PostgreSQLエンタープライズ・コンソーシアム 技術部会 WG#3

2017年度WG3活動報告書
Windows環境調査編

目次

1. ライセンス

2. はじめに

2.1. PostgreSQLエンタープライズコンソーシアムとWG3について

PostgreSQLエンタープライズコンソーシアム(略称 PGECons)は、PostgreSQL本体および各種ツールの情報収集と提供、整備などの活動を通じて、ミッションクリティカル性の高いエンタープライズ領域へのPostgreSQLの普及を推進することを目的として設立された団体です。

PGECons 技術部会ではPostgreSQLの普及に資する課題を活動テーマとし、3つのワーキンググループで具体的な活動を行っています。

- WG1（新技術検証ワーキンググループ）
- WG2（移行ワーキンググループ）
- WG3（課題検討ワーキンググループ）

これら3つのワーキンググループのうち、WG1、WG3については 2015 年度まではそれぞれ、「性能ワーキンググループ」、「設計運用ワーキンググループ」という名称で活動してきました。2016年度は、従来の活動領域を広げる意図のもとでそれらを再定義し、上記のような名称に改めました。

これに伴い、WG3ではPostgreSQLの設計運用を中心としたさまざまな課題の解決のための調査検証を行い、PostgreSQLが広く活用される事を推進していくこととしました。

2.2. 本資料の概要と目的

本資料はWG3の2017年度の活動として、WindowsにおけるPostgreSQLについて、インストレーション、ユーティリティコマンド、HAの切り口で調査、検証した結果をまとめたものです。

PostgreSQLは2005年、PostgreSQL 8.0.0のリリースからWindowsをサポートするようになりました。一方で、PostgreSQLは長らくLinux環境を中心に開発・利用されていたこともあり、Windowsでの環境構築、周辺ツールの対応状況、ユーティリティコマンドのLinux環境との比較や、高可用性を想定した構成を組むための情報があまり整理されていない状況にありました。

今年度はWindows上でのPostgreSQLに関して、実際にWindows環境での検証と机上調査を通して情報を整備することを目的として活動しました。

2.3. 本資料の構成

- はじめに
- PostgreSQLのインストレーション
  - Windows環境でのソースからのインストレーション
  - Windows環境でのインストーラからのインストレーション
  - インストレーションのカスタマイズ可否および各種ツールのWindowsでの対応状況
  - まとめ
- PostgreSQLの運用手順確認
  - WAL書き出し先を変更したデータベースクラスタの初期化
  - PostgreSQLの起動と停止
  - PostgreSQLのイベントログ出力
  - PostgreSQLのバックアップ・リストア
  - まとめ
- High-Availablity
  - 調査、検証の目的
  - 環境構築手順
  - pgpool-II運用におけるPCPコマンドの動作確認
  - 障害での動作検証
  - まとめ
- 著者

2.4. 想定読者

本資料の読者は以下のような知識を有していることを想定しています。

- DBMSを操作してデータベースの構築、保守、運用を行うDBAの知識
- PostgreSQLを利用する上での基礎的な知識

3. PostgreSQLのインストレーション

本章では、Windows環境におけるPostgreSQLのインストレーション、インストレーションを実施した環境とLinux版PostgreSQLとの差異について説明します。

3.1. Windows環境でのソースからのインストレーション

本節では、PostgreSQLのWindows環境におけるソースからのビルドおよび インストール方法について検証します。

3.1.1. 前提条件

検証した環境は以下の通りです。

- OS:Windows 7 Professional 32bit
- CPU:Core2Duo 6300 1.86GHz
- Memory:2GB

3.1.2. 環境構築

ソースからのインストレーションの検証には、以下の環境を利用しました。

- Microsoft Visual Studio Express 2017 for Windows Desktop
- Microsoft Windows SDK 8.1
- Strawberry Perl 5.26.1
- PostgreSQL 10.1(ソース)

また本検証では、PostgreSQL Document「Chapter 17. Installation from Source Code on Windows」の 記載に沿った内容として、Perlのランタイムに以下を使用した検証も実施しました。

- ActiveState Perl 5.24.2 Community Edition

ただし、ActiveState Perlは、利用目的によりはライセンス上、もしくはエディション上 制限がある場合があります。使用する目的に応じたエディションを選択ください。

3.1.2.1. Microsoft Visual Studio Express 2017 for Windows Desktopのインストール

1. MicrosoftのVisual StudioのWebページ から「Visual Studio Express」を選択してダウンロードします。
2. インストーラの手順に従ってインストールします。

3.1.2.2. Microsoft Windows SDKのインストール

1. MicrosoftのWindows SDKのWebページ から「Windows Standalone SDK」を選択してダウンロードします。
2. インストーラの手順に従ってインストールします。

3.1.2.3. Strawberry Perlのインストール

1. Strawberry PerlのWebページ からダウンロードします。
2. インストーラの手順に従ってインストールします。

3.1.3. ビルド

1. 開発者コマンドプロンプト for VS 2017を起動します。
2. [ソースの配置先]\src\tools\msvc 配下に移動し、ビルドコマンドを実行します。

```
> cd postgresql-10.0\src\tools\msvc
> build.bat
```

3. 以下のようなメッセージが表示されていれば、ビルドは成功しています(警告の件数、経過時間は環境により増減することがあります)。

```
> build.bat

(省略)

ビルドに成功しました。

(省略、警告がある場合に、ここに出力される)

    7 個の警告
    0 エラー

経過時間 00:18:52.43
```

3.1.4. インストール

1. 以下を実行します。

```
> install.bat [インストール先ディレクトリ]
```

2. 以下のようなメッセージが表示されていれば、インストールは成功しています。

```
> install.bat c:\PostgreSQL
Installing version 10 for release in c:\PostgreSQL

(省略)

Installation complete.
```

3.1.5. 動作確認

インストールしたバイナリが動作するか簡単にテストします。

- 環境変数の設定

```
# 現在の設定内容を確認
> echo %PATH%
(省略)

# PATHの末尾にPostgreSQLの
> set PATH=%PATH%;c:\PostgreSQL\bin
> echo %PATH%
(省略)...;c:\PostgreSQL\bin

# データベースクラスタの設定 (後の手順でデータベースクラスタを作成)
> set PGDATA=c:\PostgreSQL\data
```

- データベースクラスタの作成(initdb)

```
> cd C:\PostgreSQL\bin
> initdb c:\PostgreSQL\data
The files belonging to this database system will be owned by user "[コマンドを実行したユーザ名]".
This user must also own the server process.

The database cluster will be initialized with locale "Japanese_Japan.932".
Encoding "SJIS" implied by locale is not allowed as a server-side encoding.
The default database encoding will be set to "UTF8" instead.
initdb: could not find suitable text search configuration for locale "Japanese_Japan.932"
The default text search configuration will be set to "simple".

Data page checksums are disabled.

creating directory c:/PostgreSQL/data ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting dynamic shared memory implementation ... windows
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    pg_ctl -D ^"c^:^\PostgreSQL^\data^" -l logfile start
```

- DBクラスタの起動

```
> pg_ctl -l c:\PostgreSQL\log.txt start
waiting for server to start.... done
server started
```

- DB作成、テーブル作成とデータ参照

```
> createdb testdb
> echo %ERRORLEVEL%
0

> psql -U [ユーザ名] testdb
psql (10.1)
Type "help" for help.

testdb=# create table testtable(id int, name text);
CREATE TABLE

testdb=# insert into testtable values (1,'MyName');
INSERT 0 1

testdb=# select * from testtable;
 id |  name
----+--------
  1 | MyName
(1 row)
```

3.2. Windows環境でのインストーラからのインストレーション

本節では、PostgreSQLのWindows環境におけるインストーラによる インストール方法について検証します。

3.2.1. 前提条件

使用したインストーラのバージョンは以下の通りです。

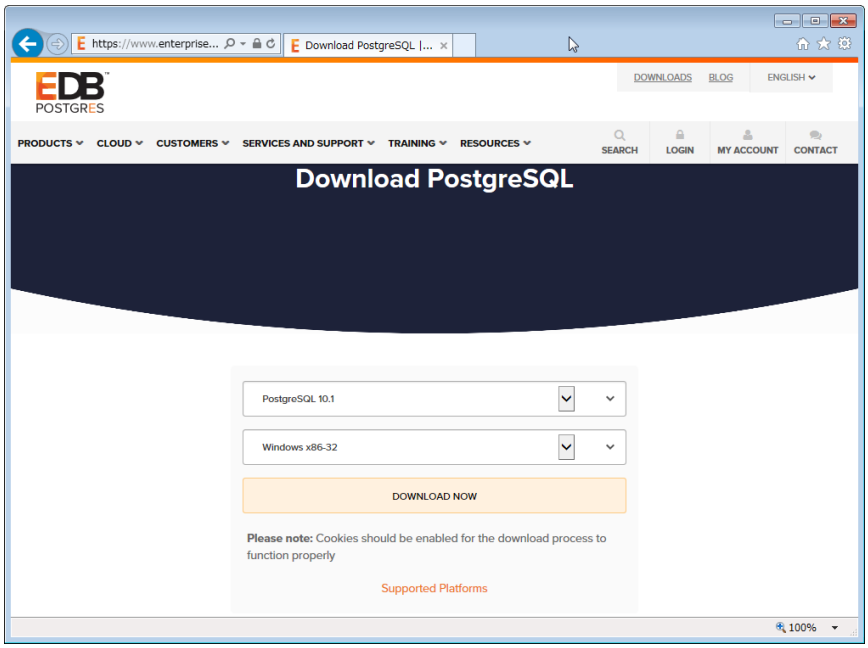
- postgresql-10.1-3-windows.exe

3.2.2. 環境構築

Windows環境でのソースからのインストレーションと同一の環境を利用しました。

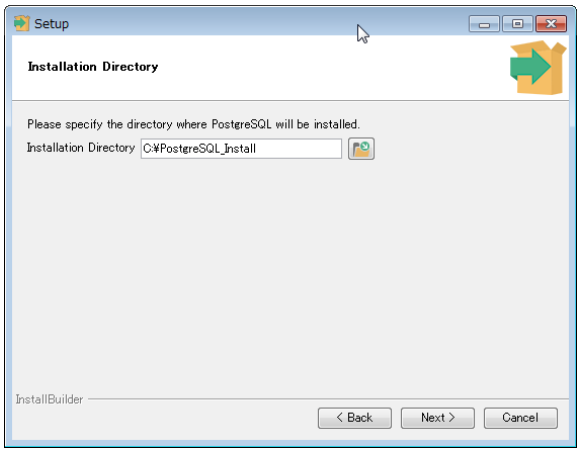
3.2.3. インストーラの入手先

Windows用のPostgreSQLのインストーラは、EnterpriseDB社のWebページ からダウンロードします。PostgreSQLのバージョンとインストール先のOSが選択できます。

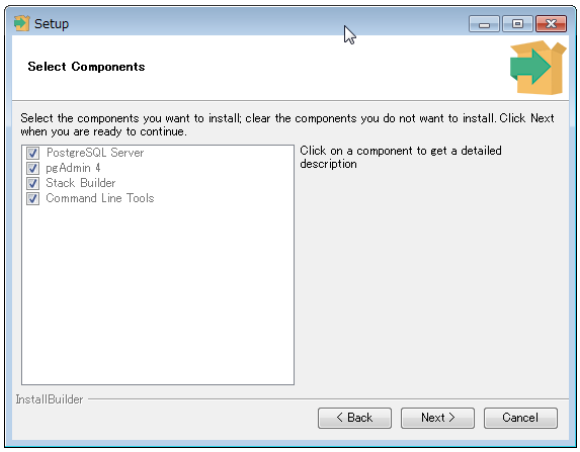


3.2.4. インストール

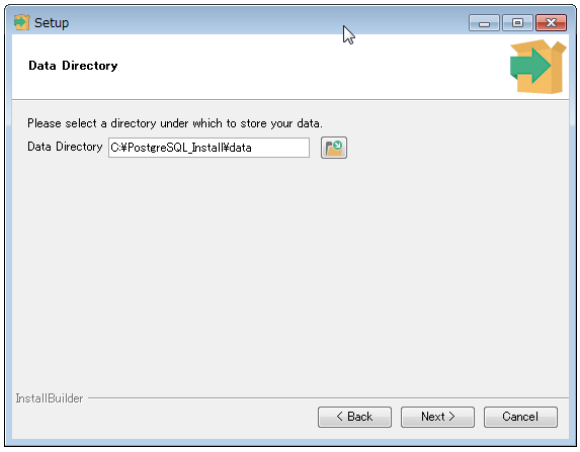
1. ダウンロードしたインストーラを起動します。
2. インストール先のディレクトリを選択します。今回は「C:\PostgreSQL_Install」とします。



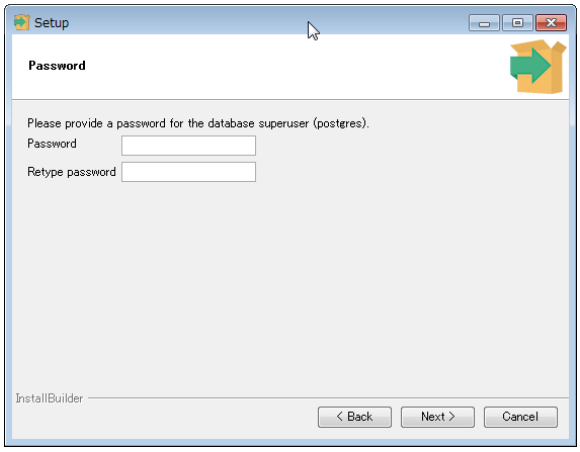
3. インストールするコンポーネントを選択します。



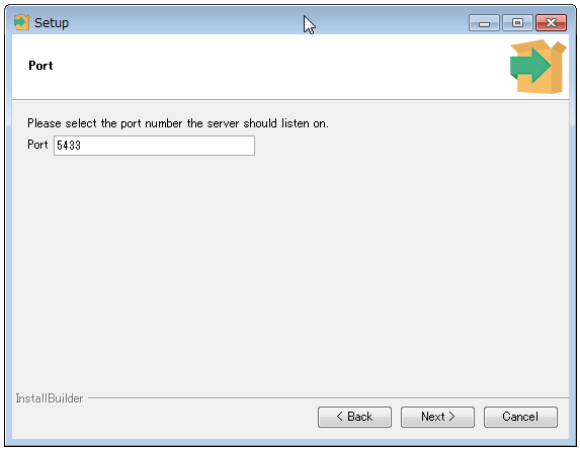
4. DBクラスタの作成先を選択します。



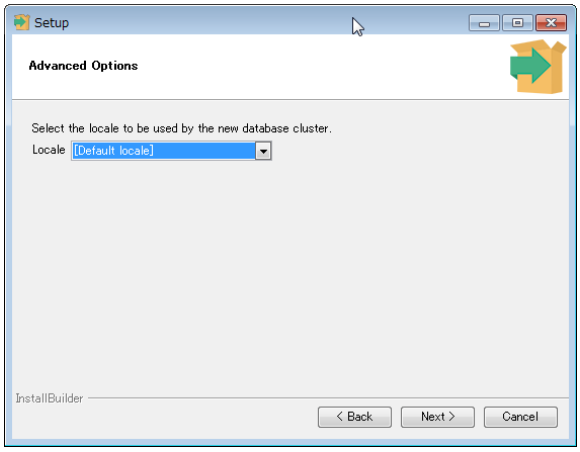
5. データベースのスーパーユーザのパスワードを入力します。



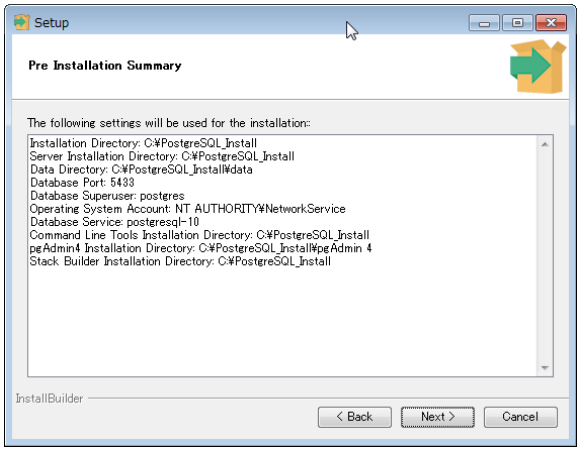
6. DBの待ち受けポート番号を選択します。本検証ではindows環境でのソースからのインストレーションと同一環境を利用したため、ポート番号を「5433」に変更しています。



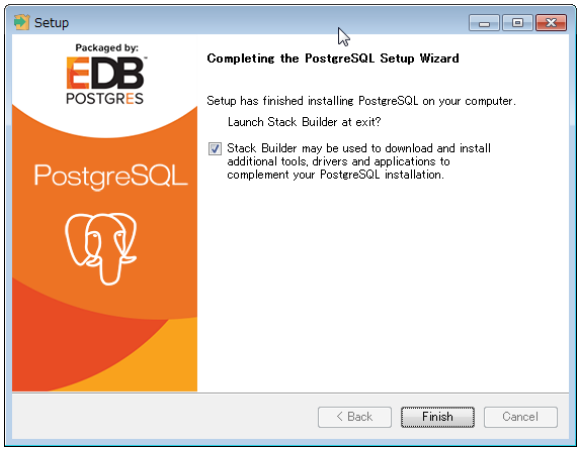
7. 使用するDBのロケールを選択します。



8. これまで選択したインストール条件を確認します。



9. インストールを実施します。以下の画面が表示されれば、インストールは完了です。

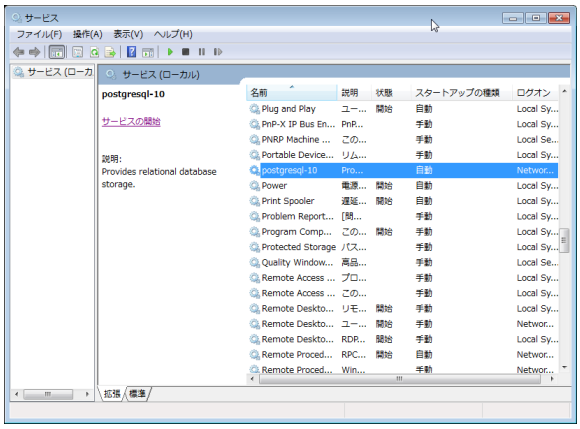


3.2.5. 動作確認

インストールしたバイナリが動作するか簡単にテストします。

- サービスの起動

[コントロール パネル] → [管理ツール] → [サービス]の順にクリックし、「postgresql-10」サービスに対して[サービスの開始]を実行します。



- 環境変数の設定

```
# 現在の設定内容を確認
> echo %PATH%
(省略)

# PATHの末尾にPostgreSQLの
> set PATH=%PATH%;c:\PostgreSQL_Install\bin
> echo %PATH%
(省略)...;c:\PostgreSQL_Install\bin

# データベースクラスタの設定(後の手順でデータベースクラスタを作成)
> set PGDATA=c:\PostgreSQL_Install\data
```

- DBクラスタの作成(initdb)

```
> initdb c:\PostgreSQL_Install\data
データベースシステム内のファイルの所有者は"[コマンドを実行したユーザ名]"ユーザでした。
このユーザがサーバプロセスを所有しなければなりません。

データベースクラスタはロケール"Japanese_Japan.932"で初期化されます。
ロケールにより暗示される符号化方式"SJIS"はサーバ側の符号化方式として使用できません。
デフォルトのデータベース符号化方式は代わりに"UTF8"に設定されます。
initdb: ロケール"Japanese_Japan.932"用の適切なテキスト検索設定が見つかりません
デフォルトのテキスト検索設定はsimpleに設定されました。

データベージのチェックサムは無効です。

ディレクトリc:/PostgreSQL_Install/dataを作成しています ... ok
サブディレクトリを作成しています ... ok
デフォルトのmax_connectionsを選択しています ... 100
デフォルトの shared_buffers を選択しています ... 128MB
selecting dynamic shared memory implementation ... windows
設定ファイルを作成しています ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
データをディスクに同期しています...ok

警告: ローカル接続向けに"trust"認証が有効です。
pg_hba.confを編集する、もしくは、次回initdbを実行する時に-Aオプション、
または、--auth-localおよび--auth-hostを使用することで変更することができます。

Success. You can now start the database server using:

    pg_ctl -D ^"c^:^\PostgreSQL^_Install^\data^" -l logfile start
```

- DBクラスタの起動

```
> pg_ctl -l c:\PostgreSQL_install\log.txt start
サーバの起動完了を待っています....完了
サーバ起動完了
```

- DB作成、テーブル作成とデータ参照

```
> createdb testdb
> echo %ERRORLEVEL%
0

> psql -U [ユーザ名] testdb
psql (10.1)
Type "help" for help.

testdb=# create table testtable(id int, name text);
CREATE TABLE

testdb=# insert into testtable values (1,'MyName');
INSERT 0 1

testdb=# select * from testtable;
 id |  name
----+--------
  1 | MyName
(1 row)
```

3.3. インストレーションのカスタマイズ可否および各種ツールのWindowsでの対応状況

3.3.1. 前提条件

本章では、以下の環境に対して比較を実施しました。

- インストレーションのカスタマイズ可否(Linuxの場合とWindowsの場合、Windowsの場合はインストーラによるインストール時の設定値との比較)
- contribのWindowsでの対応状況
- 周辺ツールのWindowsでの対応状況

3.3.2. インストレーション時のカスタマイズ可否の差異

ソースからのインストレーションを実施した場合、インストーラからのインストレーションを実施した場合 それぞれの環境でのpg_configコマンドの実行結果を以下に示します。

- ソースからのインストレーションを実施した場合(C:\PostgreSQLにインストール)

```
> pg_config
BINDIR = C:/POSTGR~1/bin
DOCDIR = C:/POSTGR~1/doc
HTMLDIR = C:/POSTGR~1/doc
INCLUDEDIR = C:/POSTGR~1/include
PKGINCLUDEDIR = C:/POSTGR~1/include
INCLUDEDIR-SERVER = C:/POSTGR~1/include/server
LIBDIR = C:/POSTGR~1/lib
PKGLIBDIR = C:/POSTGR~1/lib
LOCALEDIR = C:/PostgreSQL/share/locale
MANDIR = C:/PostgreSQL/man
SHAREDIR = C:/POSTGR~1/share
SYSCONFDIR = C:/PostgreSQL/etc
PGXS = C:/PostgreSQL/lib/pgxs/src/makefiles/pgxs.mk
CONFIGURE = --enable-thread-safety --with-ldap --without-zlib
CC = not recorded
CPPFLAGS = not recorded
CFLAGS = not recorded
CFLAGS_SL = not recorded
LDFLAGS = not recorded
LDFLAGS_EX = not recorded
LDFLAGS_SL = not recorded
LIBS = not recorded
VERSION = PostgreSQL 10.1
```

- インストーラからのインストレーションを実施した場合(C:\PostgreSQL_Installにインストール)

```
> pg_config
BINDIR = C:/POSTGR~2/bin
DOCDIR = C:/POSTGR~2/doc
HTMLDIR = C:/POSTGR~2/doc
INCLUDEDIR = C:/POSTGR~2/include
PKGINCLUDEDIR = C:/POSTGR~2/include
INCLUDEDIR-SERVER = C:/POSTGR~2/include/server
LIBDIR = C:/POSTGR~2/lib
PKGLIBDIR = C:/POSTGR~2/lib
LOCALEDIR = C:/POSTGR~2/share/locale
MANDIR = C:/PostgreSQL_Install/man
SHAREDIR = C:/POSTGR~2/share
SYSCONFDIR = C:/PostgreSQL_Install/etc
PGXS = C:/PostgreSQL_Install/lib/pgxs/src/makefiles/pgxs.mk
CONFIGURE = --enable-thread-safety --enable-nls --with-ldap --with-openssl --with-ossp-uuid -
-with-libxml --with-libxslt --with-icu --with-tcl --with-perl --with-python
CC = not recorded
CPPFLAGS = not recorded
CFLAGS = not recorded
CFLAGS_SL = not recorded
LDFLAGS = not recorded
LDFLAGS_EX = not recorded
LDFLAGS_SL = not recorded
LIBS = not recorded
VERSION = PostgreSQL 10.1
```

インストレーション時のカスタマイズ可否の差異（pdf形式）

3.3.3. contribのWindows対応状況

3.3.3.1. Windowsでのcontirbモジュールの使用方法

ソースからのインストレーションを実施した場合には、Installコマンド実行時に指定したインストール先ディレクトリ配下に、インストーラからインストレーションを実施した場合

には、インストーラ内で指定したインストール先ディレクトリ配下に 実行ファイルやDLLファイルが生成されます。

- 実行ファイル形式のcontribモジュールの場合
    - 例えば、pgbenchの場合は、[インストール先ディレクトリ]\bin 配下に実行ファイル(pgbench.exe)が生成されています。
- ライブラリ形式のcontribモジュールの場合
    - 例えば、pg_stat_statementsの場合は[インストール先ディレクトリ]\lib 配下にDLLファイル(pg_stat_statements.dll)が生成されています。postgresql.confのshared_preload_librariesに「pg_stat_statements」を設定することで利用可能になります。

3.3.3.2. 対応状況の一覧表

以下の条件を満たす場合に、「Windows対応状況」列を「○」としています。

- ビルド時にvcxprojファイルが生成され、[インストール先ディレクトリ]\lib配下に配置される場合

contribのWindows対応状況（pdf形式）

3.3.4. 周辺ツールのWindows対応状況

2015年の成果物「2015 年度 WG3 活動報告書 データベースツール編」より選定したツールの Windows対応状況を一覧にしています。「Windows対応状況」列の凡例は、リリースノート等の公式文書内の記述を元にしており、以下の基準で記載しています。

- ○：動作すること、動作のさせ方が明記されている場合
- △：動作可否の明確な記述はないが、ツールの性格(実装されている言語など)から動作する可能性がある場合
- ×：動作しないことが明記されている場合

周辺ツールのWindows対応状況（pdf形式）

3.4. まとめ

3.4.1. インストレーションについて

今年度の検証でのまとめを以下に記述します。

- ソースからのインストレーション、インストーラからのインストレーションの2種類の方法について動作検証を実施しました。
- ソースからのインストレーションには、PostgreSQLをビルドするための環境として、UNIXライクな環境ではなくVisual Studio 2017 Expressを使用したWindowsネイティブな環境を使用しました。

3.4.1.1. インストレーションでの留意点

ソースからのインストレーション、インストーラからのインストレーション共に、インストール先のディレクトリとして「Program Files」等の、スペースの含まれたパスを指定すると、PostgreSQLのコマンドを利用するサードパーティのソフトウェアの動作に影響を与える可能性があります。可能な限り、避けることをお勧めします。

3.4.2. インストレーションのカスタマイズ可否および各種ツールのWindowsでの対応状況

今年度の検証でのまとめを以下に記述します。

- インストーラからのインストールによって構築できる内容を示しました。
    - ソリューションや業務パッケージへの組み込み用途など、インストレーションをカスタマイズしたい場合には、ソースからのインストレーションを検討ください。
    - ただし、UNIX環境でのインストレーションと同様のカスタマイズはできない場合があることにご注意ください。
- contribモジュールについてはWindows環境でも利用可能なように、インストレーション実行時に必要なモジュールがインストールされます。
- PostgreSQL本体に含まれない周辺ツールについては、Windowsで使用可能なものは限定的です。そのため、データベースの構築先としてWindowsの要件がある場合には、要件を満たす代替手段の検討画必要になる場合があります。

4. PostgreSQLの運用手順確認

本章では、Linux環境で一般的に実施するPostgreSQLのユーティリティコマンドがWindows環境で正しく動作するか、また差異がないかといった観点で確認を行いました。確認した項目は下記の4項目です。

- WAL書き出し先を変更してデータベースクラスタを初期化する
- PostgreSQLを起動・停止する
- PostgreSQLサーバのログ書き出しをEventLogにする
- PostgreSQLをバックアップ・リストアする

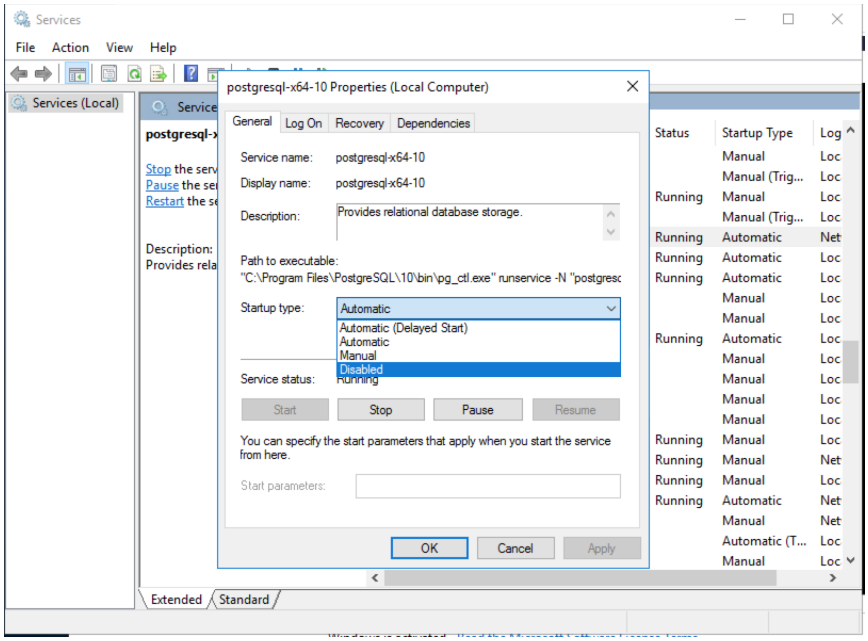
4.1. WAL書き出し先を変更したデータベースクラスタの初期化

信頼性や性能の向上を目的にWALの書き出し先を変更することがあります。本節では、Windows環境において変更する場合の手順を確認しまとめます。

4.1.1. 事前準備

インストーラでインストールした場合、デフォルトでデータベースクラスタが作成され、自動起動するようになっています。まずは、起動しているPostgreSQLを停止し、自動起動しないように設定変更を行います。

WindowsのサービスからPostgreSQLサービスの設定を開きます。



「Service status:」にある「Stop」ボタンを押すと、起動しているPostgreSQLが停止します。

また、「Startup type:」で「Disabled」を選択することで自動起動をオフにします。

4.1.2. initdbの実行

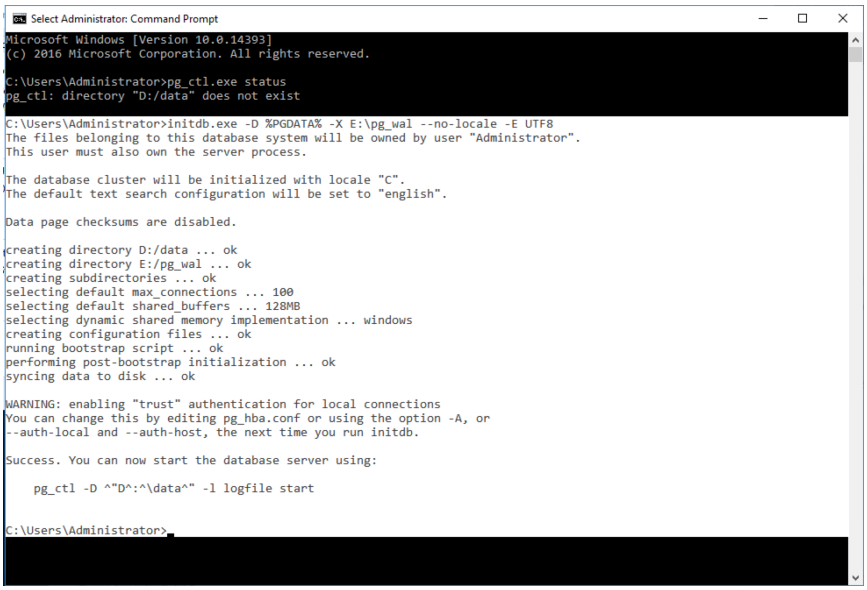
事前準備が整ったら、initdbコマンドでデータベースクラスタを初期化します。

なお、検証で用いた環境は以下の構成になっています。

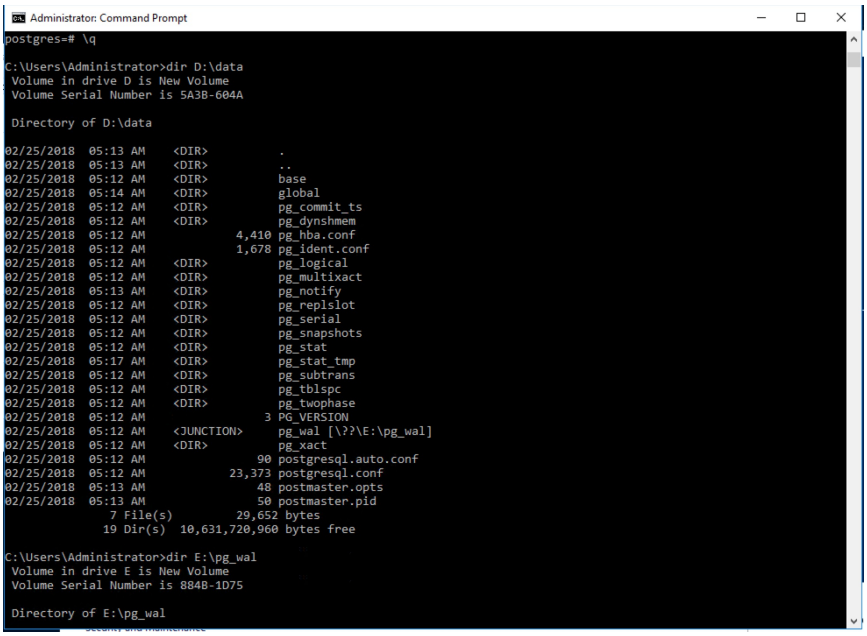
ドライブ名	用途	ディレクトリ
Cドライブ	PostgreSQLのバイナリ	C:\Program Files\PostgreSQL\10 (*)
	アーカイブWAL領域	C:\arc
Dドライブ	データベースクラスタ	D:\data
Eドライブ	WAL領域	E:\pgwal

(*)インストーラからのインストレーションで指定したディレクトリです。

Linux環境同様、initdbコマンドの-Xオプションで WAL書き出し先のディレクトリを指定することで WAL書き出し先を変更できます。

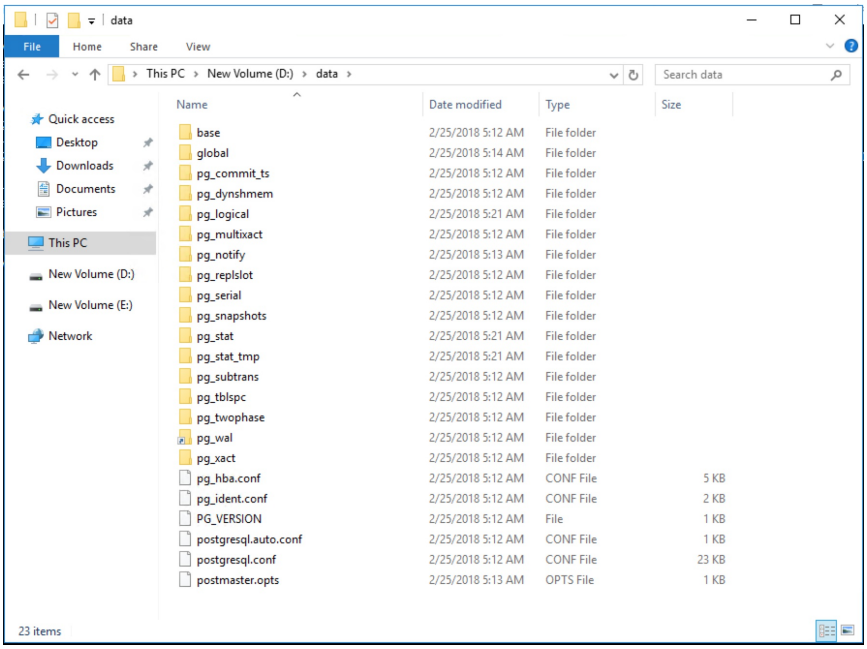


初期化後、データベースクラスタ配下のファイル・フォルダを確認すると、次のようになっています。



Linux環境では、pg_walがシンボリックリンクとして作成されますが、Windows環境では「JUNCTION」として作成されます。

Explorerから確認すると、次のようにフォルダに矢印がついた形で見えます。



実現方式の違いはありますが、Windows環境でもinitdbコマンドでWALの書き出し先を変更できることが確認できました。

4.2. PostgreSQLの起動と停止

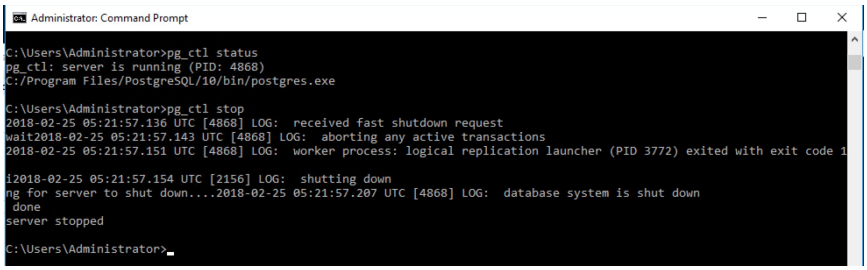
本節ではpg_ctlコマンドを用いたPostgreSQLの起動・停止を確認します。

4.2.1. PostgreSQLの起動

「3.2.4 動作確認」で示したように、必要な環境変数を設定していればLinux環境同様に「pg_ctl start」で起動できます。

4.2.2. PostgreSQLの停止

PostgreSQLの停止についても、必要な環境変数を設定していれば「pg_ctl stop」で停止できます。



また、「pg_ctl status」での状態確認も問題なく行えます。

上記の通り、pg_ctlコマンドを用いてPostgreSQLの起動・停止が行えることを確認しました。

4.3. PostgreSQLのイベントログ出力

本節では、Windows環境固有のログ出力であるイベントログへの出力について確認します。

ログの出力先はpostgresql.confのlog_destinationパラメータで変更できます。

4.3.1. syslog出力時の挙動

Linux環境では「log_destination = 'syslog'」と設定することでシステムログにPostgreSQLサーバのログを出力し、一元管理できます。

Windows環境で「log_destination = 'syslog'」を指定するとどのような挙動をとるのか確認しました。

```
#cursor_tuple_fraction = 0.1            # range 0.0-1.0
#from_collapse_limit = 8
#join_collapse_limit = 8                # 1 disables collapsing of explicit
                                        # JOIN clauses
#force_parallel_mode = off

#------------------------------------------------------------------------------
# ERROR REPORTING AND LOGGING
#------------------------------------------------------------------------------

# - Where to Log -

#log_destination = 'stderr'             # Valid values are combinations of
log_destination = 'stderr,syslog'               # Valid values are combinations of
                                        # stderr, csvlog, syslog, and eventlog,
                                        # depending on platform.  csvlog
                                        # requires logging_collector to be on.

# This is used when logging to stderr:
#logging_collector = off                # Enable capturing of stderr and csvlog
                                        # into log files.  Required to be on for
                                        # csvlogs.
                                        # (change requires restart)
```

上記のようにlog_destinationにsyslogを含めた場合、PostgreSQL起動時に次のようなエラーが起こり、PostgreSQLの起動に失敗します。

```
C:\Users\Administrator>pg_ctl start
waiting for server to start....2018-02-25 05:26:15.050 GMT [2308] LOG:  invalid value for parameter "log_destination": "
stderr,syslog"
2018-02-25 05:26:15.050 GMT [2308] DETAIL:  Unrecognized key word: "syslog".
2018-02-25 05:26:15.051 UTC [2308] FATAL:  configuration file "D:/data/postgresql.conf" contains errors
 stopped waiting
pg_ctl: could not start server
Examine the log output.
```

「Unrecognized key word: "syslog".」とあるように、Windows環境ではキーワードとしても"syslog"は受け付けないようです。

代わりに「log_destination = 'eventlog'」に設定することでイベントログへの出力がなされるようになります。

4.3.2. イベントログの確認

PostgreSQLをインストーラからインストールした場合、デフォルトのイベントソースとして"PostgreSQL"というエントリが登録されます。

"PostgreSQL"という値はevent_sourceパラメータのデフォルト値でもあるため、インストール後にlog_destinationの設定を変えるだけで、次のようにイベントビューアでイベントログを確認できます。

Application　Number of events: 120 (!) New events available

Level	Date and Time	Source	Event ID	Task Category
Error	2/25/2018 5:38:31 AM	PostgreSQL	0	None
Error	2/25/2018 5:38:19 AM	PostgreSQL	0	None
Error	2/25/2018 5:28:00 AM	PostgreSQL	0	None
Information	2/25/2018 5:27:52 AM	PostgreSQL	0	None
Information	2/25/2018 5:27:52 AM	PostgreSQL	0	None
Information	2/25/2018 5:27:52 AM	PostgreSQL	0	None
Information	2/25/2018 5:27:52 AM	PostgreSQL	0	None

event_sourceパラメータはpostgresql.confで変更可能です(もしくは起動時のオプションで変更可能)。

```
# These are relevant when logging to syslog:
#syslog_facility = 'LOCAL0'
#syslog_ident = 'postgres'
#syslog_sequence_numbers = on
#syslog_split_messages = on

# This is only relevant when logging to eventlog (win32):
# (change requires restart)
#event_source = 'PostgreSQL'
event_source = 'PGECons'
```

しかし、変更後の値(上記例では"PGECons")が、イベントソースのエントリとして登録されていないためこれだけではイベントログは出力されません。

イベントログを出力するためには、次のようにregsvr32コマンドを用いてイベントソースのエントリを登録します。

```
C:\Users\Administrator>regsvr32 /n /i:PGECons "C:\Program Files\PostgreSQL\10\lib\pgevent.dll"
```

こうすることで、イベントビューアに"PGECons"というイベントソースを持つイベントログが出力されるようになります。

Application　Number of events: 131

Level	Date and Time	Source	Event ID	Task Category
Error	2/25/2018 5:43:54 AM	PGECons	0	None
Error	2/25/2018 5:41:03 AM	PGECons	0	None
Information	2/25/2018 5:40:50 AM	PGECons	0	None
Information	2/25/2018 5:40:50 AM	PGECons	0	None
Information	2/25/2018 5:40:50 AM	PGECons	0	None
Information	2/25/2018 5:40:50 AM	PGECons	0	None
Information	2/25/2018 5:40:44 AM	PostgreSQL	0	None
Information	2/25/2018 5:40:43 AM	PostgreSQL	0	None
Information	2/25/2018 5:40:43 AM	PostgreSQL	0	None
Information	2/25/2018 5:40:43 AM	PostgreSQL	0	None
Information	2/25/2018 5:40:43 AM	PostgreSQL	0	None

上記の通り、Windows環境固有のイベントログ出力の設定方法および動作確認を行いました。

4.4. PostgreSQLのバックアップ・リストア

本節では、PostgreSQLのバックアップ、リストアについて「論理バックアップとリストア」、「物理バックアップとリカバリ」の2つを確認しました。

4.4.1. 論理バックアップとリストア

まずは、pg_dumpコマンドを用いた論理バックアップとpg_restoreコマンドを用いたリストアについて確認します。

事前準備としてtestdbデータベースを作成し、tblテーブルに1000件のデータを挿入しています。その後、pg_dumpコマンドを-Fc(カスタム形式)をつけて実行します。

```
C:\Users\Administrator>createdb testdb

C:\Users\Administrator>psql testdb
psql (10.2)
Type "help" for help.

testdb=# create table tbl(i int);
CREATE TABLE
testdb=# insert into tbl values (generate_series(1,1000));
INSERT 0 1000
testdb=# table tbl limit 10;
 i
----
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
(10 rows)


testdb=#
testdb=# \q

C:\Users\Administrator>pg_dump -Fc testdb > D:\pg_dumpFc_testdb.dmp
```

pg_restoreコマンドでカスタム形式のバックアップファイルを確認するとLinux環境同様にSQL文が列挙されていました。

```
C:\Users\Administrator>pg_restore D:\pg_dumpFc_testdb.dmp
--
-- PostgreSQL database dump
--

-- Dumped from database version 10.2
-- Dumped by pg_dump version 10.2

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;
SET row_security = off;

--
-- Name: plpgsql; Type: EXTENSION; Schema: -; Owner:
--

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;


--
-- Name: EXTENSION plpgsql; Type: COMMENT; Schema: -; Owner:
--

COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL procedural language';


SET search_path = public, pg_catalog;

SET default_tablespace = '';

SET default_with_oids = false;
```

新たにデータベースを作成し、バックアップファイルをpg_restoreコマンドでリストアすることで問題なくバックアップ取得時の状態を再現できました。

```
C:\Users\Administrator>createdb testdb

C:\Users\Administrator>dropdb testdb

C:\Users\Administrator>createdb testdb

C:\Users\Administrator>pg_restore -d testdb D:\pg_dumpFc_testdb.dmp

C:\Users\Administrator>psql -c "select count(*) from tbl" testdb
 count
-------
  1000
(1 row)
```

4.4.2. 物理バックアップとリカバリ

次に、オンライン物理バックアップとPITR(Point In Time Recovery)の動作を確認します。

まずはCドライブ直下にアーカイブWALを格納するディレクトリを作成します。

```
C:\Users\Administrator>mkdir C:\arc
```

postgresql.confでWALアーカイブの設定を行います。具体的には、archive_modeをonにし、archive_commandを設定します。archive_commandではcopyコマンドを用いてWALファイルをアーカイブWAL領域へとコピーするように設定しています。

なお、archive_commandで「"%p"」などのようにパスの部分をダブルクォテーションでくくらないと正しくarchive_commandが実行されませんでした。

```
# - Archiving -

#archive_mode = off             # enables archiving; off, on, or always
archive_mode = on               # enables archiving; off, on, or always
                                # (change requires restart)
#archive_command = ''           # command to use to archive a logfile segment
archive_command = 'copy "%p" "C:\arc\%f"'       # command to use to archive a logfile segment
                                # placeholders: %p = path of file to archive
                                #               %f = file name only
                                # e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
#archive_timeout = 0            # force a logfile segment switch after this
                                # number of seconds; 0 disables
```

設定が終わったら、PostgreSQLを再起動します。

```
C:\Users\Administrator>pg_ctl restart
2018-02-25 12:23:43.940 UTC [960] LOG:  received fast shutdown request
wai2018-02-25 12:23:43.948 UTC [960] LOG:  aborting any active transactions
2018-02-25 12:23:43.957 UTC [960] LOG:  worker process: logical replication launcher (PID 2752) exited with exit code 1
t2018-02-25 12:23:43.962 UTC [3084] LOG:  shutting down
ing for server to shut down....2018-02-25 12:23:44.032 UTC [960] LOG:  database system is shut down
 done
server stopped
waiting for server to start....2018-02-25 12:23:44.197 UTC [4920] LOG:  listening on IPv6 address "::1", port 5432
2018-02-25 12:23:44.201 UTC [4920] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2018-02-25 12:23:44.294 UTC [2972] LOG:  database system was shut down at 2018-02-25 12:23:43 UTC
2018-02-25 12:23:44.560 UTC [4920] LOG:  database system is ready to accept connections
 done
server started
```

再起動が完了したら、すべての準備が整いましたので、ベースバックアップの取得を行います。次の例では、pg_basebackupコマンドを用いてベースバックアップを取得しています。

```
C:\Users\Administrator>pg_basebackup -D D:\data_bkup -P
        1 file(s) copied.
31329/31329 kB (100%), 1/1 tablespace
        1 file(s) copied.
        1 file(s) copied.
```

PITRの挙動確認のため、追加で1000件のデータを挿入します。また、この時のトランザクションIDを記憶しておき、後ほどこの時点まで復旧するようにします。次の例では手順ミスを想定してtbl1テーブルのデータをすべて削除しています。また、以降の手順を簡単にするため、この時までのWALがすべてアーカイブされるようpg_switch_wal関数を利用しました。

```
C:\Users\Administrator>psql -c "insert into tbl values (generate_series(1001,2000));" testdb
INSERT 0 1000

C:\Users\Administrator>psql -c "select txid_current();" testdb
 txid_current
--------------
          590
(1 row)



C:\Users\Administrator>psql -c "delete from tbl;" testdb
DELETE 2000

C:\Users\Administrator>psql -c "select pg_switch_wal();" testdb
        1 file(s) copied.
 pg_switch_wal
---------------
 0/7039628
(1 row)
```

ここからPITRを行っていきます。

まずはPostgreSQLを停止し、現行のデータベースクラスタを退避させています。その後、次のようにxcopyコマンドを用いて取得したベースバックアップをデータベースクラスタとしてリストアします。

ただし、これだけではWindows環境で権限が不足していたため、別途Explorerから所有者のフルアクセス権限を付与し直しました。

```
C:\Users\Administrator>xcopy /X /E D:\data_bkup %PGDATA%
Does D:\data specify a file name
or directory name on the target
(F = file, D = directory)? D
D:\data_bkup\backup_label
D:\data_bkup\pg_hba.conf
D:\data_bkup\pg_ident.conf
D:\data_bkup\PG_VERSION
D:\data_bkup\postgresql.auto.conf
D:\data_bkup\postgresql.conf
```

引き続き、recovery.confの設定を行います。

recovery.confのサンプルファイルであるrecovery.conf.sampleは「<インストール先>\share」配下にあります。次の例ではrecovery.conf.sampleをデータベースクラスタ配下にrecovery.confという名前でコピーしています。

```
C:\Users\Administrator>copy "C:\Program Files\PostgreSQL\10\share\recovery.conf.sample" D:\data\recovery.conf
        1 file(s) copied.
```

recovery.confに、アーカイブWALのリストア設定およびリカバリ完了時点を設定します。リストア設定はarchive_commandの設定同様にcopyコマンドを用いて、アーカイブWAL領域からWAL領域へとコピーするようにしています。

```
# It is important that the command return nonzero exit status on failure.
# The command *will* be asked for log files that are not present in the
# archive; it must return nonzero when so asked.
#
# NOTE that the basename of %p will be different from %f; do not
# expect them to be interchangeable.
#
#restore_command = ''          # e.g. 'cp /mnt/server/archivedir/%f %p'
restore_command = 'copy "C:¥¥arc¥¥%f" "%p"'          # e.g. 'cp /mnt/server/archivedir/%f %p'
#
#
```

リカバリ完了時点の設定は、先ほどの手順ミス前に確認したトランザクションID(590)をrecovery_target_xidに設定しています。

```
#recovery_target_name = ''       # e.g. 'daily backup 2011-01-26'
#
#recovery_target_time = ''       # e.g. '2004-07-14 22:39:00 EST'
#
#recovery_target_xid = ''
recovery_target_xid = '590'
#
#recovery_target_lsn = ''        # e.g. '0/70006B8'
#
#recovery_target_inclusive = true
```

最後にWAL領域を削除し、E:\pg_walへのジャンクションを作成します。ジャンクションはmklinkコマンドに"/J"オプションを付けることで作成できます。

```
C:\Users\Administrator>rmdir /S D:\data\pg_wal
D:\data\pg_wal, Are you sure (Y/N)? Y

C:\Users\Administrator>mklink /J D:\data\pg_wal E:\pg_wal
Junction created for D:\data\pg_wal <<===>> E:\pg_wal
```

いよいよPostgreSQLの再起動です。アーカイブWALのリストアおよびリカバリが行われます。リカバリ完了時点に到達すると、PostgreSQLは一時停止モードになります。この状態はPostgreSQLへの接続および参照クエリのみが実行できる状態です。

```
C:\Users\Administrator>pg_ctl start
pg_ctl: another server might be running; trying to start server anyway
waiting for server to start....2018-02-25 14:57:04.929 UTC [2984] LOG:  listening on IPv6 address "::1", port 5432
2018-02-25 14:57:04.932 UTC [2984] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2018-02-25 14:57:04.942 UTC [2984] PANIC:  could not open control file "global/pg_control": Permission denied
 stopped waiting
pg_ctl: could not start server
Examine the log output.

C:\Users\Administrator>pg_ctl start
pg_ctl: another server might be running; trying to start server anyway
waiting for server to start....2018-02-25 14:59:16.789 UTC [4788] LOG:  listening on IPv6 address "::1", port 5432
2018-02-25 14:59:16.792 UTC [4788] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2018-02-25 14:59:16.877 UTC [3228] LOG:  database system was interrupted; last known up at 2018-02-25 12:50:46 UTC
...2018-02-25 14:59:20.532 UTC [3228] LOG:  starting point-in-time recovery to XID 590
        1 file(s) copied.
2018-02-25 14:59:20.620 UTC [3228] LOG:  restored log file "000000010000000000000006" from archive
2018-02-25 14:59:20.918 UTC [3228] LOG:  redo starts at 0/6000028
2018-02-25 14:59:20.950 UTC [3228] LOG:  consistent recovery state reached at 0/60000F8
2018-02-25 14:59:21.042 UTC [4788] LOG:  database system is ready to accept read only connections
 done
server started

C:\Users\Administrator>        1 file(s) copied.
2018-02-25 14:59:21.155 UTC [3228] LOG:  restored log file "000000010000000000000007" from archive
2018-02-25 14:59:21.299 UTC [3228] LOG:  recovery stopping after commit of transaction 590, time 2018-02-25 12:52:33.333
649+00
2018-02-25 14:59:21.301 UTC [3228] LOG:  recovery has paused
2018-02-25 14:59:21.301 UTC [3228] HINT:  Execute pg_wal_replay_resume() to continue.
```

tblテーブルの内容を確認し、想定する時点(例では2000件のデータが存在する時点)までのリカバリが行えてるか確認します。

```
C:\Users\Administrator>psql -c "select count(*) from tbl;" testdb
 count
-------
  2000
(1 row)
```

問題がないようであれば、pg_wal_replay_resume関数を実行してリカバリを完了させます。

```
C:\Users\Administrator>psql -c "select pg_wal_replay_resume();" testdb
 pg_wal_replay_resume
----------------------

(1 row)



C:\Users\Administrator>2018-02-25 15:01:29.039 UTC [3228] LOG:  redo done at 0/7010938
2018-02-25 15:01:29.040 UTC [3228] LOG:  last completed transaction was at log time 2018-02-25 12:52:33.333649+00
The system cannot find the file specified.
2018-02-25 15:01:29.058 UTC [3228] LOG:  selected new timeline ID: 2
The system cannot find the file specified.
2018-02-25 15:01:29.221 UTC [3228] LOG:  archive recovery complete
2018-02-25 15:01:29.484 UTC [4788] LOG:  database system is ready to accept connections
       1 file(s) copied.
```

論理バックアップからのリストア、物理バックアップからのリカバリともにLinux環境と同様の挙動となることを確認しました。

4.5. まとめ

本章では、Linux環境とWindows環境でのPostgreSQLの運用手順の差異について確認しました。

本章で確認した範囲でいえば、細かい違いはあるものの、大筋の流れは差異なく実施できると言えます。

4.5.1. PostgreSQLとしての差異

・パラメータでパスを指定する際、バックスラッシュを多用するため、値の設定方法の差異（エスケープが必要など）には注意が必要です

4.5.2. OSの違いによる差異

・イベントログへの出力を行う場合は、regsvr32コマンドを利用するなど手順が異なります

・物理的なファイルを扱う際は、当然ながらWindows環境固有のコマンドを利用する必要があります

5. High-Availability

本章ではPgpool-IIを利用した Windows 環境におけるPostgreSQLのHigh-Availability(以下HA)構成について説明します。

5.1. 調査、検証の目的

Windows環境でHA構成を組む場合、Windows Server Failover Clustering (WSFC) 等を導入する事が一般的かと思います。しかし、WSFCを利用する場合は、共有ディスクが必要になります。

WSFCを用いずPgpool-IIをクラスタソフトウェアとして利用することで、Windowsに構築したストリーミングレプリケーション構成のPostgreSQLを管理でき、HA構成を実現できます。ただし、Pgpool-II自体はWindows未対応のため、Linux環境の構築が必要となります。

本章ではPgpool-IIを利用したHA構成の環境構築・運用手順を説明し、本構成でできる事・できない事を明らかにします。なお、Pgpool-IIのフェイルオーバー等Linux環境に閉じた部分は、本章では紹介しません。

5.2. 環境構築手順

5.2.1. 本章における検証環境の構成

検証環境の概略図は以下になります。

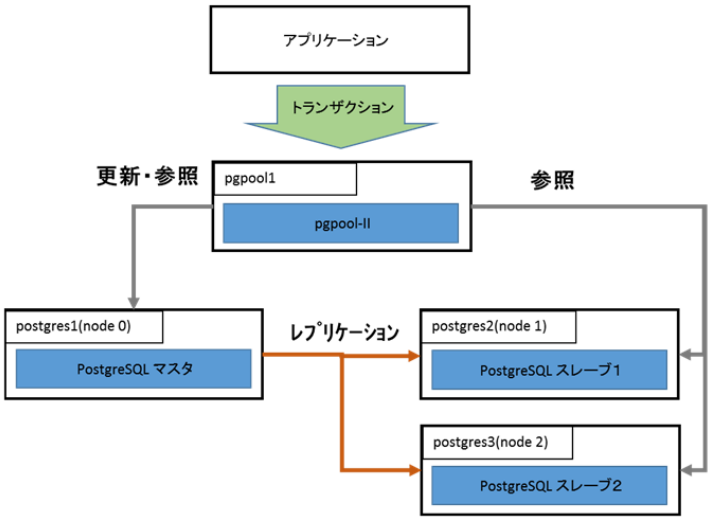


図 5.1 検証環境構成図

アプリケーションからのトランザクションは、全てLinuxサーバ(pgpool1)に構築されたPgpool-IIに投入されます。Pgpool-IIは3台のWindowsサーバ上に構築されたPostgreSQLを管理します。

Windows 環境で follow master が実施可能か検証するため、PostgreSQLは3台設定します。

更新処理はマスタDB(上図の場合はpostgres1(node 0))に投入され、ストリーミングレプリケーションによって他の2台のDBサーバへ反映されます。

参照処理はPgpool-IIのロードバランス機能によって、3台のDBサーバに割り振られます。

各サーバの詳細は下表の通りです。

表 5.1 サーバ詳細

項番	サーバ名	IPアドレス	OSバージョン	用途
1	pgpool1	192.168.1.231	CentOS 7.2 (x64)	Pgpool-II 実行環境
2	postgres1	192.168.1.228	Windows Server 2016 standard (x64)	PostgreSQL マスタ (Pgpool-IIにおけるnode 0)
3	postgres2	192.168.1.229	Windows Server 2016 standard (x64)	PostgreSQL スレーブ1 (Pgpool-IIにおけるnode 1)
4	postgres3	192.168.1.230	Windows Server 2016 standard (x64)	PostgreSQL スレーブ2 (Pgpool-IIにおけるnode 2)

また、各サーバにインストールするソフトウェアと設定概要は下表の通りです。

表 5.2 インストールするソフトウェアと設定概要

項番	ソフトウェア名	インストール先サーバ名	ソフトウェアバージョン	インストール先	実行環境等
1	Pgpool-II	pgpool1	3.7.0	/usr/pgpool-10	設定ファイル配置先：/etc/pgpool-II-10
2	PostgreSQL	postgres1,postgres2,postgres3	10.1 (64ビット版 Windows)	C:\PostgreSQL\10	DBクラスタ配置先：C:\PostgreSQL\10\data

5.2.2. 構築準備

HA構成の構築を始める前にシステムの基礎的な環境を整える必要があります。ここでは下記3点について説明します。

1. failoverやfollow master 実施時にPgpool-IIが起動するシェルの中で、ホスト名で名前解決できるようにするため、hostsファイルを設定します。
2. Pgpool-IIが起動するシェルの中でssh通信を行う際に、パスワードが要求される事により処理が停止する事態を回避するため、pgpass.confを設定します。
3. Pgpool-IIと各DBサーバ間で通信するため、Windows環境にSSHを設定します。

各サーバのホスト名を登録します。

Linux環境

```
[root@pgpool1 ~]# cat /etc/hosts
192.168.1.231  pgpool1
192.168.1.230  postgres3
192.168.1.229  postgres2
192.168.1.228  postgres1
```

Windows環境

```
C:\PostgreSQL\10\data>type C:\Windows\System32\drivers\etc\hosts
192.168.1.231  pgpool1
192.168.1.230  postgres3
192.168.1.229  postgres2
192.168.1.228  postgres1
```

follow master の処理等でPostgreSQLのパスワード入力を避けるため Windows環境にpgpass.conf を設定します。

```
C:\PostgreSQL\10\data>type "C:\Users\Administrator\AppData\Roaming\postgresql\pgpass.conf"
*:*:*:postgres:postgres
*:*:replication:postgres:postgres
```

Pgpool-IIでPostgreSQLを制御するにはssh経由で行いますが、デフォルトでsshはインストールされていません。sshはマイクロソフトがプレリリース版を出しているので、そちらを利用するため設定を行いたいと思います。

Windowsのsshは下記サイトよりダウンロードします。

https://github.com/PowerShell/Win32-OpenSSH/releases

※上記サイトのリリースノートに「This is a pre-release (non-production ready)」との表記がありますので、利用するにあたってはご注意ください。

ダウンロードファイルはzip形式になっています。このzipファイルを展開したら、PowerShellを **管理者モード** で起動して、上記zipファイルの展開によって生成されたOpenSSH-Win64フォルダに移動します。

移動した先で下記のようにコマンドを実行し、SSHをインストールします。下記の実行例は、Cドライブ直下に展開したフォルダを配置した場合の例になります。

```
PS C:\OpenSSH-Win64> powershell -ExecutionPolicy Bypass -File install-sshd.ps1
[SC] SetServiceObjectSecurity SUCCESS
[SC] ChangeServiceConfig SUCCESS
[SC] ChangeServiceConfig2 SUCCESS
sshd and ssh-agent services successfully installed
```

上記のように「sshd and ssh-agent services successfully installed」が表示されれば、sshd及びssh-agentのインストールは完了です。次に、FirewallがSSHの通信を許可するように設定を変更します。引き続きPowerShellで以下のコマンドを実行します。

```
PS C:\OpenSSH-Win64> New-NetFirewallRule -Protocol TCP -LocalPort 22 -Direction Inbound -Action
Allow -DisplayName OpenSSH -Program C:\OpenSSH-Win64\sshed.exe
Name                  : {9f880921-f3a0-41d4-973c-4ac0c4b081db}
（中略）
PrimaryStatus         : OK
Status                : 規則は、ストアから正常に解析されました。 (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType : Local
```

上記のように「Status：規則は、ストアから正常に解析されました。」が確認できたら、firewall設定は完了です。念のため設定状況を確認します。

```
PS C:\Users\Administrator> Get-NetFirewallRule -DisplayName OpenSSH


Name                  : {9f880921-f3a0-41d4-973c-4ac0c4b081db}
DisplayName           : OpenSSH
Description           :
DisplayGroup          :
Group                 :
Enabled               : True
Profile               : Any
Platform              : {}
Direction             : Inbound
Action                : Allow
EdgeTraversalPolicy   : Block
LooseSourceMapping    : False
LocalOnlyMapping      : False
Owner                 :
PrimaryStatus         : OK
Status                : 規則は、ストアから正常に解析されました。 (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType : Local
```

上記のように「DisplayName：OpenSSH」の「Action：Allow」であり、「PrimaryStatus：OK」となっていれば、SSHの通信が許可されている事になります。

次にホストの認証キーを生成するため、下記のコマンドを実行します。

```
PS C:\OpenSSH-Win64> .\ssh-keygen.exe -A
C:\OpenSSH-Win64\ssh-keygen.exe: generating new host keys: RSA DSA ECDSA ED25519
```

このコマンドを実行した結果、下記のファイルが作成されます。

```
PS C:\OpenSSH-Win64> dir ssh_host_*


    ディレクトリ: C:\OpenSSH-Win64


Mode         LastWriteTime                 Length Name
----         -------------                 ------ ----
-a----       2018/01/31     14:59            672 ssh_host_dsa_key
-a----       2018/01/31     14:59            624 ssh_host_dsa_key.pub
-a----       2018/01/31     14:59            227 ssh_host_ecdsa_key
-a----       2018/01/31     14:59            196 ssh_host_ecdsa_key.pub
-a----       2018/01/31     14:59            432 ssh_host_ed25519_key
-a----       2018/01/31     14:59            116 ssh_host_ed25519_key.pub
-a----       2018/01/31     14:59           1679 ssh_host_rsa_key
-a----       2018/01/31     14:59            416 ssh_host_rsa_key.pub
```

公開鍵認証を有効化するため C:\OpenSSH-Win64\sshd_config に記載されている PubkeyAuthentication のコメントアウトを外します。

```
#PubkeyAuthentication yes
↓
PubkeyAuthentication yes
```

次にSSHクライアントとしての秘密鍵と公開鍵のペアを作成するために、ssh-keygenコマンドを実行します。このコマンドでは、下記実行例の「※※」で示す通り、3か所で入力を求められます。

1回目の入力では、ファイルの出力先の指定を求められます。デフォルトの場合は、何も入力せずにEnterのみ入力します。

2回目の入力では、生成する鍵のパスフレーズの入力を求められます。本環境では公開鍵認証によりパスフレーズなしでログインできるようにするため、ここでも何も入力せずにEnterのみ入力します。

3回目にもう一度同じパスフレーズ（何も入力しない）を入力します。

```
PS C:\OpenSSH-Win64> .\ssh-keygen.exe -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\Administrator/.ssh/id_rsa): ※※ 空Enterを投入
Created directory 'C:\Users\Administrator/.ssh'.
Enter passphrase (empty for no passphrase):                               ※※ 空Enterを投入
Enter same passphrase again:                                              ※※ 空Enterを投入
Your identification has been saved in C:\Users\Administrator/.ssh/id_rsa.
Your public key has been saved in C:\Users\Administrator/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:2kCQtvTYgLHfZgFHFl62UBSJf+fVQEVIRgJkltOdv0k administrator@postgres1@postgres1
The key's randomart image is:
+---[RSA 2048]----+
|  .ooo*=*+=+.===o|
|  ..=*.+.+o .o=  |
|  .o *+..  .   + |
|   .ooo.. . . .Eo|
|    . = S. o .. o|
|     o +    .  o |
|      . .        |
|                 |
|                 |
+----[SHA256]-----+
```

このコマンドによって、C:\Users\Administrator\.ssh\id_rsa フォルダに id_rsa と id_rsa.pub が作成されます。

上記で生成された鍵のファイルパーミッションを設定するために FixUserFilePermissions.ps1 を実行します。

```
PS C:\OpenSSH-Win64> cd C:\Users\Administrator\.ssh
PS C:\Users\Administrator\.ssh> Powershell -executionpolicy bypass -File C:\Openssh-
Win64\FixUserFilePermissions.ps1
  [*] C:\Users\Administrator\.ssh\id_rsa
      looks good

  [*] C:\Users\Administrator\.ssh\id_rsa.pub

'Everyone' has the following access to 'C:\Users\Administrator\.ssh\id_rsa.pub': 'ReadAndExecute,
Synchronize'.
Shall I make it Read only?
[Y] はい(Y)  [A] すべて続行(A)  [N] いいえ(N)  [L] すべて無視(L)  [S] 中断(S)  [?] ヘルプ (既定値は "Y"): ※※ 空
Enterを投入
'Everyone' now has Read access to 'C:\Users\Administrator\.ssh\id_rsa.pub'.
      Repaired permissions

   Done.
```

ここでも入力を求められる場合がありますが、これらのファイルはRead only であるため、空Enter や「Y」で応答します。

クライアントの鍵を authorized_keys に登録します。ここまでの状況ではまだ authorized_keys が存在していないので、id_rsa.pub を コピーして作成します。

```
PS C:\Users\Administrator\.ssh> copy id_rsa.pub authorized_keys

PS C:\Users\Administrator\.ssh>
```

他のサーバの公開鍵を登録する際は、上記のauthorized_keysにid_rsa.pubの内容を追記します。各マシンのauthorized_keys には4台のサーバの公開鍵の情報が登録されるようにします。

最後に authorized_keys のアクセス権を設定するため、FixHostFilePermissions.ps1 を実行しますが、その前にPowerShellの実行ポリシーを変更します。プロンプトの入力要求には「Y」を入力します。

```
PS C:\Users\Administrator\.ssh> Set-ExecutionPolicy bypass

実行ポリシーの変更
実行ポリシーは、信頼されていないスクリプトからの保護に役立ちます。実行ポリシーを変更すると、about_Execution_Policies
のヘルプ トピック (http://go.microsoft.com/fwlink/?LinkID=135170)
で説明されているセキュリティ上の危険にさらされる可能性があります。実行ポリシーを変更しますか?
[Y] はい(Y)  [A] すべて続行(A)  [N] いいえ(N)  [L] すべて無視(L)  [S] 中断(S)  [?] ヘルプ (既定値は "N"): Y ※※ Y
で応答する。
```

PowerShellの実行ポリシーを変更したら FixHostFilePermissions.ps1 を実行します。

```
PS C:\Users\Administrator\.ssh> C:\OpenSSH-Win64\FixHostFilePermissions.ps1 -Confirm:$false
  [*] C:\OpenSSH-Win64\sshd_config
      looks good

  [*] C:\OpenSSH-Win64\ssh_host_dsa_key
'postgres1\Administrator' has no more access to 'C:\OpenSSH-Win64\ssh_host_dsa_key'.
'NT SERVICE\sshd' now has Read access to 'C:\OpenSSH-Win64\ssh_host_dsa_key'.
      Repaired permissions

(中略)

  [*] C:\Users\Administrator\.ssh\authorized_keys
'NT SERVICE\sshd' now has Read access to 'C:\Users\Administrator\.ssh\authorized_keys'.
      Repaired permissions

   Done.
```

sshdを起動して、接続確認を行います。

```
PS C:\Users\Administrator> Restart-Service sshd
PS C:\Users\Administrator>
```

sshd起動したサーバとは別のサーバからssh接続を試みます。以下の例はLinuxサーバ(pgpool1)からWindowsサーバ(postgres1)への接続例です。

```
[root@pgpool1 ~]# ssh Administrator@192.168.1.228

Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

administrator@postgres1 C:\Users\Administrator>hostname
postgres1
```

公開鍵認証により、パスワードなしでログインができ、コマンド実行できれば完了です。

5.2.3. 構築手順（全体像）

HA構成構築は以下のような流れで実施します。

1. Windows版 PostgreSQLのインストール
2. Pgpool-II環境構築
3. Windows版 PostgreSQLのレプリケーション環境構築
4. 上記のPostgreSQLをPgpool-IIへ登録
5. failover, follow master 設定

なおオンラインリカバリは、Pgpool-IIのpgpool-recoveryをWindows環境にインストールする術が現時点（2018年4月時点）でないため、今回は検証対象外としました。

5.2.4. Windows版 PostgreSQLのインストール

インストール手順は3章の「Windows環境でのインストーラからのインストレーション」をご参照ください。

5.2.5. Pgpool-II環境構築

Pgpool-IIの構築手順は下記サイトを参照してください。

https://lets.postgresql.jp/documents/technical/pgpool/2

今回の検証でpgpool.confに設定したPostgreSQLのノード関連情報は以下のようになります。

```
[root@pgpool1 pgpool-II-10]# grep -e backend_hostname -e backend_port \
>                                 -e backend_weight -e backend_data_directory \
>                                 -e backend_data_directory pgpool.conf
backend_hostname0 = '192.168.1.228'
backend_port0 = 5432
backend_weight0 = 0.3
backend_data_directory0 = 'C:/PostgreSQL/10/data'
backend_hostname1 = '192.168.1.229'
backend_port1 = 5432
backend_weight1 = 0.4
backend_data_directory1 = 'C:/PostgreSQL/10/data'
backend_hostname2 = '192.168.1.230'
backend_port2 = 5432
backend_weight2 = 0.3
backend_data_directory2 = 'C:/PostgreSQL/10/data'
```

5.2.6. Windows版 PostgreSQLのレプリケーション環境構築

Windows版 PostgreSQLのレプリカ環境作成します。

下表に、ホットスタンバイを利用したレプリカ環境を構築するために確認・設定したパラメータを記載します。

表 5.3 postgresql.confのデフォルトからの変更点

項番	パラメータ	設定値	パラメータの意味
1	listen_addresses	'*'	通信で使用するインターフェースを指定する。IPインターフェースを利用するので '*' とする。
2	wal_level	replica	WALに出力する情報。ホットスタンバイを利用するためにはreplicaでなければならない。
3	max_wal_senders	10	WAL送信プロセスの数。設定値はノード数に依存する。
4	hot_standby	on	ホットスタンバイを有効にするかどうか。利用するのでonとする。
5	logging_collector	on	ログ収集機構を有効にするかどうか。ログを取得するのでonとする。
6	archive_mode	on	完了したWALセグメントはアーカイブ格納領域に送信するかどうか。送信するのでonとするが、回復要件に従って設定を判断する。
7	archive_command	'copy "%p" "D:/PostgreSQL/archive/%f"'	完了したWALセグメントのアーカイブを実行するローカルのシェルコマンド。アーカイブ先のパスは環境に応じて適宜設定する。

次に、マスタDBサーバ postgres1(192.168.1.228) のpg_hba.confに下記の設定を追加します。(追記箇所は※の所)

```
C:\PostgreSQL\10\data>type pg_hba.conf
# PostgreSQL Client Authentication Configuration File
（中略）
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
host    all             all             192.168.1.0/24          md5 ※
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
host    replication     all             127.0.0.1/32            md5
host    replication     all             192.168.1.0/24          md5 ※
host    replication     all             ::1/128                 md5

C:\PostgreSQL\10\data>
```

pg_hba.confの編集が完了したら、マスタDBサーバ postgres1(192.168.1.228) のみ起動します。DBの起動はサービス画面から「postgresql-x64-10」をを選択して、サービスの開始をクリックします。

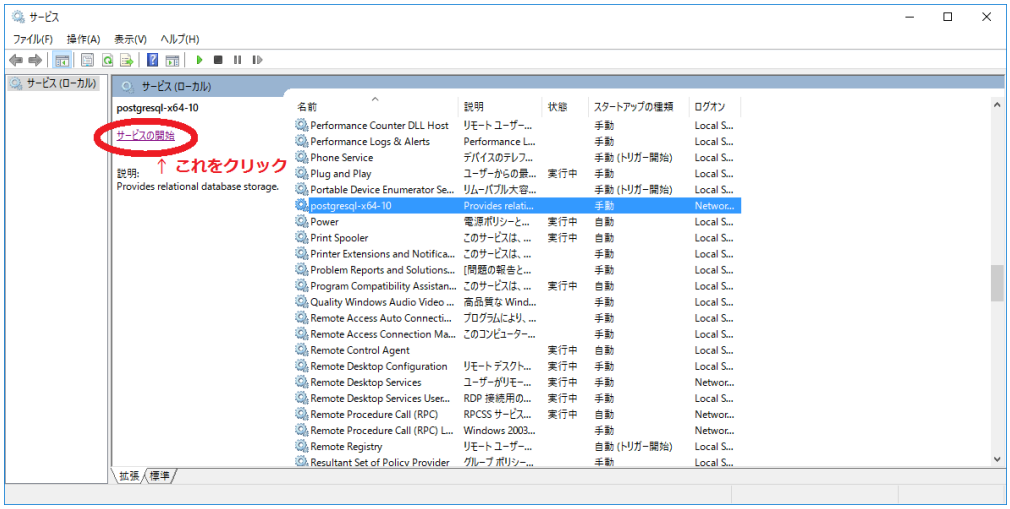


図 5.2 サービス開始のイメージ

マスタDBサーバの起動確認はpsqlで接続する等で確認できます。

```
C:\PostgreSQL\10\data>psql -U postgres
psql (10.1)
Type "help" for help.

postgres=#
```

マスタDBサーバが起動したら、2台のスレーブDBサーバでpg_basebackup を実行し、環境のコピーを行います。postgres2(192.168.1.229)、postgres3(192.168.1.230)でそれぞれ、以下のコマンドを実行します。

```
C:\PostgreSQL\10>pg_basebackup -h 192.168.1.228 -D ./data --verbose --progress -U postgres
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/D3000028 on timeline 3
pg_basebackup: starting background WAL receiver
3094044/3094044 kB (100%), 1/1 tablespace
pg_basebackup: write-ahead log end point: 0/D3000130
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: base backup completed

C:\PostgreSQL\10>
```

2台のスレーブでpg_basebackupが完了したら、マスタからコピーしたDBクラスタ(C:\PostgreSQL\10\data)に、下記のように記述されたrecovery.confを配置します。primary_conninfo にはマスタへの接続情報を記述するので、環境に合わせて変更してください。

```
standby_mode = 'on'
primary_conninfo = 'user=postgres password=postgres host=192.168.1.228 port=5432'
```

recovery.confの配置が完了したら、2台のスレーブDBサーバも起動します。起動方法は、マスタDBサーバの時と同様にサービスから起動します。スレーブDBサーバの起動が確認できたら、マスタDBサーバのコマンドプロンプトからpsql経由で下記SQLを実行し、レプリケーションが正しく行われている事を確認します。

```
postgres=# select * from pg_stat_replication;
 pid  | usesysid | usename  | application_name |  client_addr   | client_hostname | client_port |
backend_start         | backend_xmin |   state   |  sent_lsn  | write_lsn  | flush_lsn  |
replay_lsn | write_lag | flush_lag | replay_lag | sync_priority | sync_state
------+----------+----------+------------------+----------------+-----------------+-------------+--
-------------------------------+--------------+-----------+------------+------------+------------+---
---------+-----------+-----------+------------+---------------+------------
 1456 |       10 | postgres | walreceiver      | 192.168.1.230 |                 |       50328 |
2018-02-19 17:06:58.904175+09 |              | streaming | 0/D6000140 | 0/D6000140 | 0/D6000140 |
0/D6000140 |           |           |            |             0 | async
 3240 |       10 | postgres | walreceiver      | 192.168.1.229 |                 |       50482 |
2018-02-19 16:51:28.101627+09 |              | streaming | 0/D6000140 | 0/D6000140 | 0/D6000140 |
0/D6000140 |           |           |            |             0 | async
(2 rows)
```

上記SQLの結果より、client_addr が期待したIPアドレス(node 1及びnode 2)になっている事、state が streaming である事、sync_state が async である事を確認します。上記の例では、2台のスレーブDBサーバが設定されているので、pg_stat_replication の検索結果も2行表示されています。

5.2.7. Pgpool-IIへの登録

上記のPostgreSQLをPgpool-IIへの登録します。5.2.5のpgpool.confの設定により、監視対象のサーバ情報は定義されているので、Pgpool-IIを起動するだけで認識されます。下記のコマンド例では、Pgpool-IIのログは /var/log/pgpool/pgpool.log へ出力します。

```
[root@pgpool1 pgpool-II-10]# pgpool -n -d > /var/log/pgpool/pgpool.log 2>&1 &
[1] 22238
[root@pgpool1 pgpool-II-10]# psql -p9999 -Upostgres
Password for user postgres:
psql (10.1)
Type "help" for help.

postgres=# show pool_nodes;
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 0          | true              | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | standby | 0          | false             | 0
(3 rows)
```

Pgpool-IIの認識結果は show pool_nodes で確認できます。status が全て up になっている事、role は マスタDBサーバ(192.168.1.228)のみprimaryとなっていて、他2台は standby となっている事を確認します。

5.2.8. failover, follow master, 設定

本項ではfailover, follow masterの設定方法を説明します。

最初にLinux環境のpgpool.confにfailover及びfollow master 処理を記述するスクリプトの場所を定義します。pgpool.confには以下のように記述します。

```
failover_command = '/etc/pgpool-II-10/failover.sh %d %P %m %H %R'
follow_master_command = '/etc/pgpool-II-10/follow_master.sh %h %d %H %M %P'
```

次に、上記のパラメータで指定したシェルスクリプト本体をPgpool-IIサーバに配置します。

/etc/pgpool-II-10/failover.sh では下記のような処理を行います。

- promoteするノードの判定
- promoteするノードでの「pg_ctl promote」実行

下記にシェルの具体的な記述例を記載します。

```
[root@pgpool1 pgpool-II-10]# cat /etc/pgpool-II-10/failover.sh
#! /bin/sh
log=/var/log/pgpool/failover.log

failed_node_id=$1
old_primary_node_id=$2
new_master_id=$3
new_master_host=$4
new_master_dir=$5

# Do promote only when the primary node failes
if [ $failed_node_id = $old_primary_node_id ]; then
  echo "The primary node (node $failed_node_id) dies." >> $log
  echo "Node $new_master_id takes over the primary." >> $log

  ssh Administrator@$new_master_host -T '"C:/PostgreSQL/10/bin/pg_ctl" -D "C:/PostgreSQL/10/data"
promote' >> $log

#  source config_for_script.$new_master_id
#  ssh $PG_USER@$new_master_host -T '$PG_CTL -D $PG_DATA promote' >> $log

else
  echo "Node $failed_node_id which is not the primary dies. This script doesnt't anything." >> $log
fi
```

/etc/pgpool-II-10/follow_master.sh では下記のような処理を行います。

- follow master 実行ノードの判定
- follow master 実行ノードに対して、follow master 用バッチの実行
- follow master 用バッチの実行後の PostgreSQL 起動確認
- follow master 実行ノードを Pgpool-II へ attach

下記にシェルの具体的な記述例を記載します。

```
[root@pgpool1 ~]# cat /etc/pgpool-II-10/follow_master.sh
#!/bin/sh
log=/var/log/pgpool/follow_master.log

attach_node_host=$1
hostnode=$2
new_master_node_host=$3

if [ $hostnode != 2 ]; then
    exit 0
fi

TimeoutCount=10
CurrentCount=0

ssh Administrator@$attach_node_host -T "C:/PostgreSQL/10/do_followmaster.bat" $new_master_node_host
>> $log

ssh Administrator@$attach_node_host -T 'pg_ctl status -D "C:\PostgreSQL\10\data"' > /dev/null
while [ $? -ne 0 ]
do
        CurrentCount=`expr $CurrentCount + 1`
        sleep 30
        if [ $CurrentCount -gt $TimeoutCount ];then
                echo "$0 exit due to TimeoutCount exceeded." >> $log
                exit 0
        fi
        ssh Administrator@$attach_node_host -T 'pg_ctl status -D "C:\PostgreSQL\10\data"' >
/dev/null
done

pcp_attach_node $hostnode -U postgres
echo "$0 exit due to PostgreSQL@$attach_node_host started." >> $log

exit 0
```

続いて、/etc/pgpool-II-10/follow_master.sh で指定したバッチファイルを2台目のスレーブDBサーバ postgres3(192.168.1.230) に配置します。配置場所は follow_master.sh に記載した通り、C:/PostgreSQL/10 になります。

このバッチファイルでの処理概要は以下のようになります。

- pg_basebackup 取得前の PostgreSQL 関係プロセスの停止
- 新マスターから pg_basebackup でDBクラスタをコピー
- follow master 後のスレーブとして動作するための設定ファイル配置
- DBクラスタを格納したフォルダのパーミッション変更
- PostgreSQL を起動

下記にバッチファイルの具体的な記述例を記載します。

```
C:\PostgreSQL\10>type "C:/PostgreSQL/10/do_followmaster.bat"
@ECHO OFF
REM The script sets environment variables helpful for PostgreSQL

@SET PATH="C:\PostgreSQL\10\bin";%PATH%
@SET PGDATA="C:\PostgreSQL\10\data"
@SET PGDATABASE=postgres
@SET PGUSER=postgres
@SET PGPORT=5432
@SET PGLOCALEDIR=C:\PostgreSQL\10\share\locale

set PGBASEBACKUP="C:\PostgreSQL\10\bin\pg_basebackup.exe"
set PGCTL="C:\PostgreSQL\10\bin\pg_ctl.exe"
set COPYFROM="C:\PostgreSQL\10\taihi_files"
set LOG="C:/PostgreSQL/10/do_dollowmaster.log"
set newmaster=%1

taskkill /IM pg_basebackup.exe /F /T
%PGCTL% -D %PGDATA% -m immediate stop
rmdir /S /Q %PGDATA%
%PGBASEBACKUP% -h %newmaster% -D %PGDATA% -X stream -U postgres -w

copy %COPYFROM%\postgresql.conf %PGDATA%
copy %COPYFROM%\pg_hba.conf %PGDATA%
copy %COPYFROM%\follow_master_recovery.conf %PGDATA%\recovery.conf

icacls %PGDATA% /grant "NETWORK SERVICE:F" /T
net start postgresql-x64-10 >> %LOG%
```

failover 及び follow master の設定は以上になります。

5.3. Pgpool-II運用におけるPCPコマンドの動作確認

本節ではWindows上に構築したPostgreSQLに対して、PCPコマンドが利用可能かどうかを検証します。

Pgpool-II でサポートしているPCPコマンド及び、検証結果の概要は下表の通りになります。この内、pcp_watchdog_infoとpcp_recovery_nodeは今回の検証環境では実行できないため、検証の対象外とします。

表 5.4 PCPコマンド一覧

項番	コマンド名	機能説明	利用可否(○:可能,×:不可能)
1	pcp_node_count	Pgpool-II の pgpool.conf で定義されたノードの総数を表示する	○
2	pcp_node_info	指定されたノードの情報を表示する	○
3	pcp_watchdog_info	Pgpool-II の watchdog ステータスを表示します	対象外(watchdog を利用しないため)
4	pcp_proc_count	Pgpool-II の子プロセスのプロセス ID を一覧表示する	○
5	pcp_proc_info	Pgpool-II の子プロセス情報を表示する	○
6	pcp_pool_status	pgpool.conf のパラメータ設定値を取得する	○
7	pcp_detach_node	Pgpool-II からノードを切り離す	○
8	pcp_attach_node	Pgpool-II にノードを復帰させる	○
9	pcp_promote_node	Pgpool-II のノードをマスタに昇格させる	○
10	pcp_stop_pgpool	Pgpool-II を指定されたモードでシャットダウンする	○
11	pcp_recovery_node	Pgpool-II のノードのデータを再同期させた上で復帰させる	×(前提である pgpool-recoveryが2018年4月時点では Windows に対応していないため)

以下に具体的な実施例を示します。

5.3.1. pcp_node_count

本検証環境ではPgpool-IIに3ノード認識させているので、コマンドの実行結果は「3」と表示されます。

```
[root@pgpool1 ~]# pcp_node_count -Upostgres
Password:
3
[root@pgpool1 ~]#
```

5.3.2. pcp_node_info

pcp_node_infoは各ノード毎に実施します。

```
[root@pgpool1 ~]# pcp_node_info 0 -Upostgres
Password:
192.168.1.228 5432 2 0.300000 up primary
[root@pgpool1 ~]# pcp_node_info 1 -Upostgres
Password:
192.168.1.229 5432 2 0.400000 up standby
[root@pgpool1 ~]# pcp_node_info 2 -Upostgres
Password:
192.168.1.230 5432 2 0.300000 up standby
[root@pgpool1 ~]#
```

5.3.3. pcp_proc_count

Pgpool-IIのプロセスが表示されます。このコマンドによる表示結果はpsコマンドで表示されるPgpool-IIのプロセスIDと同じになります。

```
[root@pgpool1 pgpool-II-10]# pcp_proc_count -Upostgres
Password:
29818 29819 29820 29821 29822 29823 29824 29825 29826 29827 29828 29829 29830 29831 29832 29833
29834 29835 29836 29837 29838 29839 29881 29841 29842 29843 29844 29845 29846 29847 29848 29849
[root@pgpool1 pgpool-II-10]# ps -ef | grep pgpool
root      2896  2722  0 14:30 pts/0    00:00:00 grep --color=auto pgpool
root     29817     1  0  3月06 ?      00:00:07 pgpool -n -d -D
root     29818 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29819 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29820 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29821 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29822 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29823 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29824 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29825 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29826 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29827 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29828 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29829 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29830 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29831 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29832 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29833 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29834 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29835 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29836 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29837 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29838 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29839 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29841 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29842 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29843 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29844 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29845 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29846 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29847 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29848 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29849 29817  0  3月06 ?      00:00:00 pgpool: wait for connection request
root     29850 29817  0  3月06 ?      00:00:00 pgpool: PCP: wait for connection request
root     29851 29817  0  3月06 ?      00:02:17 pgpool: worker process
root     29852 29817  0  3月06 ?      00:00:43 pgpool: health check process(0)
root     29853 29817  0  3月06 ?      00:00:43 pgpool: health check process(1)
root     29854 29817  0  3月06 ?      00:00:43 pgpool: health check process(2)
root     29881 29817  0  3月06 ?      00:00:00 pgpool: postgres postgres  idle
```

5.3.4. pcp_proc_info

接続中のPgpool-IIプロセスの情報を表示します。

```
[root@pgpool1 pgpool-II-10]# pcp_proc_info -Upostgres -P 29881
Password:
postgres postgres 2018-03-06 12:05:36 2018-03-14 14:29:56 3 0 1 7828 1
postgres postgres 2018-03-06 12:05:36 2018-03-14 14:29:56 3 0 1 7020 1
postgres postgres 2018-03-06 12:05:36 2018-03-14 14:29:56 3 0 1 4772 1
```

5.3.5. pcp_pool_status

Pgpool-II実行時パラメータの現在の設定を name,value,descriptionの順に表示します。

```
[root@pgpool1 ~]# pcp_pool_status -Upostgres
Password:
name : listen_addresses
value: *
desc : host name(s) or IP address(es) to listen on
（以下略）
```

5.3.6. pcp_detach_node

Pgpool-IIからノードを切り離します。

```
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 8          | false             | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | true              | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | standby | 0          | false             | 0
(3 rows)

[root@pgpool1 ~]# pcp_detach_node -n 2 -Upostgres
Password:
pcp_detach_node -- Command Successful
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 8          | true              | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | down   | 0.300000  | standby | 0          | false             | 0
(3 rows)
```

5.3.7. pcp_attach_node

Pgpool-IIにノードを復帰させます。

```
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 8          | true              | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | down   | 0.300000  | standby | 0          | false             | 0
(3 rows)

[root@pgpool1 ~]# pcp_attach_node  -n 2 -Upostgres
Password:
pcp_attach_node -- Command Successful
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 8          | true              | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | standby | 0          | false             | 0
(3 rows)
```

5.3.8. pcp_promote_node

Pgpool-IIのノードをマスタに昇格させます。ただし、PostgreSQLの昇格は行われません。

```
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 8          | true              | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | standby | 0          | false             | 0
(3 rows)

[root@pgpool1 pgpool-II-10]# psql -U postgres -h 192.168.1.228 -c "select
application_name,client_addr,state,sync_state from pg_stat_replication;"
 application_name |  client_addr   |   state   | sync_state
------------------+----------------+-----------+------------
 walreceiver      | 192.168.1.229 | streaming | async
 walreceiver      | 192.168.1.230 | streaming | async
(2 rows)

[root@pgpool1 ~]# pcp_promote_node  -n 2 -Upostgres
Password:
pcp_promote_node -- Command Successful
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | down   | 0.300000  | standby | 8          | false             | 0
 1       | 192.168.1.229 | 5432 | down   | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | primary | 0          | true              | 0
(3 rows)

[root@pgpool1 pgpool-II-10]# psql -U postgres -h 192.168.1.228 -c "select
application_name,client_addr,state,sync_state from pg_stat_replication;"
 application_name |  client_addr   |   state   | sync_state
------------------+----------------+-----------+------------
 walreceiver      | 192.168.1.229 | streaming | async
 walreceiver      | 192.168.1.230 | streaming | async
(2 rows)
```

上記のように、「pcp_promote_node -n 2 -Upostgres」の実行によって Pgpool-II としては node 2 がマスタになりますが、PostgreSQLのレプリケーションの状態は変わりません。

5.3.9. pcp_stop_pgpool

Pgpool-IIをシャットダウンします。

```
[root@pgpool1 ~]# pcp_stop_pgpool -m s -Upostgres
Password:
pcp_stop_pgpool -- Command Successful
[root@pgpool1 ~]# pcp_node_count -Upostgres
Password:
ERROR: connection to socket "/tmp/.s.PGSQL.9898" failed with error "No such file or directory"
```

5.4. 障害での動作検証

今回の構成における障害発生点は大きく分けて

1. PostgreSQL サーバ
2. Pgpool-II サーバ
3. 各サーバ間のネットワーク

の3点になりますが、今回はWindows環境における障害検知・切り替えの動作検証を実施するため、Windows環境に関わる項番1のみ動作検証します。なお、項番2,3につきましては、2013年WG3の成果物として公開していますので、そちらをご覧ください。

PostgreSQL サーバの障害が発生するケースとして、

- PostgreSQLプロセスがダウンする
- Windowsサーバ自体がダウンする
- DBクラスタを格納しているストレージが故障する
- サービス側のネットワークが切断する

などを挙げることができます。いずれのパターンもPgpool-IIからは定期的にバックエンドに接続を試みて、接続できなくなったら障害が発生したと判定しています。そのため、上記ケースのいずれも Pgpool-II の障害判定は同じなので、本検証ではWindowsの PostgreSQLサービスをシャットダウンして状況をシミュレーションします。

本検証の前提として、各ノードの初期状態は下記コマンドラインの例に示すように Pgpool-II のノードから見て3台の PostgreSQL が正常に動作し、192.168.1.228 のノード(node 0)がマスタとして、それ以外のノードはスレーブとして動作してかつ、各ノード間でストリーミングレプリケーションが行われている状態を初期状態とします。

```
[root@pgpool1 pgpool-II-10]#  psql -p9999 -Upostgres
Password for user postgres:
psql (10.1)
Type "help" for help.

postgres=# show pool_nodes;
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 11         | false             | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | standby | 0          | true              | 0
(3 rows)

postgres=# select * from pg_stat_replication;
 pid  | usesysid | usename  | application_name |  client_addr   | client_hostname | client_port |
backend_start         | backend_xmin |   state   |  sent_lsn  | write_lsn  | flush_lsn  | replay_lsn
| write_lag | flush_lag | replay_lag | sync_priority | sync_state
------+----------+----------+------------------+----------------+-----------------+-------------+--
-------------------------------+--------------+-----------+------------+------------+------------+---
---------+-----------+-----------+------------+---------------+------------
 1456 |       10 | postgres | walreceiver      | 192.168.1.230 |                 |       50328 |
2018-02-19 17:06:58.904175+09 |              | streaming | 0/D6000140 | 0/D6000140 | 0/D6000140 |
0/D6000140 |           |           |    |             0 | async
 3240 |       10 | postgres | walreceiver      | 192.168.1.229 |                 |       50482 |
2018-02-19 16:51:28.101627+09 |              | streaming | 0/D6000140 | 0/D6000140 | 0/D6000140 |
0/D6000140 |           |           |    |             0 | async
(2 rows)
```

なお、Pgpool-IIのログは環境構築で述べたとおり /var/log/pgpool/pgpool.log に出力されます。

以後の項では、

- スレーブDBがダウンした場合(5.4.1)
- 3台中1台目のマスタDBがダウンした場合(5.4.2)
- 3台中2台目のマスタDBがダウンした場合(5.4.3)

の3パターンの障害にについて、挙動と回復方法を検証します。

5.4.1. スレーブDBがダウンした場合の障害検証

スレーブDBに障害が発生した時の状況の概念図が下記になります。

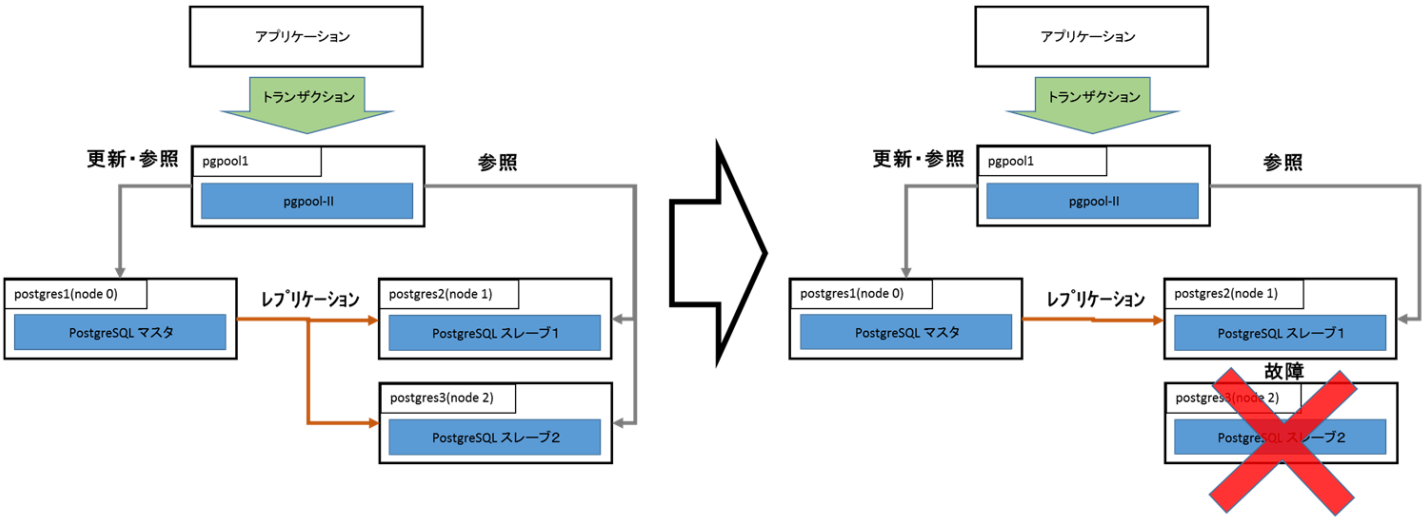


図 5.3 node 2 障害時の概念図

スレーブDBである(node 2)のPostgreSQLを停止すると、Pgpool-IIは node 2に接続できない事を検知し、ステータスをdownに変更してサービスを続行します。

- 障害検知

以下に実際の挙動を記載します。スレーブDB(node 2)がダウンした時のPgpool-IIのログは以下のようになりました。

```
2018-03-14 18:16:54: pid 3170: LOG:  failed to connect to PostgreSQL server on
"192.168.1.230:5432", getsockopt() detected error "Connection refused"
2018-03-14 18:16:54: pid 3170: ERROR:  failed to make persistent db connection
2018-03-14 18:16:54: pid 3170: DETAIL:  connection to host:"192.168.1.230:5432" failed ※1
2018-03-14 18:16:54: pid 3170: LOG:  health check failed on node 2 (timeout:0)
2018-03-14 18:16:54: pid 3170: LOG:  received degenerate backend request for node_id: 2 from pid
[3170]
2018-03-14 18:16:54: pid 3133: LOG:  Pgpool-II parent process has received failover request
2018-03-14 18:16:54: pid 3133: DETAIL:  kind: 1 flags: 1 node_count: 1 index:0
2018-03-14 18:16:54: pid 3133: DETAIL:  starting to select new master node
2018-03-14 18:16:54: pid 3133: LOG:  starting degeneration. shutdown host 192.168.1.230(5432)
2018-03-14 18:16:54: pid 3133: LOG:  Do not restart children because we are switching over node id
2 host: 192.168.1.230 port: 5432 and we are in streaming replication mode
2018-03-14 18:16:54: pid 3133: LOG:  execute command: /etc/pgpool-II-10/failover.sh 2 0 0
192.168.1.228 C:/PostgreSQL/10/data ※2
2018-03-14 18:16:54: pid 3133: LOG:  failover: set new primary node: 0
2018-03-14 18:16:54: pid 3133: LOG:  failover: set new master node: 0
failover done. shutdown host 192.168.1.230(5432)2018-03-14 18:16:54: pid 3133: LOG:  failover done.
shutdown host 192.168.1.230(5432) ※3
2018-03-14 18:16:54: pid 3167: LOG:  worker process received restart request
2018-03-14 18:16:55: pid 3166: LOG:  restart request received in pcp child process
2018-03-14 18:16:55: pid 3133: LOG:  PCP child 3166 exits with status 0 in failover()
2018-03-14 18:16:55: pid 3133: LOG:  fork a new PCP child pid 3178 in failover()
2018-03-14 18:16:55: pid 3133: LOG:  worker child process with pid: 3167 exits with status 256
2018-03-14 18:16:55: pid 3133: LOG:  fork a new worker child process with pid: 3179
```

上記ログの ※1 で node 2(192.168.1.230)の障害を検出し、192.168.1.230 ※2 で failover.sh を実行しています。しかし、スレーブDBなので、ノードを切り離すだけで処理が終了します。(※3)

この時の show pool_nodes の結果は以下のようになりました。

```
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
------------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 1          | true              | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | down   | 0.300000  | standby | 0          | false             | 0
(3 rows)
```

status 列に着目すると、node 2 はプロセスダウンにより downとなっています。その他は特に変更は無く、他の2ノードは up のままであり、node 0 がマスタである事に変わりありません。

この時のレプリケーション状態は下記の通りでした。

```
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "select * from pg_stat_replication;"
 pid  | usesysid | usename  | application_name |  client_addr   | client_hostname | client_port |
backend_start         | backend_xmin |   state   |  sent_lsn  | write_lsn  | flush_lsn  |
replay_lsn | write_lsn | flush_lsn | replay_lag | sync_priority | sync_state
------+----------+----------+------------------+----------------+-----------------+-------------+--
-----------------------------+-------------+-----------+------------+------------+------------+---
---------+-----------+-----------+------------+---------------+------------
 4892 |       10 | postgres | walreceiver      | 192.168.1.229 |                 |       54456 |
2018-03-01 14:39:22.521469+09 |             | streaming | 0/E6000140 | 0/E6000140 | 0/E6000140 |
0/E6000140 |           |           |            |             0 | async
(1 row)
```

node 1(192.168.1.229) のみレプリケーションが行われていて、node 2(192.168.1.230) はレプリケーションされていない状態となっています。

- 復旧手順

続いて、ダウンした node 2 を復旧し、Pgpool-II に再度参加させます。復旧の大まかな流れは以下のようになります。

1. 障害が発生したノードを最新の状態に戻すため、現状のマスタDBから pg_basebackup を取得する
2. 障害が発生したノードをスレーブとして起動するために、コピーしたDBクラスタ領域の直下に recovery.conf を配置する
3. 障害が発生したノードの PostgreSQL を起動し、HAの状態を確認する

1. pg_basebackup の取得

node 2 のDBサーバを現マスタ(node 0)と同期させます。これは本環境の構築時にスレーブDBを設定した時と同様に、pg_basebackupを実行することで実現します。pg_basebackupの実行の前に、古くなったDBクラスタディレクトリ「C:\PostgreSQL\10\data」を削除又はリネームします。削除が完了したら、下記のようにpg_basebackupを実行します。

```
C:\PostgreSQL\10>pg_basebackup -h 192.168.1.228 -D ./data --verbose --progress -U postgres
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/E6000060 on timeline 5
pg_basebackup: starting background WAL receiver
3094081/3094081 kB (100%), 1/1 tablespace
pg_basebackup: write-ahead log end point: 0/E6000140
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: base backup completed
```

pg_basebackupが完了したら、生成された「C:\PostgreSQL\10\data」のパーミッションを変更します。

```
C:\PostgreSQL\10>icacls "C:\PostgreSQL\10\data" /grant "NETWORK SERVICE:F" /T
処理ファイル: C:\PostgreSQL\10\data
(中略)
処理ファイル: C:\PostgreSQL\10\data\pg_xact\0000
1377 個のファイルが正常に処理されました。0 個のファイルを処理できませんでした
```

pg_basebackupが終了したら、「C:\PostgreSQL\10\data」が新たに作成されます。コピーしたDBクラスタ内に recovery.conf と recovery.done が存在している事もありますが、これらは削除します。

```
C:\PostgreSQL\10\data>hostname
postgres3

C:\PostgreSQL\10\data>cd "C:\PostgreSQL\10\data"

C:\PostgreSQL\10\data>del recovery.conf

C:\PostgreSQL\10\data>del recovery.done

C:\PostgreSQL\10\data>
```

2. recovery.conf の配置

レプリケーションのスレーブとして動作させるため「5.2.6. Windows版 PostgreSQLのレプリケーション環境構築」で記載した recovery.conf を配置します。

```
C:\PostgreSQL\10\data>hostname
postgres3

C:\PostgreSQL\10\data>type recovery.conf
standby_mode = 'on'
primary_conninfo = 'user=postgres password=postgres host=192.168.1.228 port=5432'

C:\PostgreSQL\10>
```

3. PostgreSQL の起動と状態確認

recovery.confを配置して、レプリケーションのスレーブとして起動する準備が完了したら、node 2 のDBサーバを実際に起動します。PostgreSQLの起動はこれまで通り、Windowsのサービス経由で起動します。

node 2 のDBサーバの起動が完了したら、node 0 に接続してselect * from pg_stat_replication を発行し、レプリケーションの状態を確認します。

```
[root@pgpool1 ~]# psql -U postgres -p 5432 -h 192.168.1.228 -c "select * from pg_stat_replication;"
 pid  | usesysid | usename  | application_name |  client_addr   | client_hostname | client_port |
backend_start         | backend_xmin |   state   |  sent_lsn  | write_lsn  | flush_lsn  |
replay_lsn | write_lsn | flush_lag | replay_lag | sync_priority | sync_state
------+----------+----------+------------------+----------------+-----------------+-------------+--
-----------------------------+--------------+-----------+------------+------------+------------+---
---------+-----------+-----------+------------+---------------+------------
 4892 |       10 | postgres | walreceiver      | 192.168.1.229 |                 |       54456 |
2018-03-01 14:39:22.521469+09 |              | streaming | 0/E6000140 | 0/E6000140 | 0/E6000140 |
0/E6000140 |           |           |            |             0 | async
 4980 |       10 | postgres | walreceiver      | 192.168.1.230 |                 |       59779 |
2018-03-14 19:01:40.850203+09 |              | streaming | 0/E6000140 | 0/E6000140 | 0/E6000140 |
0/E6000140 |           |           |            |             0 | async
(2 rows)
```

ノードの認識状況を下記のコマンドで確認します。

```
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
------------------
 0       | 192.168.1.228 | 5432 | up     | 0.300000  | primary | 0          | false             | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | standby | 0          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | standby | 0          | true              | 0
(3 rows)
```

上記のように、3台ともstatusがupで、roleも期待した配置になっていれば復旧は完了です。

5.4.2. 3台中1台目のマスタDBがダウンした場合の障害検証

マスタDBに障害が発生した時の状況の概念図が下記になります。

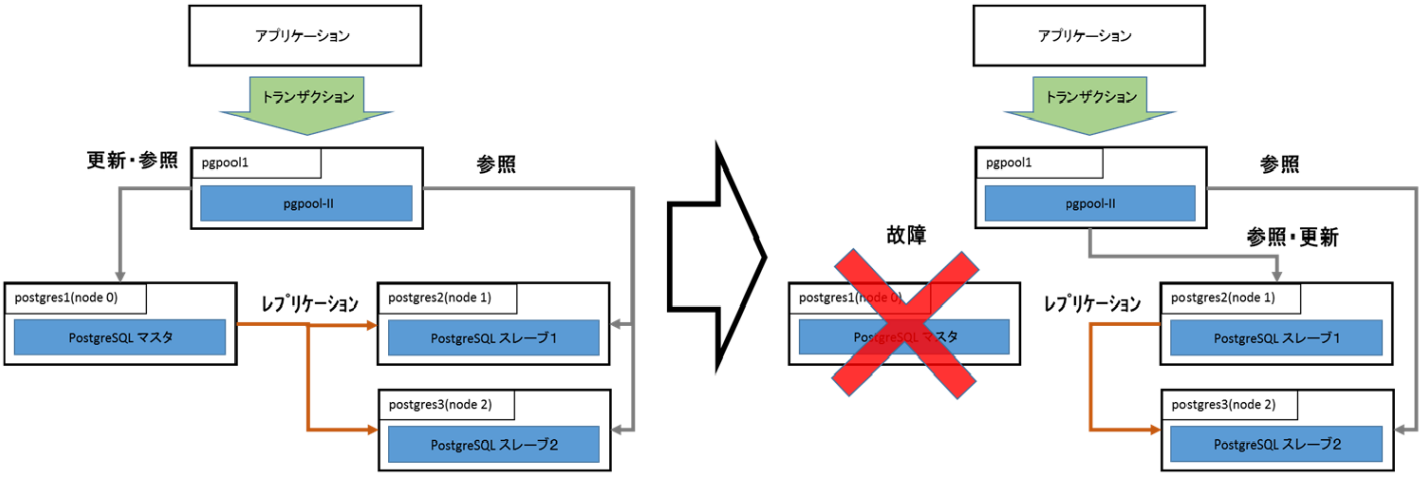


図 5.4 node 0 障害時の概念図

処理の概要

1. node 1 をマスタに昇格する

マスタDBであるnode 0 のPostgreSQLを停止すると、Pgpool-IIは node 0に接続できない事を検知し、Pgpool-IIの failover_command パラメータに記載した処理を実行します。この中で、node 1 がマスタに昇格します。

2. node 2 をnode 1 のスレーブとして認識する

Pgpool-IIの follow_master_command パラメータに記載した処理を行います。この中で node 2 を node 1 に同期させ、同期後にPgpool-IIのセカンダリノードとして接続します。

- 障害検知

以下に実際の挙動を記載します。

1. node 1 をマスタに昇格する

マスタDBがダウンした時のPgpool-IIのログは以下のようになりました。

```
2018-02-27 14:42:40: pid 23903: LOG:  failed to connect to PostgreSQL server on
"192.168.1.228:5432", getsockopt() detected error "Connection refused"
2018-02-27 14:42:40: pid 23903: ERROR:  failed to make persistent db connection ※4
2018-02-27 14:42:40: pid 23903: DETAIL:  connection to host:"192.168.1.228:5432" failed
（中略）
2018-02-27 14:42:40: pid 23868: LOG:  execute command: /etc/pgpool-II-10/failover.sh 0 0 1
192.168.1.229 C:/PostgreSQL/10/data
（中略）
2018-02-27 14:42:45: pid 23868: LOG:  find_primary_node: primary node id is 1 ※5
2018-02-27 14:42:45: pid 23868: LOG:  starting follow degeneration. shutdown host
192.168.1.228(5432)
2018-02-27 14:42:45: pid 23868: LOG:  starting follow degeneration. shutdown host
192.168.1.230(5432)
2018-02-27 14:42:45: pid 23868: LOG:  failover: 2 follow backends have been degenerated
2018-02-27 14:42:45: pid 23868: LOG:  failover: set new primary node: 1
2018-02-27 14:42:45: pid 23868: LOG:  failover: set new master node: 1
2018-02-27 14:42:45: pid 23908: LOG:  start triggering follow command.
2018-02-27 14:42:45: pid 23908: LOG:  execute command: /etc/pgpool-II-10/follow_master.sh
192.168.1.228 0 192.168.1.229 0 0 ※6
```

上記は 14:42:35 頃にnode 0のPostgreSQLを停止した時のログになります。※4 の所で Pgpool-IIがマスタDBのダウンを検出し、即座に failover.sh が実行されます。その結果、5秒後には ※5 で示す通り node 1 が新たなマスタDBに昇格している事が確認できます。また、新たなマスタDBを認識した直後に※6 で follow_master.sh が実行されている事も確認できます。

この時の show pool_nodes の結果は以下のようになりました。

```
[root@pgpool1 pgpool-II-10]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | down   | 0.300000  | standby | 0          | false             | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | primary | 0          | true              | 0
 2       | 192.168.1.230 | 5432 | down   | 0.300000  | standby | 0          | false             | 0
```

2. node 2 をnode 1 のスレーブとして認識する

status 列に着目すると、node 0 はプロセスダウンにより downとなっています。一方で node 1 は failover により マスタへプロモートしています。node 2 は follow master の処理によりpg_basebackup取得中であるため down となっています。

更に、この時のレプリケーション状態を pg_stat_replication を通じて確認します。

```
[root@pgpool1 pgpool-II-10]# psql -U postgres -p 9999 -c "select * from pg_stat_replication;"
 pid  | usesysid | usename  | application_name |  client_addr   | client_hostname | client_port |
backend_start         | backend_xmin |   state   |sent_lsn   | write_lsn  | flush_lsn  | replay_lsn
|    write_lag    |    flush_lag    |   replay_lag    | sync_priority | sync_state
------+----------+----------+------------------+----------------+-----------------+-------------+--
---------------------------+-------------+-----------+-----------+-----------+-----------+---
---------+-----------------+-----------------+-----------------+---------------+------------
 5408 |       10 | postgres | pg_basebackup    | 192.168.1.230 |                 |       53898 |
2018-02-27 14:42:52.118282+09 |             | streaming | 0/DF000108 | 0/DF000108 | 0/DF000000 |
| 00:00:02.920826 | 00:01:53.028275 | 00:01:53.028275 |             0 | async
 5484 |       10 | postgres | pg_basebackup    | 192.168.1.230 |                 |       53897 |
2018-02-27 14:42:51.369439+09 |             | backup    |           |           |            |
|                 |                 |                 |             0 | async
(2 rows)
```

上記より、node 2(192.168.1.230) で pg_basebackup が実行中であることが確認できます。この状態より更に待ち続けると、Pgpool-IIのログに以下のようなメッセージが出力されます。

```
2018-02-27 14:46:35: pid 23944: LOG:  forked new pcp worker, pid=23954 socket=7
2018-02-27 14:46:35: pid 23954: LOG:  received failback request for node_id: 2 from pid [23954]
2018-02-27 14:46:35: pid 23868: LOG:  Pgpool-II parent process has received failover request
2018-02-27 14:46:35: pid 23868: LOG:  starting fail back. reconnect host 192.168.1.230(5432) ※7
2018-02-27 14:46:35: pid 23868: LOG:  Node 1 is not down (status: 2)
pcp_attach_node -- Command Successful
2018-02-27 14:46:35: pid 23868: LOG:  Do not restart children because we are failing back node id 2
host: 192.168.1.230 port: 5432 and we are in streaming replication mode and not all backends were
down
2018-02-27 14:46:35: pid 23868: LOG:  find_primary_node_repeatedly: waiting for finding a primary
node
2018-02-27 14:46:35: pid 23868: LOG:  find_primary_node: checking backend no 0
2018-02-27 14:46:35: pid 23868: LOG:  find_primary_node: checking backend no 1
2018-02-27 14:46:35: pid 23944: DEBUG:  PCP child reaper handler
```

※7の付近でnode 2(192.168.1.230)がPgpool-IIに復帰しています。この出力の後で、show pool_nodes 及び pg_stat_replication を確認すると、以下のようになります。

```
[root@pgpool1 pgpool-II-10]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | down   | 0.300000  | standby | 0          | false             | 0
 1       | 192.168.1.229 | 5432 | up     | 0.400000  | primary | 1          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | standby | 0          | true              | 0
(3 rows)

[root@pgpool1 pgpool-II-10]# psql -U postgres -p 9999 -c "select * from pg_stat_replication;"
 pid  | usesysid | usename  | application_name |  client_addr   | client_hostname | client_port |
backend_start         | backend_xmin |   state   | sent_lsn  | write_lsn  | flush_lsn  | replay_lsn
| write_lag | flush_lag | replay_lag | sync_priority | sync_state
------+----------+----------+------------------+----------------+-----------------+-------------+--
-----------------------------+--------------+-----------+-----------+-----------+-----------+---
---------+-----------+-----------+------------+---------------+------------
 2764 |       10 | postgres | walreceiver      | 192.168.1.230 |                 |       53899 |
2018-02-27 14:46:32.605142+09 |              | streaming | 0/E0000060 | 0/E0000060 | 0/E0000060 |
0/E0000060 |           |           |            |             0 | async
(1 row)
```

この結果から、node 1 がマスタ、node 2 がスレーブとして起動している事が確認できます。また、pg_stat_replication の結果から、残る2台のDBサーバが正常にレプリケーションしている事も確認できます。

- 復旧手順

障害復旧手順は現在のサービスが停止しないように、旧マスタを新スレーブとして追加します。

手順は スレーブの復旧手順と同じになります。

5.4.3. 3台中2台目のマスタDBがダウンした場合の障害検証

本項では「マスタDBがダウンした場合の障害検証(3台中1台目の障害)」で follow master が行われた状態から、更に node 1 がダウンした時の挙動を確認します。状況を図に示すと以下のようになります。

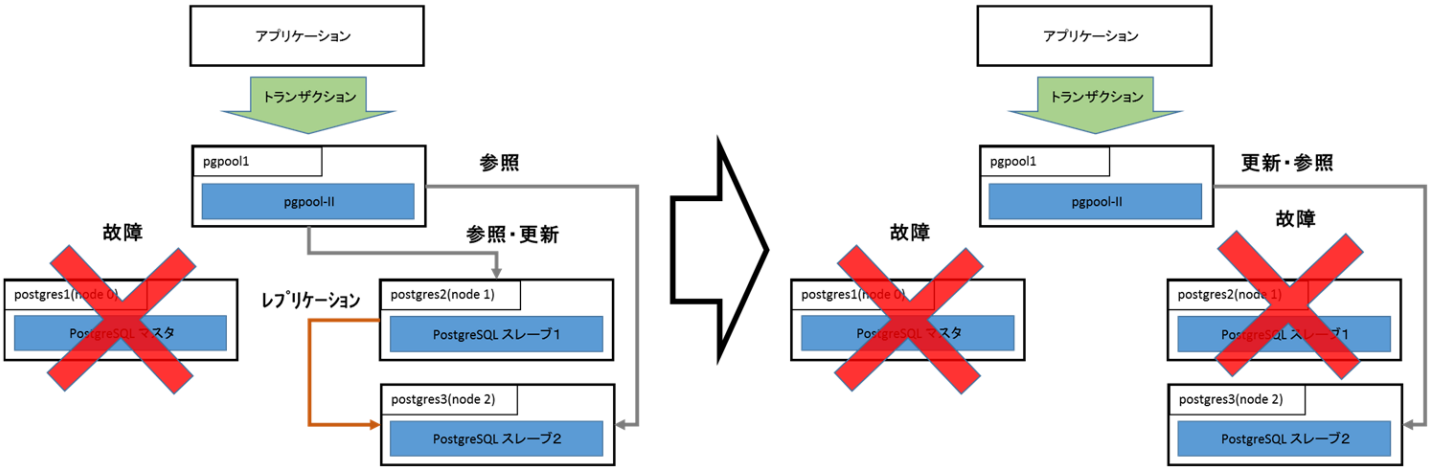


図 5.5 node 1 障害時の概念図

node 1 がダウンして新たにnode 2がマスタに昇格し、レプリケーションは行われないままサービスを続行します。

- 障害検知

この時のPgpool-IIのログは以下のようになりました。

```
2018-03-01 11:59:02: pid 25590: LOG:  failed to connect to PostgreSQL server on
"192.168.1.229:5432", getsockopt() detected error "Connection refused"
2018-03-01 11:59:02: pid 25590: ERROR:  failed to make persistent db connection
2018-03-01 11:59:02: pid 25590: DETAIL:  connection to host:"192.168.1.229:5432" failed ※8
(中略)
2018-03-01 11:59:02: pid 25554: LOG:  execute command: /etc/pgpool-II-10/failover.sh 1 1 2
192.168.1.230 C:/PostgreSQL/10/data ※9
2018-03-01 11:59:04: pid 25596: ERROR:  Failed to check replication time lag
2018-03-01 11:59:04: pid 25596: DETAIL:  No persistent db connection for the node 1
2018-03-01 11:59:04: pid 25596: HINT:  check sr_check_user and sr_check_password
2018-03-01 11:59:04: pid 25596: CONTEXT:  while checking replication time lag
2018-03-01 11:59:04: pid 25596: LOG:  failed to connect to PostgreSQL server on
"192.168.1.229:5432", getsockopt() detected error "Connection refused"
2018-03-01 11:59:04: pid 25596: ERROR:  failed to make persistent db connection
2018-03-01 11:59:04: pid 25596: DETAIL:  connection to host:"192.168.1.229:5432" failed
2018-03-01 11:59:06: pid 25554: LOG:  find_primary_node_repeatedly: waiting for finding a primary
node
2018-03-01 11:59:06: pid 25554: LOG:  find_primary_node: checking backend no 0
2018-03-01 11:59:06: pid 25554: LOG:  find_primary_node: checking backend no 1
2018-03-01 11:59:06: pid 25554: LOG:  find_primary_node: checking backend no 2
2018-03-01 11:59:06: pid 25554: LOG:  find_primary_node: primary node id is 2 ※10
2018-03-01 11:59:06: pid 25554: LOG:  starting follow degeneration. shutdown host
192.168.1.228(5432)
2018-03-01 11:59:06: pid 25554: LOG:  starting follow degeneration. shutdown host
192.168.1.229(5432)
2018-03-01 11:59:06: pid 25554: LOG:  failover: 2 follow backends have been degenerated
2018-03-01 11:59:06: pid 25554: LOG:  failover: set new primary node: 2
2018-03-01 11:59:06: pid 25554: LOG:  failover: set new master node: 2
2018-03-01 11:59:06: pid 25657: LOG:  start triggering follow command.
2018-03-01 11:59:06: pid 25657: LOG:  execute command: /etc/pgpool-II-10/follow_master.sh
192.168.1.228 0 192.168.1.230 1 1 ※11
2018-03-01 11:59:06: pid 25657: LOG:  execute command: /etc/pgpool-II-10/follow_master.sh
192.168.1.229 1 192.168.1.230 1 1
failover done. shutdown host 192.168.1.229(5432)2018-03-01 11:59:06: pid 25554: LOG:  failover
done. shutdown host 192.168.1.229(5432)
```

node 0 がダウンした時と同様に ※8 で node 1 のダウンを検出すると ※9 の所で failover.sh を実行し、4秒程度で node 2 がマスタに昇格します。(※10) この時、※11 のように follow_master.sh は実行されますが、follow master を行うノードが無いので何も処理を行わないで終了します。この時のshow pool_nodes の結果は以下のようになりました。

```
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "show pool_nodes;"
 node_id |    hostname    | port | status | lb_weight |  role   | select_cnt | load_balance_node |
replication_delay
---------+----------------+------+--------+-----------+---------+------------+-------------------+-
-----------------
 0       | 192.168.1.228 | 5432 | down   | 0.300000  | standby | 0          | false             | 0
 1       | 192.168.1.229 | 5432 | down   | 0.400000  | standby | 1          | false             | 0
 2       | 192.168.1.230 | 5432 | up     | 0.300000  | primary | 0          | true              | 0
(3 rows)
```

node 2 がマスタに昇格し、他の2ノードはdownしている事が確認できます。

また select * from pg_stat_replication より、DBサーバ間でのレプリケーションが行われていない事も確認できます。

```
[root@pgpool1 ~]# psql -U postgres -p 9999 -c "select * from pg_stat_replication;"
 pid | usesysid | usename | application_name | client_addr | client_hostname | client_port |
backend_start | backend_xmin | state | sent_lsn | write_lsn | flush_lsn | replay_lsn | write_lag |
flush_lag | replay_lag | sync_priority | sync_state
-----+----------+---------+------------------+-------------+-----------------+-------------+-------
--------+--------------+-------+----------+-----------+-----------+------------+-----------+-------
----+------------+---------------+------------
(0 rows)
```

- 復旧手順

障害復旧手順は、１台障害と同様に障害復旧で現在のサービスが停止しないように、障害が起きたサーバをスレーブとして追加します。

手順は スレーブの復旧手順と同じになります。

5.5. まとめ

本検証では、Windows 上に構築した PostgreSQL を Pgpool-II で管理するHA構成の挙動を検証しました。本検証で確認できたことは以下の通りです。

- Windows でも共有ディスクを用いずにHA構成を構築できる。
- Pgpool-II を利用する事によって運用上はプラットフォームの差異を意識する必要が無い。
- PCPコマンドはWindows環境であっても、一部のコマンド除き(※1) 利用できる。

※1 現時点では pgpool-recovery はソースビルドする必要があり、Windows に対応していないため pcp_recovery_node が利用できない。

一方で下記に示すように、環境構築に関して課題も確認できました。

- WindowsにsshをセットアップするWindowsのsshは2018年4月時点ではベータ版(This is a pre-release (non-production ready))となっており、使用するに当たっては検討が必要である。
- Pgpool-IIがWindowsに対応してないので、Linuxサーバが最低でも1台必要になる。
- Pgpool-IIがWindowsに向けて発行するsshコマンドや、呼び出すバッチの書き方等はWindows特有のノウハウが必要になる。

6. 著者

（企業・団体名順）

版	所属企業・団体名	部署名	氏名
第1.0版 (2017年度WG3)	NTTテクノクロス株式会社	IoTイノベーション事業部	勝俣 智成
	NTTテクノクロス株式会社	IoTイノベーション事業部	原田 登志
	株式会社日立製作所	OSSソリューションセンタ	稲垣 毅
	株式会社日立製作所	OSSソリューションセンタ	田畑 義之
	株式会社日立ソリューションズ	システム基盤本部 第1部	大田黒 顕仁
	富士通株式会社	ミドルウェア事業本部	山本 貢嗣