 2013年 9月 24日 火曜日 13:21:44 JST
0, 2013年 9月 24日 火曜日 13:21:54 JST
2, 2013年 9月 24日 火曜日 13:22:04 JST
・・・
```

・注意点

pg_isreadyで他サーバを監視すると以下のようなメッセージがサーバログに出力される。

```
FATAL:  no pg_hba.conf entry for host "172.16.3.101", user "postgres", database "postgres"
```

pg_hba.confに適切なエントリを追加することで防止できる。

```
[/disk1/data/pg_hba.conf]
host    postgres    postgres        172.16.3.101/32     md5
host    postgres    postgres        172.16.3.102/32     md5
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

□ 環境準備(続き)

- **レプリケーション状況確認**

  ・マスタ側 (postgresユーザで実行)

```
$ psql -x -c "select * from pg_stat_replication"
-[ RECORD 1 ]----+------------------------------
pid              | 16965
usesysid         | 16384
usename          | repuser
application_name | walreceiver
client_addr      | 172.16.1.102
client_hostname  |
client_port      | 54694
backend_start    | 2013-09-24 13:32:50.573419+09
state            | streaming
sent_location    | 0/17000090
write_location   | 0/17000090
flush_location   | 0/17000090
replay_location  | 0/17000090
sync_priority    | 0
sync_state       | async
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

① 障害発生

マスタサーバのWAL領域のディスク障害が発生し、WALを書き込めなくなったためにマスタサーバのPostgreSQLがダウン。

障害シミュレート手順：WAL領域を読み取り専用に

```
$ chmod u-w /disk2/pg_xlog
```

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

② 監視スクリプトにより障害検知

環境準備で実行した監視スクリプトの出力を確認。
行頭に「２」が出力されていることを確認。

```
0,2013年  10月  15日  火曜日  12:13:06 JST
・・・中略
0,2013年  10月  15日  火曜日  12:13:36 JST
0,2013年  10月  15日  火曜日  12:13:46 JST
1,2013年  10月  15日  火曜日  12:13:56 JST
1,2013年  10月  15日  火曜日  12:14:06 JST
・・・中略
1,2013年  10月  15日  火曜日  12:14:46 JST
1,2013年  10月  15日  火曜日  12:14:56 JST
2,2013年  10月  15日  火曜日  12:15:06 JST
2,2013年  10月  15日  火曜日  12:15:16 JST
・・・
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

② 監視スクリプトにより障害検知(続き)

```
0,2013年 1
・・・中略
0,2013年 1
0,2013年 1
1,2013
1,2013
・・・中略
1,2013年 10月 15日 火曜日 12:14:40 JST
1,2013年 10月 15日 火曜日 12:14:56 JST
2,2013年 10月 15日 火曜日 12:15:06 JST
2,2013年 10月 15日 火曜日 12:15:16 JST
・・・
```

pg_isreadyのリターンコードが１のとき

・「pg_ctl status」コマンドで状態確認した結果
　→ "pg_ctl: server is running (PID: 12615)"

・psコマンドで確認したプロセスの状態

```
postgres 12615     1  0 12:09 pts/6    00:00:02 /usr/local/pgsql/bin/postgres
postgres 13235 12615 93 12:13 ?        00:00:01 postgres: startup process   recovering
000000010000000100000033
```

pg_isreadyのリターンコードが２のとき

・「pg_ctl status」コマンドで状態確認した結果
　→ "pg_ctl: no server running"

・psコマンドで確認したプロセスの状態
プロセスなし

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

② 監視スクリプトにより障害検知（続き）

サーバログ抜粋

```
LOG:  archive command failed with exit code 1
DETAIL:  The failed archive command was: test ! -f /disk3/archive/000000010000000100000020 && cp
pg_xlog/000000010000000100000020 /disk3/archive/000000010000000100000020
・・・
LOG:  archiver process (PID 12622) exited with exit code 1
LOG:  all server processes terminated; reinitializing
LOG:  database system was interrupted; last known up at 2013-10-15 12:09:53 JST
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:  redo starts at 1/20FFF688
LOG:  redo done at 1/3AFFE418
LOG:  last completed transaction was at log time 2013-10-15 12:13:26.321251+09
FATAL:  the database system is in recovery mode
・・・
FATAL:  the database system is in recovery mode
LOG:  could not link file "pg_xlog/000000010000000000000C1" to "pg_xlog/000000010000000010000003B" (initialization of log file): 許可がありません
LOG:  could not remove old transaction log file "pg_xlog/000000010000000000000C1": 許可がありません
・・・
FATAL:  could not create file "pg_xlog/xlogtemp.13235": 許可がありません
LOG:  startup process (PID 13235) exited with exit code 1
LOG:  aborting startup due to startup process failure
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

③ 障害状況の分析と、スレーブをマスタに昇格

- **障害状況の分析**
  実運用では障害状況・障害原因の分析を行なうが、本検証では割愛。
  マスタサーバでPostgreSQLのプロセスが残っている場合には、スレーブをマスタに昇格させる前にPostgreSQLを停止することが必要である。PostgreSQLを停止するには

```
$ pg_ctl -m fast stop
```

  を実行する。上記で停止できない場合

```
$ pg_ctl -m immediate stop
```

  で強制停止させる。

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

③ 障害状況の分析と、スレーブをマスタに昇格(続き)

- **スレーブをマスタに昇格**
  マスタサーバ上のPostgreSQLの停止を確認後、スレーブサーバのPostgreSQLをプロモートし、マスタとして動作させる。

```
## 最後にレプリケーションされたトランザクションの
## タイムスタンプを確認しておく
$ psql -c "SELECT pg_last_xact_replay_timestamp()"

## プロモート
$ pg_ctl promote
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

③ 障害状況の分析と、スレーブをマスタに昇格(続き)

- **スレーブをマスタに昇格(続き)**
  ログに以下のようなメッセージが出力されることを確認

```
LOG:  received promote request
LOG:  redo done at 0/17000028
LOG:  last completed transaction was at log time 2013-09-20 16:34:13.988285+09
LOG:  selected new timeline ID: 2
LOG:  archive recovery complete
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
```

このメッセージが出力されるとデータベースとして使用可能な状態

- **マスタに昇格後、業務旧に必要な処理を行う。**

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

④ 新マスタのバックアップを取得する

新たにマスタとなったPostgreSQLのバックアップを早急に取得する。
この時点では旧マスタサーバのハードウェアはまだ復旧していない想定で、バックアップは新マスタサーバのローカルで取得する。

```
## バックアップ用ディレクトリの作成
# mkdir -p /disk3/backup/data
# chown postgres:postgres /disk3/backup/data

## バックアップ取得
# su - postgres
$ pg_basebackup -h 127.0.0.1 -U repuser -D /disk3/backup/data --xlog --progress
```

"localhost" と指定すると設定次第によってはIPv6で接続要求され、pg_hba.confにIPV6のエントリがないため接続できないことがあるので注意。

pg_hba.conf に

host replication repuser 127.0.0.1/32 md5

のエントリがないとエラーになる。

「--progress」オプションをつけると、端末へのバックアップ状況が出力されるが、その処理がボトルネックになることもあるので注意が必要。

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

⑤ 旧マスタを新スレーブとするSR環境構築

- 旧マスタのハードウェア復旧、PostgreSQL設定後、新マスタとSR環境を構築する。（旧マスタが新スレーブとなり、障害発生前とは逆方向のSRとなる。）

- 旧マスタのデータがディスク上に残っている場合、退避しておく。

```
$ mv /disk1/data  /disk4/broken/
$ mv /disk2/pg_xlog  /disk4/broken/
$ mv /disk3/archive  /disk4/broken/
```

データ退避用のディレクトリを適宜指定

- 新スレーブで「 1. 環境準備 (1) PostgreSQL 9.3環境構築 インストール」までが完了している状態とする。

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

⑤ 旧マスタを新スレーブとするSR環境構築（続き）

- 新マスタのベースバックアップを取得し、SR環境を構築する。

```
# mkdir /disk1/data
# mkdir /disk2/pg_xlog
# mkdir /disk3/archive
# chown postgres:postgres /disk1/data
# chown postgres:postgres /disk2/pg_xlog
# chown postgres:postgres /disk3/archive
# su - postgres
$ pg_basebackup -h 172.16.3.102 -U repuser -D /disk1/data --progress
 password:

$ rmdir /disk1/data/pg_xlog
$ cd /disk1/data
$ ln -s /disk2/pg_xlog pg_xlog
```

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

⑤ 旧マスタを新スレーブとするSR環境構築（続き）

- recovery.confを作成し、postgresql.confを修正

新スレーブで以下を実施

```
[/disk1/data/recovery.conf]
standby_mode = 'on'
primary_conninfo = 'host=172.16.3.102 port=5432 user=repuser password=repuser'
restore_command = 'scp /disk2/pg_xlog/%f "%p" 2> /dev/null'

[/disk1/data/postgresql.conf]
hot_standby = on
```

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

⑤ 旧マスタを新スレーブとするSR環境構築（続き）

- 新スレーブでPostgreSQLを起動し、レプリケーションの確認

新スレーブで以下を実施

```
$ chmod 700 /disk1/data
$ pg_ctl start
```

新マスタサーバで以下を実施

```
$ psql -x -c "select * from pg_stat_replication"
-[ RECORD 1 ]----+-----------------------------
 ・ ・ ・
client_addr      | 172.16.3.101
 ・ ・ ・
state            | streaming
 ・ ・ ・
sync_state       | async
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

⑤ 旧マスタを新スレーブとするSR環境構築(続き)

- 新マスタでトランザクションを実行し、レプリケーションの確認

新マスタで以下を実施

```
$ pgbench -T 180 testdb
```

終了後新マスタサーバで以下を実施してレプリケーション状況確認

```
$ psql -x -c "select * from pg_stat_replication"
```

新スレーブサーバでは以下を実行して確認

```
$ psql -c "SELECT pg_last_xact_replay_timestamp()"
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

⑥ 新スレーブをマスタに戻す(スイッチバック)

新マスタ-新スレーブのSR構成の役割を入れ替え、障害発生前の構成に戻す。

- 新マスタサーバでrecovery.confを作成し、postgresql.confを修正

```
[/usr/local/pgsql/data/recovery.conf]
standby_mode = 'on'
primary_conninfo = 'host=172.16.3.101 port=5432 user=repuser password=repuser'
restore_command = 'scp /disk2/pg_xlog/%f "%p" 2> /dev/null'

[/usr/local/pgsql/data/postgresql.conf]
hot_standby = on
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

⑥ 新スレーブをマスタに戻す(スイッチバック)(続き)

- 新マスタでPostgreSQLを正常終了させる

```
$ pg_ctl -m fast stop
```

- 正常終了を確認後、新スレーブサーバでレプリケーション状況を確認し、マスタに昇格させる。

```
## 最後にレプリケーションされたトランザクションの
## タイムスタンプを確認しておく
$ psql -c "SELECT pg_last_xact_replay_timestamp()"

## プロモート
$ pg_ctl promote
```

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

⑥ 新スレーブをマスタに戻す（スイッチバック）（続き）

- マスタとなったサーバのログに以下のようなメッセージが出力されていることを確かめる。

```
LOG:  received promote request
LOG:  redo done at 0/22000028
LOG:  last completed transaction was at log time 2013-09-24 15:57:27.366339+09
LOG:  selected new timeline ID: 3
LOG:  archive recovery complete
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
```

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

⑥ 新スレーブをマスタに戻す（スイッチバック）（続き）

- 新マスタであったサーバ上でPostgreSQLを起動する。（スレーブとして起動される。）

```
$ pg_ctl start
```

3. SR環境での障害／復旧シナリオ（１）（続き）
～スレーブサーバの昇格による業務継続～

(2) 検証手順（続き）

⑥ 新スレーブをマスタに戻す（スイッチバック）（続き）

- スレーブサーバとなったサーバのログを確認

```
LOG:  database system was shut down at 2013-09-24 16:24:59 JST
LOG:  entering standby mode
LOG:  restored log file "00000002.history" from archive
LOG:  restored log file "000000020000000000000022" from archive
LOG:  consistent recovery state reached at 0/22000090
LOG:  database system is ready to accept read only connections
LOG:  record with zero length at 0/22000090
LOG:  record with zero length at 0/22000090
LOG:  fetching timeline history file for timeline 3 from primary server
LOG:  started streaming WAL from primary at 0/22000000 on timeline 2
LOG:  replication terminated by primary server
DETAIL:  End of WAL reached on timeline 2 at 0/22000090.
LOG:  restored log file "00000003.history" from archive
LOG:  restored log file "00000003.history" from archive
LOG:  new target timeline is 3
LOG:  restored log file "000000020000000000000022" from archive
LOG:  record with zero length at 0/22000090
LOG:  restarted WAL streaming at 0/22000000 on timeline 3
LOG:  redo starts at 0/22000090
```

3. SR環境での障害／復旧シナリオ(１)(続き)
～スレーブサーバの昇格による業務継続～

(2) 検証手順(続き)

⑥ 新スレーブをマスタに戻す(スイッチバック)(続き)

- 最終的なマスタでトランザクションを実行し、レプリケーションの確認

新マスタで以下を実施

```
$ pgbench -T 180 testdb
```

終了後マスタサーバで以下を実施してレプリケーション状況確認

```
$ psql -x -c "select * from pg_stat_replication"
```

スレーブサーバでは以下を実行して確認

```
$ psql -c "SELECT pg_last_xact_replay_timestamp()"
```

4. SR環境での障害／復旧シナリオ（２）
～マスタをバックアップからリカバリ～

（1）シナリオ概要

- ○○日△△時××分、オペレーションミスにより、データを誤って削除。
- スレーブサーバの状況を確認したところ、スレーブサーバでもすでに削除。
- スレーブサーバで毎日取得しているバックアップからマスタを○○日△△時××分の直前までリカバリ。
- SR環境再構築

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順

□ 環境準備

- スレーブのバックアップ取得(通常運用時に取得しているバックアップの想定)

```
## バックアップディレクトリを作成
# mkdir -p /disk3/backup/YYYYMMDD/data
# chown -R postgres:postgres /disk3/backup/YYYYMMDD

## バックアップ取得
# su - postgres
$ pg_basebackup -h 127.0.0.1 -U repuser -D /disk3/backup/YYYYMMDD/data --xlog --progress
```

YYYYMMDDには実施日を指定

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

□ 環境準備(続き)

- マスタサーバでリカバリ時の確認用データを作成

```
# su - postgres
$ pgbench -T 60 testdb
$ psql testdb
testdb=# select max(mtime) from pgbench_history;
 max
----------------------------
 2013-09-24 16:39:14.379332
(1 row)
```

リカバリしたときに
このデータが見えること

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

① データを誤って削除。
マスタサーバでリカバリ時の確認用データを作成後、
数分待ちテーブルをtruncateする。

```
$ psql testdb
testdb=# select max(mtime) from pgbench_history;
 max
----------------------------
 2013-09-24 16:39:14.379332
(1 row)

testdb=# select current_timestamp;
              now
-------------------------------
 2013-09-24 16:45:04.834111+09
(1 row)

testdb=# truncate table pgbench_history;
```

数分時間の差があること
を確認

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

② スレーブの状況確認。

```
$ psql testdb
testdb=# select count(*) from pgbench_history;
 count
-------
     0
(1 row)
```

スレーブもTruncateされている。

4. SR環境での障害／復旧シナリオ（２）（続き）
～マスタをバックアップからリカバリ～

(2) 検証手順（続き）

③ マスタをTruncate直前までリカバリ

- マスタ、スレーブのPostgreSQLを停止。
  両サーバで以下を実施（マスタ側から実施）

```
$ pg_ctl -m fast stop
```

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

③ マスタをTruncate直前までリカバリ(続き)

- **データ領域を退避する。**
  マスタサーバで以下を実施

```
# mkdir -p /disk4/broken/YYYYMMDD/
# mv /disk1/data /disk4/broken/YYYYMMDD/
```

mv先は適宜指定。
必要に応じてWAL領域も退避してもよい。

- **スレーブのバックアップをマスタにコピー**
  マスタサーバで以下を実施

```
# su - postgres
$ scp -pr 172.16.3.102:/disk3/backup/YYYYMMDD/data  /disk1
$ chmod 700 /disk1/data
```

最新のバックアップデータが配置されているディレクトリ

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

③ マスタをTruncate直前までリカバリ(続き)

- ベースバックアップをリストアするとWAL領域(pg_xlog)がdataディレクトリの下に作成されるため、正しいパスに設定しなおす。
  マスタサーバで以下を実施

```
$ rm -rf /disk1/data/pg_xlog
$ cd /disk1/data
$ ln -s /disk2/pg_xlog pg_xlog
```

WAL領域がdataディレクトリの下にあるデフォルト構成の場合、左記のかわりに退避したWALをpg_xlogに戻す作業が必要になる。

※ 今回の検証シナリオではテーブル削除という論理障害であるが、
データ領域破損、データファイル破損という物理障害であっても、
マスタのWALが破損していなければ同じ手順でリカバリ可能である。

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

③ マスタをTruncate直前までリカバリ(続き)

- マスタサーバでrecovery.confを設定し、postgresql.confを修正

```
[/disk1/data/recovery.conf]
restore_command = 'cp /disk3/archive/%f "%p" 2> /dev/null'
recovery_target_time = '2013-09-24 16:45:04'

[/disk1/data/postgresql.conf]
hot_standby = off
```

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

③ マスタをTruncate直前までリカバリ(続き)

- マスタサーバでPostgreSQLを起動

```
$ pg_ctl start
```

- ログに以下のようなメッセージが出力されていることを確認

```
LOG:  starting point-in-time recovery to 2013-09-24 16:45:04+09
LOG:  redo starts at 0/25D2D920
LOG:  consistent recovery state reached at 0/25D2D9C0
LOG:  record with zero length at 0/25D2D9C0
LOG:  redo done at 0/25D2D958
LOG:  database system is ready to accept read only connections
LOG:  selected new timeline ID: 4
LOG:  archive recovery complete
LOG:  database system is ready to accept connections
LOG:  autovacuum launcher started
```

1行前の「archive recovery complete 」がでてから、この行がでるまで数分かかる場合もあるので注意

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

③ マスタをTruncate直前までリカバリ(続き)

- マスタサーバでデータ確認

```
$ psql testdb
testdb=# select max(mtime) from pgbench_history;
            max
----------------------------
 2013-09-24 16:39:14.379332
(1 row)
```

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

④ SR環境再構築

- スレーブサーバのデータを退避し、ベースバックアップを取得

```
# mkdir -p /disk4/broken/YYYYMMDD/
# mv /disk1/data /disk4/broken/YYYYMMDD/
# mv /disk2/pg_xlog /disk4/broken/YYYYMMDD/
# mv /disk3/archive /disk4/broken/YYYYMMDD/

# mkdir /disk1/data
# mkdir /disk2/pg_xlog
# mkdir /disk3/archive
# chown postgres:postgres /disk1/data
# chown postgres:postgres /disk2/pg_xlog
# chown postgres:postgres /disk3/archive
# su - postgres
$ pg_basebackup -h 172.16.3.101 -U repuser -D /disk1/data --progress
 password:

$ rmdir /disk1/data/pg_xlog
$ cd /disk1/data
$ ln -s /disk2/pg_xlog pg_xlog
```

退避先は適宜指定

マスタをPITRでリカバリした場合、再度マスタからベースバックアップを取得する必要がある。

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

④ SR環境再構築

- スレーブサーバでrecovery.conf、postgresql.confを設定

```
[/disk1data/recovery.conf]
standby_mode = 'on'
primary_conninfo = 'host=172.16.3.101 port=5432 user=repuser password=repuser'
restore_command = 'scp /disk2/pg_xlog/%f "%p" 2> /dev/null'
recovery_target_timeline='latest'

[/disk1data/postgresql.conf]
hot_standby = on
```

4. SR環境での障害／復旧シナリオ（２）（続き）
～マスタをバックアップからリカバリ～

(2) 検証手順（続き）

④ SR環境再構築（続き）

- スレーブサーバでPostgreSQLを起動

```
$ pg_ctl start
```

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

④ SR環境再構築(続き)

- ログに以下のようなメッセージが出力されていることを確認

```
LOG:  entering standby mode
LOG:  restored log file "00000003.history" from archive
LOG:  restored log file "000000030000000000000025" from archive
LOG:  redo starts at 0/25D2D920
LOG:  consistent recovery state reached at 0/25D2D9C0
LOG:  record with zero length at 0/25D2D9C0
LOG:  record with zero length at 0/25D2D9C0
LOG:  database system is ready to accept read only connections
LOG:  fetching timeline history file for timeline 4 from primary server
LOG:  started streaming WAL from primary at 0/25000000 on timeline 3
LOG:  replication terminated by primary server
DETAIL:  End of WAL reached on timeline 3 at 0/25D2D9C0.
LOG:  restored log file "00000004.history" from archive
LOG:  restored log file "00000004.history" from archive
LOG:  new target timeline is 4
LOG:  restored log file "000000030000000000000025" from archive
LOG:  record with zero length at 0/25D2D9C0
LOG:  restarted WAL streaming at 0/25000000 on timeline 4
```

4. SR環境での障害／復旧シナリオ(２)(続き)
～マスタをバックアップからリカバリ～

(2) 検証手順(続き)

④ SR環境再構築(続き)

- スレーブサーバでデータ確認

```
$ psql testdb
testdb=# select max(mtime) from pgbench_history;
            max
----------------------------
 2013-09-24 16:39:14.379332
(1 row)
```

再構築完了

5. ssh/scp設定

- マスタサーバ(172.16.3.101)において
  postgresユーザで以下を実行

```
$ ssh-keygen -t rsa
（入力項目はすべてenter）

$ cat ~/.ssh/id_rsa.pub | ssh 172.16.3.102 "cat >>.ssh/authorized_keys && chmod
600 .ssh/authorized_keys"
```

- スレーブサーバ(172.16.3.102)において
  postgresユーザで以下を実行

```
$ ssh-keygen -t rsa
（入力項目はすべてenter）

$ cat ~/.ssh/id_rsa.pub | ssh 172.16.3.101 "cat >>.ssh/authorized_keys && chmod
600 .ssh/authorized_keys"
```