PGECons
PostgreSQL Enterprise Consortium

PostgreSQL エンタープライズ・コンソーシアム 技術部会 WG#2

バージョンアップ編

製作者
担当企業名:
株式会社富士通ソーシアルサイエンスラボラトリ

改訂履歴

版	改訂日	変更内容
1.0	2014/4/4	初版作成

はじめに

■本資料の概要と目的

本資料では、既存の PostgreSQL のバージョンアップに関しての手順とその手順を実際に試行した結果を記載しています。

■資料内の記述について

本資料では、最初に PostgreSQL のリリースバージョン番号についての理解を図ります。その後にバージョンアップの手法とその際に用いるツールの利用手順を記載します。最後に実機試験を行い、その結果を考察しています。

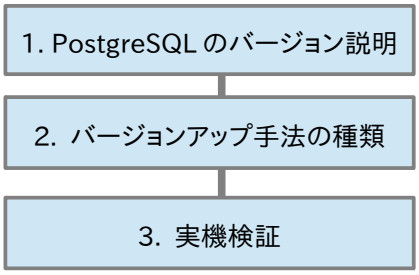

図 1: 本資料の流れ

■本資料で扱う用語の定義

本資料では、曖昧な意味としてとらえることができる用語を用いますが、次のような意味で記載しています。

表 1: 用語定義

No.	用語	意味
1	DBMS （でーびーえむえす）	データベース管理システムを指します。本書では、PostgreSQL を指します。
2	データベースクラスタ	任意のデータベース群の格納領域を指します。複数のサーバ等に跨ることもありますが、ここでは、単に 1 つの PostgreSQL が管理する、データベース群を指します。 ファイルシステム上の実体は$PGDATA（/home/postgres/data 等）となります。
3	新規/既存 PostgreSQL	バージョンアップ元と先を「既存」「新規」という接頭情報を記載して表現しています。たとえば、バージョンアップ元と先の PostgreSQL は「既存 PostgreSQL」、「新規 PostgreSQL」、バージョンアップ元の PostgreSQL のデータベースクラスタを「既存データベースクラスタ」のように記載しています。

■本資料で扱うソフトウェア

本資料では、次のソフトウェアを用いてコマンド例証等を挙げています。以下ソフトウェアバージョンと異なるソフトウェアを用いる場合や、特別な設定等を入れて実行する場合は、実行結果を記載する章で利用するソフトウェアや設定について紹介します。

表 2: 動作環境

No.	環境名	実装	バージョン
1	検証対象 DBMS	PostgreSQL	9.3.2
2	検証対象オペレーティングシステム	RedHat Enterprise Linux 6.4	intel CPU 6.4 FINAL 64bit

表 3: コマンド・ツール一覧

No.	コマンド・ツール	バージョン	ライセンス	入手元（URI）	概要
1	psql （ぴー・えす・きゅー・える）	- (PostgreSQL 標準 SQL ツール)	The PostgreSQL Lincense	（PostgreSQL 同梱）	PostgreSQL に同梱されている、ターミナル型フロントエンド。ファイルから入力を読み込むことも可能である。
2	pg_dump （ぴーじー・だんぷ）	- (PostgreSQL 標準 SQL ツール)	The PostgreSQL Lincense	（PostgreSQL 同梱）	PostgreSQL に同梱されている、オンラインバックアップツール。データベース内容を平文 SQL やバイナリ形式で出力する。
3	pg_restore （ぴーじー・りすとあ）	- (PostgreSQL 標準 SQL ツール)	The PostgreSQL Lincense	（PostgreSQL 同梱）	pg_dump の出力結果を用いて、PostgreSQL のデータベースを復元するツール。 pg_dump で平文形式以外を出力した際に用いる。
4	pg_upgrade （ぴーじー・あっぷぐれーど）	- (PostgreSQL 標準 SQL ツール)	The PostgreSQL Lincense	（PostgreSQL 同梱）	ダンプやリストアを行わずに、PostgreSQL のメジャーバージョンアップを行うためのツール。
5	Slony-I （すろーにー・わん）	2.2.0	The PostgreSQL Lincense	http://www.slony.info/	PostgreSQL データベースをレプリケーションするツール。 異なる PostgreSQL のバージョン同士でもレプリケーションを行

					うことが可能である。
6	pg_receivexlog （ぴーじー・れしーぶえっくすろぐ）	- (PostgreSQL 標準 SQL ツール)	The PostgreSQL Lincense	（PostgreSQL 同梱）	指定した PostgreSQL から WAL を受領し、特定ディレクトリに保存するツール。
7	pg_basebackup (ぴーじー・べーすばっくあっぷ)	- (PostgreSQL 標準 SQL ツール)	The PostgreSQL Lincense	（PostgreSQL 同梱）	稼働中のデータベースクラスタのベースバックアップを取得するツール

目次

1. PostgreSQL のバージョン説明

PostgreSQL のバージョン番号は以下のように、三つの数字をドットで区切って表現されます。

表 1.1: PostgreSQL のバージョン番号

PostgreSQL X.Y.Z

上記の X, Y, Z は以下の様な定義で管理されています。

表 1.2: PostgreSQL バージョン番号詳細

No.	バージョン記号	メジャーバージョン/マイナーバージョン番号	バージョンアップのタイミング
1	X	メジャーバージョン番号	システムテーブルやデータファイルの構造が変更された場合。
2	Y	メジャーバージョン番号	上記メジャーバージョンとの相違点は明確に定義されていません。
3	Z	マイナーバージョン番号	セキュリティバグやデータ破損の可能性のあるバグ等が修正された場合。その他の軽微な修正も同時に行われる。

1.1. リリースポリシー

PostgreSQL は The PostgreSQL Global Development Group によって一定期間のサポート（維持管理）が行われてます。リリース後、5 年間は過去にリリースしたメジャーバージョンと共に並行して維持管理が実施されます。この 5 年の期間はエンドオブライフ（EOL）と呼ばれ、原則遵守されますが、ベストエフォート（必ずしも守るわけではない）である注意書きがあります。この例外については後述します。

なおリリース後、5 年経過したバージョンについては、ベストエフォートにより致命的なバグに対処したソースコードとして提供される場合がありますが、公式リリースやバイナリパッケージ化の対応は実施されません。

また、上記 5 年間の保守開発がベストエフォートであることについては、深刻なバグが発見され、当該バグの修正が既存のソースコードを維持しつつ修正することが困難である場合に、5 年経過せずにコミュニティサポートの対象外となる場合があります。

2014 年 3 月時点のメジャーバージョンごとの EOL は下記の通りです。

参考：http://www.postgresql.org/support/versioning/

表 1.1.1: PostgreSQL の EOL

バージョン	最新の マイナーバージョン （2014/4/4 時点）	コミュニティサポート対象か	初回リリース日時	EOL
9.3	9.3.4	対象	2013 年 9 月	2018 年 9 月
9.2	9.2.8	対象	2012 年 9 月	2017 年 9 月
9.1	9.1.13	対象	2011 年 9 月	2016 年 9 月
9.0	9.0.17	対象	2010 年 9 月	2015 年 9 月
8.4	8.4.21	対象	2009 年 7 月	2014 年 7 月
8.3	8.3.23	対象外	2008 年 2 月	2013 年 2 月
8.2	8.2.23	対象外	2006 年 12 月	2011 年 12 月
8.1	8.1.23	対象外	2005 年 11 月	2010 年 11 月
8.0	8.0.26	対象外	2005 年 1 月	2010 年 1 月
7.4	7.4.30	対象外	2003 年 11 月	2010 年 11 月
7.3	7.3.21	対象外	2002 年 11 月	2007 年 11 月
7.2	7.2.8	対象外	2002 年 2 月	2007 年 2 月
7.1	7.1.3	対象外	2001 年 4 月	2006 年 4 月
7.0	7.0.3	対象外	2000 年 5 月	2005 年 5 月
6.5	6.5.3	対象外	1999 年 6 月	2004 年 6 月
6.4	6.4.2	対象外	1998 年 10 月	2003 年 10 月
6.3	6.3.2	対象外	1998 年 3 月	2003 年 3 月

2. バージョンアップ手法の種類

PostgreSQL のバージョンアップ手法は以下のようなものがあります。それぞれについて節を設け、特徴と用途を記載します。

表 2.1: PostgreSQL の移行手法

No.	手法	対応バージョンアップ	利用するツール・プログラム	対応節
1	データベースクラスタの継続利用	マイナーバージョン	-	2.1
2	バックアップ・リストア	メジャーバージョン	pg_dump、pg_restore	2.2
3	バージョンアップツールの利用	メジャーバージョン	pg_upgrade	2.3
4	レプリケーション	メジャーバージョン	Slony-I	2.4
5	ベースバックアップとアーカイブログ	メジャーバージョン	pg_receivexlog、pg_basebackup、pg_upgrade	2.5

2.1. データベースクラスタの継続利用

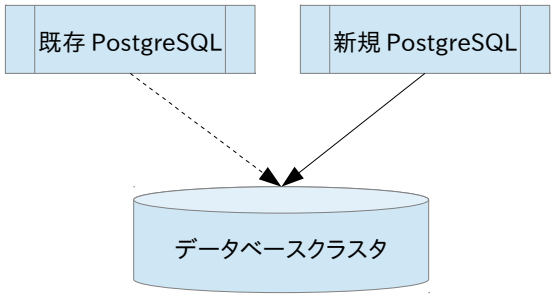

図 2.1: データベースクラスタの継続利用

データベースクラスタの継続利用とは、既存データベースクラスタを新規 PostgreSQL においてもそのまま利用する手法です。

本手法は、マイナーバージョンアップに限定されますが、データベースクラスタをそのまま残し、PostgreSQL のバイナリのみ置き換えることでバージョンアップを行えるため簡易な手順で完了します。なぜ PostgreSQL のマイナーバージョンアップに限定されるかについては前章で記述した通りであり、マイナーバージョンアップではデータベースクラスタに変更は加えられないため、データベースクラスタをそのまま利用することが可能であるからです（データベースクラスタ内に作成される PG_VERSION の中を見ると、メジャーバージョンのみ記載されていることからもわかります）。任意のマイナーバージョンアップ先の新規 PostgreSQL をインストールした後、既存データベースクラスタを参照させることでデータベースクラスタの継続利用は完了しますが、新規 PostgreSQL が既存データベースクラスタに接続後は既存データに更新が開始されるため、作業前に既存データベースクラスタのバックアップを取得する等の安全策を適宜追加する形での利用が推奨されます。

以下に既存データベースクラスタを継続利用するマイナーバージョンアップの手順を記載します。既存 PostgreSQL のバイナリを残存させたまま、環境変数の変更（バイナリパスの変更等）で対応するかについては、読者の運用方針に依りますので、検討をお願いします。下記手順では既存 PostgreSQL はアンインストールしています。

表 2.1.1: データベースクラスタ継続利用手順

No	実施内容	操作/コマンド	備考
1	既存 PostgreSQL configure オプションの確認	$ pg_config --configure	ソースからインストールする場合に必要です。
2	既存 PostgreSQL の停止	$ pg_ctl stop	-
3	既存 PostgreSQL のアンインストール	# cd [make 実行ディレクトリ] # make uninstall など、環境に合わせて実施をしてください。	RPM パッケージなどのパッケージシステムでは事前にアンインストールを行う必要があります。 make 等の場合は、インストールディレクトリを変更することで、既存 PostgreSQL のアンインストールを遅らせることが可能です。
4	新規 PostgreSQL のインストール	$ ./configure $ make world # make install-world など	-
5	新規 PostgreSQL の起動	環境変数$PGDATA に既存データベースクラスタのパスを設定するか、以下のように-D オプションで指定して起動します。 $ pg_ctl -D [既存データベースクラスタのパス] start	同一サーバにインストールするのであれば、環境変数$PGDATA は設定済の可能性があります。
6	バージョンアップ確認	$ psql =# SELECT VERSION(); PostgreSQL X.Y.Z　のように、新規 PostgreSQL のバージョンが表示されることを確認する。	-

2.2. バックアップ・リストア（pg_dump/pg_restore）

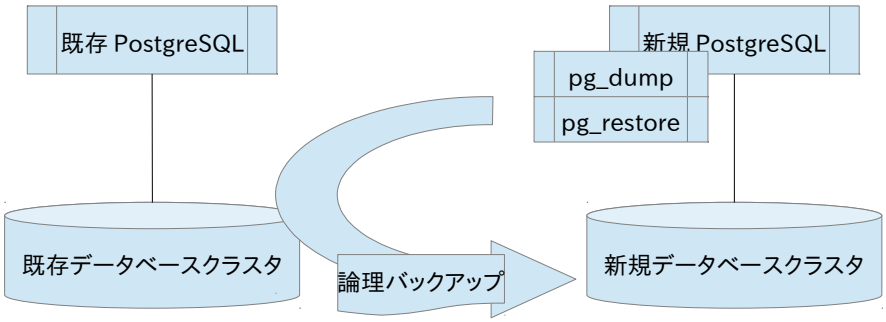

図 2.2: バックアップ・リストアでバージョンアップ

pg_dump はバックアップツールとして有名ですが、メジャーバージョンアップのツールとしても利用が可能です。pg_dump はデータベースやテーブル毎など細かい単位でバックアップの取得が可能ですが、全てのデータベースに対して一括で行うための pg_dumpall というコマンドも用意されています。pg_dumpall は、内部的には pg_dump を呼び出しているため、本書ではこれらの区別について詳しく記載せず、基本的に pg_dump について記載します。

pg_dump はメジャーバージョンアップに利用されることを想定して、新規 PostgreSQL の pg_dump が既存データベースクラスタにアクセスし、論理バックアップを取得することが可能です（参考として、PostgreSQL9.3 の pg_dump は PostgreSQL7.0 のデータベースクラスタまで読み取ることが可能です）。

pg_dump で取得した論理バックアップは、psql または、pg_restore を用いて新規 PostgreSQL にリストアします。

下表に pg_dump を用いたメジャーバージョンアップの手順を記載します。本資料の手順では、pg_dump と pg_restore は共にリモートの新規 PostgreSQL（別サーバ）のものを利用するように記載していますが、pg_dump については既存 PostgreSQL のものを利用していただいても構いません。また、新規 PostgreSQL を既存 PostgreSQL と同一サーバにインストールすることも可能です。また、メジャーバージョンアップ後の詳細な動作確認については、アプリケーション動作確認レベルでの試験をするなど、読者の環境に合わせて適宜実施してください。

pg_dump および、pg_restore の詳細なオプションについては、PostgreSQL 文書[1]を参照して下さい。

1 PostgreSQL9.3 文書 pg_dump: http://www.postgresql.jp/document/9.3/html/app-pgdump.html
PostgreSQL9.3 文書 pg_restore: http://www.postgresql.jp/document/9.3/html/app-pgrestore.html

表 2.2.1: pg_dump/pg_restore 手順

No	実施内容	操作/コマンド	備考
1	既存 PostgreSQL configure オプションの確認	$ pg_config --configure	-
2	新規 PostgreSQL のインストール	$ ./configure $ make world # make install-world など	ソースからインストールする場合は、No.1 で取得した configure オプションを参考にインストールしてください。
3	pg_dump（または pg_dumpall）の実行	（新規 PostgreSQL のサーバで実行） $ pg_dump -h [既存 PostgreSQL の IP アドレス] -Fc --verbose [データベース名] > [バックアップファイル名]	--verbose オプションによって、バックアップの取得進捗状況が標準エラーに出力されます。 -Fc オプションによって、pg_restore に適した形式で出力されます。
		（参考） $ pg_dumpall -h [既存 PostgreSQL の IP アドレス] --verbose > [バックアップファイル名]	（参考） pg_dumpall では、-Fc オプションは利用できず、平文形式のみ選択可能です。
4	pg_restore の実行	（新規 PostgreSQL のサーバで実行） $ pg_restore -C --verbose -d postgres [バックアップファイル名]	-C オプションによって、リストア前にデータベースが作成されます。 左記のコマンドで例示している postgres データベースは、CREATE DATABASE コマンドを実行するために利用するだけですので、pg_restore 実行ユーザがアクセスできるデータベースであれば何でもよいです。
5	バージョンアップ確認	$ psql =# SELECT VERSION(); PostgreSQL X.Y.Z のように、新規 PostgreSQL のバージョンが表示されることを確認する。	-
6	ANALYZE の実施	$ psql =# ANALYZE;	データ投入後、しばらくすると自動的に ANALYZE 処理が実施されますが、明示的に実行することを推奨します。
7	既存 PostgreSQL のアンインストール	# cd [make 実行ディレクトリ] # make uninstall など、環境に合わせて実施をしてください。	-

2.3. バージョンアップツールの利用（pg_upgrade）

PostgreSQL にはバージョンアップのためのツール「pg_upgrade」が同梱されています。pg_upgrade はメジャーバージョンアップ向けのプログラムです。pg_upgrade は、PostgreSQL8.3 以降[2]から現時点の PostgreSQL の最新バージョンにアップデートが可能です。PostgreSQL9.0 以降の contrib に同梱されているため、別途入手する必要はありません。

pg_upgrade を用いると、メジャーバージョンアップの際に、バックアップ・リストアを行うことなく、既存 PostgreSQL に保持されたデータを新規 PostgreSQL に移行することが可能です。メジャーバージョンアップは前述の pg_dump で同様に実施可能ですが、pg_upgrade ではデータ移行処理が高速です。ただし、pg_upgrade は、異なるサーバにインストールされた PostgreSQL のデータ移行、バージョンアップは行えません。1 台のサーバに新規 PostgreSQL と既存 PostgreSQL がインストールされている場合にのみ利用可能です。

表 2.3.1: pg_upgrade が利用できる条件

No.	条件	条件値
1	既存 PostgreSQL の最低限バージョン	8.3 以降～
2	サーバ配置	既存 PostgreSQL、新規 PostgreSQL が同一のサーバにインストールされていること

pg_upgrade を利用したバージョンアップ手法は、単純なコピーとハードリンクを用いた 2 種類の方法があります。以下にそれぞれをコピーモード、リンクモードと呼称し、移行方法のイメージを図示します。

表 2.3.2: pg_upgrade のモード

No.	モード	概要	備考
1	コピーモード	既存のデータベースクラスタのデータを、新規 PostgreSQL のデータベースクラスタにコピーするモード。	明示的なオプションを付与しない場合はコピーモードになる。
2	リンクモード	既存のデータベースクラスタと新規データベースクラスタをハードリンクで繋ぎ、データを共有するモード。	オプション(-k)を pg_upgrade 実行時に付与するとリンクモードになる。

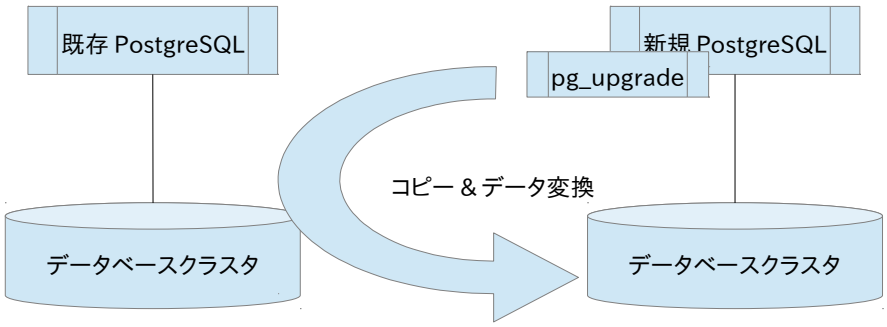

図 2.3: pg_upgrade（コピーモード）

2 PostgreSQL8.3 以前のバージョンから、最新の PostgreSQL にアップデートする場合は、データ型に互換性がないものが存在します。後述の pg_upgrade の注意点を参照して下さい

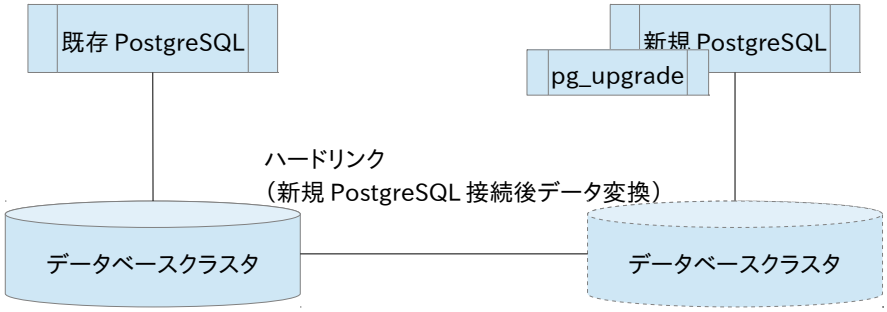

図 2.4: pg_upgrade（リンクモード）

下表に、pg_upgrade を利用したメジャーバージョンアップ（コピーモード）の手順を記載します。

表 2.3.3: pg_upgrade 手順

No.	モード	概要	備考
1	既存 PostgreSQL configure オプションの確認	$ pg_config --configure	-
2	既存 PostgreSQL の停止	$ pg_ctl stop	既存 PostgreSQL を停止します。
3	既存 PostgreSQL のデータベースクラスタの移動	$ mv /home/postgres/data/ /home/postgres/data_old	新規 PostgreSQL のデータベースクラスタ格納ディレクトリを変更する場合は、本手順は不要です。
4	新規 PostgreSQL のインストール	$ ./configure –prefix=/usr/local/pgsql_new $ make world # make install-world など	make install-world で PostgreSQL をインストールした場合、pg_upgrade バイナリおよび pg_upgrade_support ライブラリもインストールされます。
5	新規 PostgreSQL のデータベースクラスタ作成	$ initdb　[任意のオプション]	initdb コマンドを使用して、新しいデータベースクラスを作成します。この際に、既存のデータベースクラスタと互換性がある initdb のオプションを使用して下さい。また、initdb 後、pgcrypto.sql などのスキーマを定義する SQL は実行しないで下さい。これらは、pg_upgrade 実行時に自動的に実行されます。
6	環境変数設定	pg_upgrade で使用する環境変数を設定します。 PGDATAOLD（既存データベースクラスタのディレクトリ） PGDATANEW（新規データベースクラスタのディレクトリ） PGBINOLD（既存 PostgreSQL バイナリパス） PGBINNEW（新規 PostgreSQL バイナリパス）	-
7	pg_upgrade による移行可否確認	$ pg_upgrade -c	実際にデータ移行は行わずに移行が可能かをチェックします。「2.3.1. pg_upgrade による移行可否確認」を参照
8	pg_upgrade によるバージョンアップ	データ移行を行い、PostgreSQL のバージョンアップを行います。 $ pg_upgrade	デフォルトでは、コピーモードですが、-k(--link) オプションを付けると、リンクモードになります。
9	新規 PostgreSQL の起動	環境変数$PGDATA に既存データベースクラスタのパスを設定するか、以下のように-D オプションで指定して起動します。 $ pg_ctl -D [新規データベースクラスタのディレクトリ] start	
10	バージョンアップ確認	$ psql =# SELECT VERSION(); PostgreSQL X.Y.Z　のように、新規 PostgreSQL のバージョンが表示されることを確認する。	-
11	ANALYZE の実施	$ psql =# ANALYZE;	データ投入後、しばらくすると自動的に ANALYZE 処理が実施されますが、明示的に実行することを推奨します。
12	既存 PostgreSQL のアンインストール	# cd [make 実行ディレクトリ] # make uninstall など、環境に合わせて実施をしてください。	-

2.3.1. pg_upgrade による移行可否確認

pg_upgrade に-c オプションを付与し、データ移行が可能かを確認した場合、以下のようなログメッセージがコンソールに出力されます。

```
$ pg_upgrade -c
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is a superuser                       ok
Checking for prepared transactions                          ok
Checking for reg* system OID user data types                ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for presence of required libraries                 ok
Checking database user is a superuser                       ok
Checking for prepared transactions                          ok

*Clusters are compatible
```

図 2.3.4: 実行例:pg_upgrade による移行可否確認

2.3.2. pg_upgrade の利用時の注意点

pg_upgrade を用いてバージョンアップ実施中は、データベースクラスタにアクセスしてはいけません。pg_upgrade の変換途中にデータベースクラスタを参照・更新することは、データベースの破壊につながる可能性があります。
pg_upgrade は予期せぬ接続を回避するために、デフォルトで 50432 ポートで PostgreSQL を起動します。

その他の留意事項を、PostgreSQL 文書の抜粋および、pg_upgrade のソース「contrib/pg_upgrade/version_old_8_3.c[3]」を参考に以下に記載します。

表 2.3.5: pg_upgrade 利用時の留意事項

No.	注意点		概要
1	非対応データ型		次のデータ型には対応していません。 regproc、regprocedure、regoper、regoperator、regconfig、regdictionary
2	設定ファイル外部化の留意点		PostgreSQL の設定ファイル（postgresql.conf）をデータベースクラスタとは別の場所で管理している場合、pg_upgrade の-o オプションを利用して、その場所を以下のように明示する必要があります（-o オプションで指定した値は、postgres コマンドへ渡されます）。 -d /既存データベースクラスタ -o '-D /設定ファイルのパス'
3	Streaming Replication のスレーブのバージョンアップ		pg_upgrade を実行する対象 PostgreSQL は書き込みを許可する必要があるため、読み取り専用のスレーブは直接 pg_upgrade でバージョンアップすることはできません。マスタのみバージョンアップした後、マスタのデータベースクラスタを利用してスレーブを構築し直してください。
4	リンクモード使用時の既存データベースクラスタの管理		pg_upgrade には、ハードリンクを利用することでバージョンアップを高速に完了させるリンクモードが用意されています。リンクモードを使用した場合、バージョンアップ後も、既存データベースクラスタに対して、データ更新が行われます。そのため、新規データベースクラスタと既存データベースクラスタの両方の管理が必要となり、運用が煩雑になる可能性があります。
5	リンクモード使用時の既存データベースクラスタのバックアップ		pg_upgrade のリンクモードを利用しつつ、既存データベースクラスタを保管したい場合は、pg_upgrade 実行前に、既存データベースクラスタのバックアップを取得しておく必要があります。
6	バージョン 8.3 の場合	データ型 注意点①	・tsquery データ型を利用している ・name データ型が定義されていて、かつ当該列がテーブルの先頭ではない 上記のいずれかが該当する場合は、pg_upgrade が正常に動作しないため、削除した後、手作業で当該列をバージョンアップする必要があります。
7		データ型 注意点②	tsvector データ型を利用している場合、8.3 と 8.4 以降でソート方法が異なるため、バージョンアップ後、テーブルを再作成する必要があります。（pg_upgrade がメッセージを出力します。）
8		データ型 注意点③	PostgreSQL8.3 より後のバージョンでは、日付時刻データの格納書式が整数型へ変更されているため、pg_upgrade はバージョンアップ可能か検査しますが、明示的に新規 PostgreSQL をインストールする際に--disable-integer-datetimes を指定してください。
9		追加 モジュール	contrib/ltree モジュールがインストールされている場合は、pg_upgrade は動作しません。
10		インデックス	・ハッシュ、GIN、GiST インデックスを利用している ・インデックスの演算子クラス、bpchar_pattern_ops を利用している 上記のいずれかが該当する場合は、インデックスを再作成する必要があります。（pg_upgrade がメッセージを出力します。）
11		Windows 特別バージョン	8.3 で提供されていた msi インストーラによってインストールされた PostgreSQL と現在配布されているインストーラでインストールされた PostgreSQL は整数日付時刻設定が異なるため、バージョンアップができません。
12		シーケンス	8.3 より後のバージョンではシーケンスに対して新たな列が追加されているため、pg_upgrade は CREATE SEQUENCE と setval()を呼び出して対応します。

3 version_old_8.3.c File Reference : http://doxygen.postgresql.org/version__old__8__3_8c.html

2.4. レプリケーション（Slony-I）

Slony-I は、PostgreSQL に対してレプリケーションやフェイルオーバの機能を付加するソフトウェアです。レプリケーションを構成する PostgreSQL 同士のメジャーバージョンが異なっていても動作可能なことから、稼働中に異なるメジャーバージョンの新規 PostgreSQL にデータを移行することに応用することで、バージョンアップに伴うダウンタイムを大幅に短縮できる可能性があります。

最新版の PostgreSQL9.3 にバージョンアップするためには、Slony-I 2.2.0 以降を利用する必要がありますが、Slony-I 2.2.*が対応する PostgreSQL は 8.3 以降ですので注意してください。この時、Slony-I のバージョンは既存/新規 PostgreSQL の双方でそろえる必要があります。

以下に、Slony-I と PostgreSQL のバージョンの対応表を記載します。PostgreSQL のバージョン管理と同様に、Slony-I も 3 桁目はマイナーバージョンであり、ここが異なっていても互換性のない機能等はありませんので、マニュアルに明記されていない限りは、下記表では*として表記を省略しています。

表 2.4.1: Slony-I と PostgreSQL のバージョン対応

No.	Slony-I バージョン	PostgreSQL バージョン	備考
1	2.2.*	8.3.*, 8.4.*, 9.0.*, 9.1.*, 9.2.*, 9.3.*	-
2	2.1.*	8.3.*, 8.4.* 9.0.*, 9.1.*, 9.2.*	-
3	2.0.*	8.3.*, 8.4.* 9.0.*	-
4	1.2.*	7.3.3 以降, 7.4.*, 8.0.*, 8.1.*, 8.4.*	マニュアルに明記されていないため、ソースに同梱されている、PostgreSQL のバージョンごとのストアドプロシージャ定義文も参考にしました。 e.g. slony1_funcs.v84.sql また、8.2.*, 8.3.*は確認できなかったため、記載していません。

表 2.4.1 に記載されている通り、基本的に PostgreSQL の古いバージョンとしては、8.3.*以降がサポートされています。Slony-I のバージョン 1.2.*を用いることで、7.3.3 からのバージョンアップが可能となりますが、バージョンアップ先は 8.4.*までとなるため、Slony-I を用いたバージョンアップの対応範囲は、8.3.* ～ 9.3.* までと考えてください。

以下に、Slony-I を用いたメジャーバージョンアップの手順を記載します。

表 2.4.2: Slony-I を用いたメジャーバージョンアップ手順

No.	実施内容	操作/コマンド	備考
1	既存 PostgreSQL configure オプションの確認	$ pg_config --configure	ソースからインストールする場合に必要です。
2	新規 PostgreSQL のインストール	$ ./configure –prefix=/usr/local/pgsql $ make world # make install-world など	新規 PostgreSQL に contrib やその他ソースから共有オブジェクト(または DLL)をインストールして下さい。

3	Slony-I のインストール	$ ./configure --prefix=/usr/local/slony1 with-pgconfigdir=/usr/local/pgsql/bin
		$ make
		$ make install など
			既存・新規 PostgreSQL の両サーバに導入する必要があります。

4	スキーマの移行	$ pg_dump -C -s [データベース名] | psql -h [新規 PostgreSQL サーバの IP アドレス]	Slony-I は DDL をレプリケーションできないため、左記のように別途移行しておく必要があります。
5	レプリケーション不可テーブルの確認	=# select relname as table_name from pg_stat_user_tables where relname not in (select distinct table_name from information_schema.table_constraints where constraint_type='PRIMARY KEY');	Slony-I は主キーが存在しないテーブルはレプリケーションできないため、確認を行います。Slony-I でレプリケーションできないテーブルについては、別途 pg_dump で移行します。
6	初期設定スクリプトの実行	$ sh setup.sh など	既存 PostgreSQL サーバ側で実行します。http://www.slony.info/documentation/2.2/tutorial.html#FIRSTDB
7	Slony-I の起動	$ slon slony_cluster "dbname=tpcc user=postgres host=localhost"など	既存・新規 PostgreSQL 両系の Slony-I を起動します。
8	レプリケーション開始スクリプトの実行	$ sh subscribe.sh など	既存 PostgreSQL サーバ側で実行します。http://www.slony.info/documentation/2.2/tutorial.html#FIRSTDB
9	レプリケーション完了待機	-	Slony-I はデフォルトで既存 PostgreSQL のデータを全てコピーするためデータ量に順じて所要時間が増加します。
10	フェイルオーバの実行	$ sh failover.sh など	既存 PostgreSQL サーバ側で実行します。http://main.slony.info/documentation/failover.html
11	Slony-I の停止	Ctrl+C、kill（SIGTERM）等	既存・新規 PostgreSQL の両サーバの Slony-I を停止します。
12	レプリケーション対象外テーブルの移行	pg_dump/pg_restore など	主キーが付与されていないテーブルが存在する場合は、サービスを停止し、本手順を実施する必要があります。
13	バージョンアップ確認	$ psql =# SELECT VERSION(); PostgreSQL X.Y.Z　のように、新規 PostgreSQL のバージョンが表示されることを確認する。	-
14	既存 PostgreSQL の停止	$ pg_ctl stop	-

2.4.1. Slony-I の制約

Slony-I を用いてデータのレプリケーションを行う場合、下表の制約[4]があります。

4　Slony-I 2.2 文書 1.4. Current Limitations http://slony.info/documentation/2.2/limitations.html

表 2.4.1.1: Slony-I の制約

No.	制約	制約
1	ラージオブジェクトの変更不可	ラージオブジェクトの変更は、スレーブには反映されません。
2	DDL コマンドによる変更不可	DDL コマンドによるスキーマ変更は、スレーブには反映されません。そのため、レプリケーションを開始する前に、マスタのスキーマをスレーブに移行する必要があります。 また、レプリケーション後、DDL コマンドを使用する場合は、SLONIK EXECUTE SCRIPT[5]を使用する必要があります。
3	ユーザ、ロールの変更不可	ユーザ、ロールの変更はスレーブに反映されません。そのため、ユーザ、ロールの変更作業を行う場合は、スレーブに対して、同じ操作を行う必要があります。

また、Slony-I は主キーが存在しないテーブルのデータをレプリケーションできません。そのため、事前に主キーがないテーブルを確認しておく必要があります。以下は、主キーが定義されていないテーブルを取得する SQL の実行結果です。

```
$ psql tpcc

tpcc=#  select relname as table_name from pg_stat_user_tables where relname not in (select distinct
table_name from information_schema.table_constraints where constraint_type='PRIMARY KEY');
 table_name
------------
 history
(1 row)
```

図 2.5: 実行例：主キーが存在しないテーブルの確認

5 Slony-I 2.2 文書 SLONIK EXECUTE SCRIPT http://slony.info/documentation/2.2/stmtddlscript.html

2.5. ベースバックアップとアーカイブログ

既存 PostgreSQL と新規 PostgreSQL の間にもう一台の PostgreSQL（以下、中間 PostgreSQL）を用意することによって、異なるサーバ上に構築した新規 PostgreSQL へのバージョンアップを段階的に行います。

pg_upgrade と PITR を組み合わる事で、事前バックアップ取得によるバージョンアップの試行と、移行本番日の差分アーカイブログ適用でバージョンアップが可能です。そのため、作業の確実性向上と移行日の作業時間短縮につながる可能性があります。手順としては、ベースバックアップ取得(一次バージョンアップ)とアーカイブログ適応と pg_upgrade(二次バージョンアップ)の 2 段階で行います。詳細は表「2.6.1: 各移行手法のメリット・デメリット」を参照ください。

本手法では、ベースバックアップ取得後に発生するアーカイブログを取得する必要があります。

PostgreSQL が 9.2 未満の場合は、cp コマンドや rsync コマンドでアーカイブログをコピーします。

PostgreSQL が 9.2 以上であれば、pg_receivexlog[6]を使用し、アーカイブログを新規 PostgreSQL サーバに転送します。ただし、pg_receivexlog を利用するためには既存 PostgreSQL の max_wal_senders が 1 以上である必要があります。本値の変更には再起動の必要があります。再起動が許容できない場合、もしくは 9.2 未満と同様に、アーカイブログをコピーして下さい。

また、留意点として、一次バージョンアップ（ベースバックアップの取得）から、二次バージョンアップ（差分のアーカイブログ適用と pg_upgrade 実行）までの期間が長く、アーカイブログが大量に発生する場合は、アーカイブログの適用に時間がかかる可能性があります。そのため、一次バージョンアップの実施日は、二次バージョンアップ 日に極力近づけて下さい。

pg_upgrade は、「2.3. バージョンアップツールの利用（pg_upgrade）」で紹介したリンクモードを使用し、非常に高速にバージョンアップが行えるため、PostgreSQL のダウンタイムを削減することが可能です。

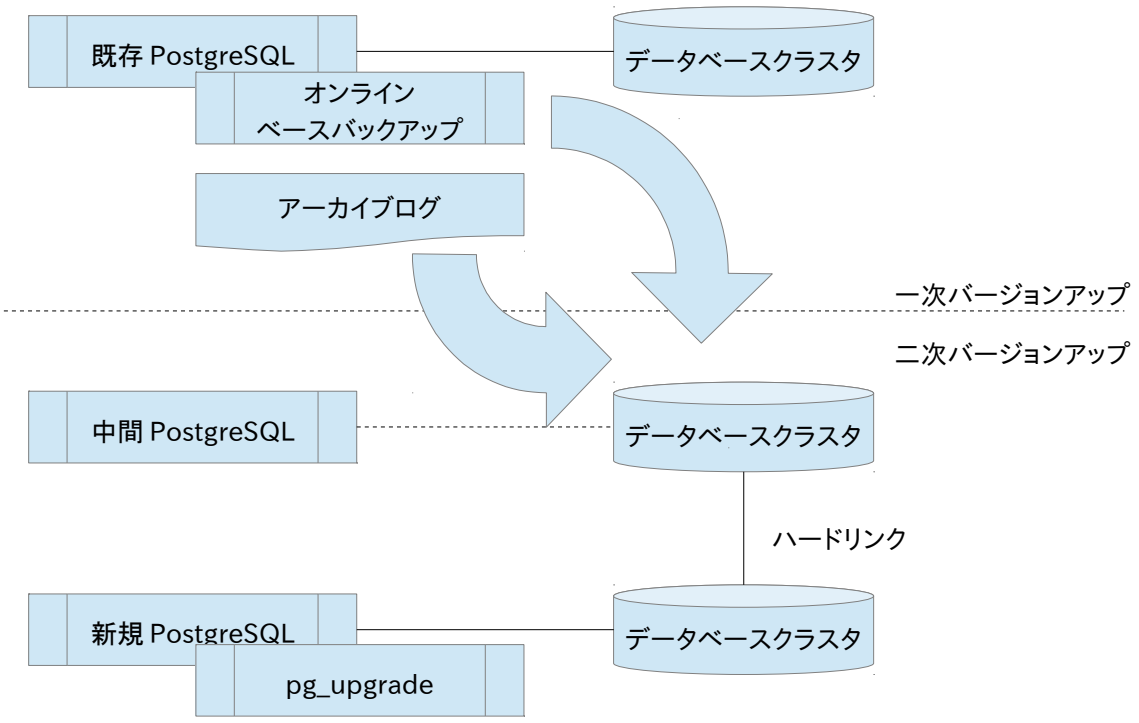

図 2.5.1: ベースバックアップとアーカイブログ

6 PostgreSQL9.3 文書 pg_receivexlog http://www.postgresql.jp/document/9.3/html/app-pgreceivexlog.html

下表に「ベースバックアップとアーカイブログ」の手順を記載します。

表 2.5.2: ベースバックアップとアーカイブログの手順

No.	モード	概要	備考
1	既存 PostgreSQL configure オプションの確認	$ pg_config --configure	ソースからインストールする場合に必要です。

2	既存 PostgreSQL の設定確認/変更	$ vi /home/postgres/data/postgresql.conf
		max_wal_senders = （設定値をインクリメント）
		
		$ vi /home/postgres/data/pg_hba.conf
		host replication all [新規 PostgreSQL サーバの IP アドレス] trust
			max_wal_senders に余裕がある場合は必要ありません。
			変更する場合は再起動が必要です。

3	新規 PostgreSQL サーバに中間/新規 PostgreSQL をインストール	新規サーバに以下の二つの PostgreSQL をインストール。 中間 PostgreSQL: /usr/local/pgsql_middle 新規 PostgreSQL: /usr/local/pgsql $ make world # make install-world など	データベースクラスタの作成は行いません。
4	中間/新規 PostgreSQL 用データベースクラスタ作成	$ mkdir /home/postgres/data_middle $ mkdir /home/postgres/data	-
5	一次バージョンアップ ベースバックアップの取得	$ pg_basebackup -h [既存 PostgreSQL サーバの IP アドレス] -p 5432 -D /home/postgres_middle/data_middle/ --xlog --progress --verbose	-
6	二次バージョンアップ① アーカイブログの移動	$ pg_receivexlog -D /home/postgres/data_middle/pg_xlog -h [既存 PostgreSQL サーバの IP アドレス] --verbose pg_receivexlog: starting log streaming at 0/A000000 (timeline 1)	WAL の受信処理が持続するため、既存 PostgreSQL を停止したのちに Ctrl+C 等で停止させます。
7	環境変数設定	pg_upgrade で使用する環境変数を設定します。 PGDATAOLD（中間データベースクラスタのディレクトリ） PGDATANEW（新規データベースクラスタのディレクトリ） PGBINOLD（中間 PostgreSQL バイナリパス） PGBINNEW（新規 PostgreSQL バイナリパス）	-
8	pg_upgrade による移行可否確認	$ pg_upgrade -c	実際にデータ移行は行わずに移行が可能かをチェックします。詳細は、pg_upgrade の移行可否チェックについては、「2.3.1 pg_upgrade による移行可否確認」を参照
9	二次バージョンアップ② pg_upgrade によるバージョンアップ	リンクモードで、PostgreSQL のバージョンアップを行います。 $ pg_upgrade --link	デフォルトでは、コピーモードですが、-k(--link) オプションを付けると、リンクモードになります。
10	既存 PostgreSQL のアンインストール	# cd [make 実行ディレクトリ] # make uninstall など、環境に合わせて実施をしてください。	-

2.6. 各手順のメリット・デメリット

５つの移行手法それぞれについて考えられるメリット・デメリットを以下に記述します。

表 2.6.1: 各移行手法のメリット・デメリット

No.	手法	メリット	デメリット・注意点
1	データベースクラスタの継続利用	・バイナリの追加インストールだけでバージョンアップできる。 ・新規 PostgreSQL のための領域を確保しなくて良い。（オペミス等を考慮した退避（コピー）は行うべきであるが、必須ではない。）	メジャーバージョンアップでは利用できない。 切り替え時にシステムの停止を伴う。（既存 PostgreSQL の停止、新規 PostgreSQL の起動）
2	バックアップ・リストア (pg_dump/pg_restore)	一般的な（枯れた）手法であるため、参考資料が豊富であり、難易度も高くない。	・差分が発生した場合に再度バックアップとリストアを実行する必要がある。 ・PostgreSQL の停止を伴う。（バックアップ＋リストア時間） ・メジャーバージョンアップの際の動作維持のための変換情報はマニュアルを追う必要があり、煩雑である。
3	バージョンアップツールの利用 （pg_upgrade）	・リンクモードの場合は、新規 PostgreSQL のための領域を確保しなく良い。 ・バージョンアップのためのツールであり、バージョンアップ時の変換等、挙動について注意点が明記されている。	・差分が発生した場合に再度バックアップとリストアを実行する必要がある。 ・PostgreSQL の停止を伴う。（バージョンアップ時間） ・サーバ間のデータ移行ができない
4	レプリケーション（Slony-I）	稼働中の PostgreSQL に対して実施できる（負荷はかかるため試験は必須）。	・レプリケーションソフトウェアであるため、バージョンアップ向けのドキュメントが整備されているとは言い難い。 ・実施手順が他の手法に比べて複雑。
5	ベースバックアップとアーカイブログ	・移行日のサービス停止後にベースバックアップの取得やアーカイブログの転送を行う必要がないため、ダウンタイムを削減することが可能。 ・中間 PostgreSQL と新規 PostgreSQL サーバで試験的にバージョンアップを行うことが行えるため、バージョンアップ中のトラブルを未然に防ぐことが可能。	アーカイブログが蓄積（物理バックアップ時点からの差分）されすぎると、その適用が長時間になる可能性がある。

3. 実機検証

本章では、前章までに挙げた手法を用いて、バージョンアップの実機検証を行いました。検証パターンは次の通りです。

表 3.1: 検証パターン

No.	手法	移行元 バージョン	移行先 バージョン
1	データベースクラスタの継続利用	9.3.0	9.3.2
2	バックアップ・リストア(pg_dump/pg_restore)	9.2.4	9.3.2
3	バージョンアップツールの利用 (pg_upgrade/コピーモード)	9.2.4	9.3.2
4	バージョンアップツールの利用 (pg_upgrade/リンクモード)	9.2.4	9.3.2
5	レプリケーション(Slony-I)	9.2.4	9.3.2
6	ベースバックアップとアーカイブログ	9.2.4	9.3.2

3.1. 検証対象環境

本検証で利用するサーバは次の通りです。PostgreSQL のパラメータ設定は、「別紙_00_PostgreSQL インストール_手順」を参照して下さい。既存、新規 PostgreSQL 向けに 2 台の物理マシンを用意し、それぞれを PostgreSQL サーバ 1、PostgreSQL サーバ 2 と呼称しています。

表 3.1.1: PostgreSQL サーバ 1 動作環境

No.	環境名	実装	バージョン情報
1	プラットフォーム	PRIMERGY TX100 S3	-
2	OS	RedHat Enterprise Linux	6.4
3	CPU	Pentium G640 2.8Ghz	-
4	RAM	4GB	-
5	HDD	SATA 3.0Gbps 250GB * 2	-
6	利用アプリケーション	JdbcRunner	1.2
7	JdbcRunner 利用 PostgreSQL JdbcDriver	JDBC 4	使用する PostgreSQL のバージョンに応じて変更

表 3.1.2: PostgreSQL サーバ 2 動作環境

No.	環境名	実装	バージョン情報
1	プラットフォーム	PRIMERGY TX100 S3	-
2	OS	RedHat Enterprise Linux	6.4
3	CPU	Pentium G640 2.8Ghz	-
4	RAM	4GB	-
5	HDD	SATA 3.0Gbps 250GB * 2	-
6	利用アプリケーション	JdbcRunner	1.2
7	JdbcRunner 利用 PostgreSQL JdbcDriver	JDBC 4	使用する PostgreSQL のバージョンに応じて変更

各手法の検証で利用したサーバは下表の通りです。

表 3.1.3: 各手法の検証で利用したサーバ

No.	手法	新規 PostgreSQL	中間 PostgreSQL	既存 PostgreSQL
1	データベースクラスタの継続利用	PostgreSQL サーバ 1	-	PostgreSQL サーバ 1
2	バックアップ･リストア (pg_dump/pg_restore)	PostgreSQL サーバ 1	-	PostgreSQL サーバ 2
3	バージョンアップツールの利用 (pg_upgrade/コピーモード)	PostgreSQL サーバ 1	-	PostgreSQL サーバ 2
4	バージョンアップツールの利用 (pg_upgrade/リンクモード)	PostgreSQL サーバ 1	-	PostgreSQL サーバ 1
5	レプリケーション(Slony-I)	PostgreSQL サーバ 1	-	PostgreSQL サーバ 2
6	ベースバックアップとアーカイブログ	PostgreSQL サーバ 1	PostgreSQL サーバ 2	PostgreSQL サーバ 2

3.1.1. JdbcRunner について

本書では、既存 PostgreSQL のデータ作成と、新規 PostgreSQL にバージョンアップ後のアプリケーションテストを行うために、JdbcRunner[7]と付属のテーブル構造の 1 つである「Tiny TPC-C[8]」を利用しています。以下のような利点があるため、JdbcRunner および同梱の「Tiny TPC-C」」を採用しました。

・JdbcRunner には、scale_factor（データ量生成因子）が実装されており、データ量を容易に変更することができる。

(データ量増加によるバージョンアップ時間の変化を測定する際に使用)

・TinyTPC-C は、TPC−C[9]に基づいており、データ構造やアプリケーションの挙動が良く知られている。

本試行では、データ構造の妥当性及び、アプリケーションの動作保証を JdbcRunner に担ってもらうことで、バージョンアップ手法の検討および、バージョンアップ後の情報収集に専念することができました。

3.1.2. 検証対象ユーザ定義テーブル

本検証でユーザデータ確認として利用するテーブル群は次の通りです。検証対象ユーザ定義テーブルとしては、JdbcRunner で利用できるベンチマークの 1 つである「Tiny TPC-C」におけるテーブルを利用しています。

表 1: warehouse テーブル

No.	カラム名	データ型
1	w_id	INTEGER
2	w_name	VARCHAR(10)
3	w_street_1	VARCHAR(20)
4	w_street_2	VARCHAR(20)
5	w_city	VARCHAR(20)
6	w_state	CHAR(2)
7	w_zip	CHAR(9)
8	w_tax	DECIMAL(4, 4)
9	w_ytd	DECIMAL(12, 2)

表 2: distinct テーブル

No.	カラム名	データ型
1	d_id	INTEGER
2	d_w_id	INTEGER
3	d_name	VARCHAR(10)
4	d_street_1	VARCHAR(20)
5	d_street_2	VARCHAR(20)
6	d_city	VARCHAR(20)
7	d_state	CHAR(2)
8	d_zip	CHAR(9)
9	d_tax	DECIMAL(4, 4)
10	d_ytd	DECIMAL(12, 2)
11	d_next_o_id	INTEGER

表 3: customer テーブル

No.	カラム名	データ型
1	c_id	INTEGER
2	c_d_id	INTEGER
3	c_w_id	INTEGER
4	c_first	VARCHAR(16)
5	c_middle	CHAR(2)
6	c_last	VARCHAR(16)
7	c_street_1	VARCHAR(20)

7 JdbcRunner : http://hp.vector.co.jp/authors/VA052413/jdbcrunner/

8 JdbcRunner「テストキット Tiny TPC-C」: http://hp.vector.co.jp/authors/VA052413/jdbcrunner/manual_ja/tpc-c.html

9 TPC-C : http://www.tpc.org/tpcc/

8	c_street_2	VARCHAR(20)
9	c_city	VARCHAR(20)
10	c_state	CHAR(2)
11	c_zip	CHAR(9)
12	c_phone	CHAR(16)
13	c_since	TIMESTAMP
14	c_credit	CHAR(2)
15	c_credit_lim	DECIMAL(12, 2)
16	c_discount	DECIMAL(4, 4)
17	c_balance	DECIMAL(12, 2)
18	c_ytd_payment	DECIMAL(12, 2)
19	c_payment_cnt	DECIMAL(4, 0)
20	c_delivery_cnt	DECIMAL(4, 0)
21	c_data	VARCHAR(500)

表 4: history テーブル

No.	カラム名	データ型
1	h_c_id	INTEGER
2	h_c_d_id	INTEGER
3	h_c_w_id	INTEGER
4	h_d_id	INTEGER
5	h_w_id	INTEGER
6	h_date	TIMESTAMP
7	h_amount	DECIMAL(6, 2)
8	h_data	VARCHAR(24)

表 5: item テーブル

No.	カラム名	データ型
1	i_id	INTEGER
2	i_im_id	INTEGER
3	i_name	VARCHAR(24)
4	i_price	DECIMAL(5, 2)
5	i_data	VARCHAR(50)

表 6: stock テーブル

No.	カラム名	データ型
1	s_i_id	INTEGER
2	s_w_id	INTEGER
3	s_quantity	DECIMAL(4, 0)
4	s_dist_01	CHAR(24)
5	s_dist_02	CHAR(24)
6	s_dist_03	CHAR(24)
7	s_dist_04	CHAR(24)
8	s_dist_05	CHAR(24)
9	s_dist_06	CHAR(24)
10	s_dist_07	CHAR(24)
11	s_dist_08	CHAR(24)
12	s_dist_09	CHAR(24)
13	s_dist_10	CHAR(24)
14	s_ytd	DECIMAL(8, 0)
15	s_order_cnt	DECIMAL(4, 0)
16	s_remote_cnt	DECIMAL(4, 0)
17	s_data	VARCHAR(50)

表 7: orders テーブル

No.	カラム名	データ型
1	o_id	INTEGER
2	o_d_id	INTEGER
3	o_w_id	INTEGER
4	o_c_id	INTEGER
5	o_entry_d	TIMESTAMP
6	o_carrier_id	INTEGER
7	o_ol_cnt	DECIMAL(2, 0)
8	o_all_local	DECIMAL(1, 0)

表 8: new_orders テーブル

No.	カラム名	データ型
1	no_o_id	INTEGER
2	no_d_id	INTEGER
3	no_w_id	INTEGER

上記の 9 つのテーブルを JdbcRunner の初期化スクリプト（tpcc_load.js）を用いて登録しています。登録する際に利用する scale_factor（データ量生成因子）を今回の試行では値を”16”として利用しました。各テーブルのレコード件数とテーブルサイズは下表の通りです。

表 9: テーブルとレコード件数

No.	テーブル名	レコード件数
1	warehouse	16
2	district	160
3	customer	480,000
4	history	480,000
5	item	100,000
6	stock	1,600,000
7	orders	480,000
8	new_orders	144,000
9	order_line	4,800,000

3.1.3. バージョンアップの成功判定

各試行では、JdbcRunner のベンチマーク走行が可能であった場合に、バージョンアップが完了したと判定しています。

```
$ export CLASSPATH=Jdbcrunner を展開したディレクトリ/jdbcrunner-1.2.jar
$ java JR jdbcrunnerを展開したディレクトリ/scripts/tpcc.js
```

図 3.1: 実行例:JdbcRunner の実行

3.2. バージョンアップの結果

3.2.1. マイナーバージョンアップの実行時間

本試行では、前述の「データベースクラスタの継続利用」で、マイナーバージョンアップを行いました。その際に、既存 PostgreSQL を停止させてから、新規 PostgreSQL が利用可能な状態になるまでの時間を計測し、「メンテナンス時間」としています。ただし、メンテナンス時間には、新規 PostgreSQL のインストールや環境変数(PATH)の変更などは含んでおらず、最低限必要なコマンドの実行時間を測定しています。

表 3.2.1: マイナーバージョンアップの実行時間

No.	手法	作業内容	実施コマンド	サービスの停止	実施時間	メンテナンス時間

1	データベースクラスタの継続利用	既存 PostgreSQL の停止	$ pg_ctl stop	必要	5 秒	9 秒
		新規 PostgreSQL の起動	$ pg_ctl start	必要	4 秒	
		-	-	-	9 秒 （合計時間）	

3.2.2. メジャーバージョンアップの実行時間

本節では、本書で紹介した５つのメジャーバージョンアップ手法を実際に試行した結果を記載しています。その際に、マイナーバージョンアップと同様に「メンテナンス時間」を計測しています。

表 3.2.2: メジャーバージョンアップの実行時間

No.	手法	作業内容	実施コマンド	サービスの停止	実行時間	メンテナンス時間
1	バックアップ・リストア (pg_dump/pg_restore)	データ抽出	$ pg_dump	必要	104 秒	289 秒
		データ投入	$ pg_restore	必要	185 秒	
		-	-	-	289 秒 (合計時間)	
2	バージョンアップツールの利用 (pg_upgrade/コピーモード)	既存 PostgreSQL の停止	$ pg_ctl stop	必要	5 秒	40 秒
		データ抽出・投入	$ pg_upgrade	必要	31 秒	
		新規 PostgreSQL の起動	$ pg_ctl start	必要	4 秒	
		-	-	-	40 秒 (合計時間)	
3	バージョンアップツールの利用 (pg_upgrade/リンクモード)	既存 PostgreSQL の停止	$ pg_ctl stop	必要	4 秒	19 秒
		ハードリンク作成	$ pg_upgrade --link	必要	10 秒	
		新規 PostgreSQL の起動	$ pg_ctl start	必要	5 秒	
		-	-	-	19 秒 (合計時間)	
4	レプリケーション (Slony-I)	slony-I によるレプリケーション	$ sh subscribe.sh	不要	125 秒	48 秒
		slony-I のフェイルオーバ	$ sh failover.sh	不要	21 秒	
		データ抽出	$ pg_dump[10]	必要	3 秒	
		データ投入	$ pg_restore	必要	45 秒	
		-	-	-	194 秒 (合計時間)	
5	ベースバックアップとアーカイブログ	ベースバックアップの取得	$pg_basebackup	不要	150 秒	28 秒
		アーカイブログの移動	$pg_receivexlog	不要	- (順次取得)	
		既存 PostgreSQL の停止	$ pg_ctl stop	必要	4 秒	
		中間 PostgreSQL の起動[11]	$ pg_ctl start	必要	5 秒	
		中間 PostgreSQL の停止	$ pg_ctl stop	必要	5 秒	
		ハードリンク作成	$ pg_upgrade --link	必要	4 秒	
		新規 PostgreSQL の起動	$ pg_ctl start	必要	10 秒	

10 slony-I では主キーが定義されていないテーブルをレプリケーションすることができません。そのため、主キーが定義されていないテーブル (history)は、pg_dump/pg_restore を用いて、データを移行しました。

11 中間 PostgreSQL 起動時に、アーカイブログを適用するため、更新量が多いシステムでは起動に時間がかかる場合があります。

		-	-	-	178 秒 (合計時間)	

3.3. バージョン後の作業

バージョンアップ後の作業として以下を行い、データ移行が完了していることを確認しました。

3.3.1. データ件数の確認

COUNT(*)にてバージョンアップ後の各テーブルの行数が、バージョンアップ前の行数と一致しているか確認しました。いずれの手法でも、行数が一致していることが確認できました。

表 3.3.1: バージョンアップ後のテーブルのレコード件数

No.	手法	バージョンアップ前と同じ件数か
1	データベースクラスタの継続利用	○
2	バックアップ・リストア(pg_dump/pg_restore)	○
3	バージョンアップツールの利用(pg_upgrade/コピーモード)	○
4	バージョンアップツールの利用(pg_upgrade/リンクモード)	○
5	レプリケーション(Slony-I)	○
6	ベースバックアップとアーカイブログ	○

3.3.2. VACUUM および ANALYZE の実行

PostgreSQL のバージョンアップ後、不要領域の回収と統計情報の最新化のため、VACUUM と ANALYZE コマンドを実行しました。VACUUM と ANALYZE 処理の所要時間は、psql のメタコマンドである¥timing を利用して測定しています

正しい時間を計測するために自動 Vacuum は無効にしておくことが望ましいですが、今回は運用環境を疑似することを優先し、無効としていません。そのため、計測中または、計測前に自動 Vacuum が実行されている可能性があります。

表 3.3.2: VACUUM に要した時間

No.	手法	VACUUM に要した時間
1	データベースクラスタの継続利用	データベースクラスタを継続利用するため必要ない
2	バックアップ・リストア(pg_dump/pg_restore)	12390ms
3	バージョンアップツールの利用(pg_upgrade/コピーモード)	890ms
4	バージョンアップツールの利用(pg_upgrade/リンクモード)	442ms
5	レプリケーション(Slony-I)	15351ms
6	ベースバックアップとアーカイブログ	435ms

表 3.3.3: ANALYZE に要した時間

No.		ANALYZE に要した時間
1	データベースクラスタの継続利用	データベースクラスタを継続利用するため必要ない
2	バックアップ・リストア(pg_dump/pg_restore)	5757ms
3	バージョンアップツールの利用(pg_upgrade/コピーモード)	11473ms
4	バージョンアップツールの利用(pg_upgrade/リンクモード)	2452ms
5	レプリケーション(Slony-I)	2624ms
6	ベースバックアップとアーカイブログ	2453ms

3.3.3. アプリケーションテスト

JdbcRunner を使用してアプリケーションテストを行い、データ移行後のデータベースが問題なく動作することを確認しました。全て問題なく完了しました。

表 3.3.4: アプリケーションテスト結果

No.	手法	アプリケーションテスト
1	データベースクラスタの継続利用	○
2	バックアップ・リストア(pg_dump/pg_restore)	○
3	バージョンアップツールの利用(pg_upgrade/コピーモード)	○
4	バージョンアップツールの利用(pg_upgrade/リンクモード)	○
5	レプリケーション(Slony-I)	○
6	ベースバックアップとアーカイブログ	○

3.4. まとめ

前述の各手法を、実機検証にて確認した結果は下表の通りです。

表 3.4.1: 実機検証結果

No.	手法	実施バージョンアップ	サービスの停止	メンテナンス時間	難易度	異なるサーバ間のバージョンアップ	推奨
1	データベースクラスタの継続利用	マイナーバージョンアップ	必要	9 秒	易	不可	○
2	バックアップ・リストア (pg_dump/pg_restore)	メジャーバージョンアップ	必要	289 秒	易	可能	○
3	バージョンアップツールの利用 (pg_upgrade/コピーモード)	メジャーバージョンアップ	必要	40 秒	易	不可	○
4	バージョンアップツールの利用 (pg_upgrade/リンクモード)	メジャーバージョンアップ	必要	19 秒	易	不可	△
5	レプリケーション (Slony-I)	メジャーバージョンアップ	必要	48 秒（※）	難	可能	△
6	ベースバックアップとアーカイブログ	メジャーバージョンアップ	必要	28 秒	並	可能	△

（※）全てのテーブルが Slony-I のレプリケーションで転送が可能な場合、メンテナンス時間はより短い時間になります。
本検証では、主キーが存在しないテーブルが存在したため、別途転送(pg_dump/pg_restore)が必要となり、メンテナンス時間が長くなっています。

マイナーバージョンアップは「データベースクラスタの継続利用」で問題なく実施することができ、若干のメンテナンス時間が発生する以外の懸念事項は特にないと考えます。
メジャーバージョンアップに関しては、枯れた手法である「バックアップ・リストア(pg_dump/pg_restore)」で確実に実行することができます。しかし、データ量が増えるとバックアップ・リストアに時間がかかり、メンテナンス時間が長くなります。(詳細は、「3.5 データ量の変化に伴うバージョンアップ時間の変化」を参照して下さい。)
短時間でバージョンアップを完了させたい場合は、「バージョンアップツールの利用(pg_upgrade/コピーモード)」もしくは、「バージョンアップツールの利用(pg_upgrade/リンクモード)」でバージョンアップを行って下さい。ただし、「バージョンアップツールの利用(pg_upgrade/リンクモード)」については、既存データベースクラスタのディレクトリも保持が必要なため、運用管理が複雑になる可能性があり、注意が必要です。
「レプリケーション（Slony-I）」については、当初想定していた通り停止時間を非常に短くできる可能性がありますが、主キーが定義されていないテーブルやラージオブジェクトの変更を、Slony-I ではレプリケーションできない等の制限があるため、現状のスキーマをご確認いただき、適性を判断してください。また、「レプリケーション」は他のバージョンアップ手法と比較すると、実行するコマンドの引数や確認すべき項目(主キーが定義されてていないテーブルを確認等)が多く、手順が煩雑になります。(詳細は、「別紙 05: レプリケーション（Slony-I）_手順」を参照して下さい。)
また、異なるサーバ間でメジャーバージョンアップを行いたい場合は、「ベースバックアップとアーカイブログ」でバージョンアップを行うと便利です。ただし、「ベースバックアップとアーカイブログ」でも、pg_upgrade のリンクモードを使用するため、運用管理が複雑になる可能性があります。

3.5. データ量の変化に伴うバージョンアップ時間の変化

データ量の変更に伴う所要時間の増加を確認するため、「バックアップ・リストア(pg_dump/pg_restore)」のみとなりますが、データ量を変更し、検証を行いました。データ量を変更するため、JdbcRunner の scale_factor を 16、32、64 に変更し、バックアップ・リストアの実行時間の変化を確認しました。その結果、データ件数増加に伴いリストアを行うための pg_restore の所要時間が特に大きく変化することがわかりました。

scale_factor=32 と scale_factor=64 における各テーブルのレコード件数は下表の通りです。

表 3.5.1: テーブルとレコード件数（scale_factor=32）

No.	テーブル名	レコード件数
1	warehouse	32
2	district	320
3	customer	960,000
4	history	960,000
5	item	100,000
6	stock	3,200,000
7	orders	960,000
8	new_orders	288,000
9	order_line	9,598,412

表 3.5.2: テーブルとレコード件数（scale_factor=64）

No.	テーブル名	レコード件数
1	warehouse	64
2	district	640
3	customer	1,920,000
4	history	1,920,000
5	item	100,000
6	stock	6,400,000
7	orders	1,920,000
8	new_orders	576,000
9	order_line	19,198,786

バックアップ・リストア（pg_dump/pg_restore）検証において、scale_factor を変更した場合の、pg_dump、pg_restore、VACUUM、ANALYZE 実行時間の推移を以下のグラフに記載します。全体的にレコード件数の増加と実行時間が比例していることが確認できます。特に、pg_restore とデータ件数の比例関係の傾きは他に比べて大きいことがわかります。バージョンアップのために、pg_dump、pg_restore を実行する際には、サービスを停止させる必要がある（コマンド自体は PostgreSQL を停止することなく実行可能ですが、実行中の差分吸収が困難である）ため、バックアップ・リストアするレコード件数が多い場合は、バージョンアップ時に十分なメンテナンス時間を確保する必要があります。必要な実行時間(メンテナンス時間)についてはデータ量と概ね比例関係にありますので、ある程度のデータ量でテストを行うことで、実際のデータ量における所要時間を予測することが可能であると考えます。

表 3.5.3: 各 scale_factor における pg_dump/pg_restore 実行時間

No.	scale_factor	計測項目			
		pg_dump	pg_restore	VACUUM	ANALYZE
1	16	104 秒	194 秒	12390ms	5757ms
2	32	207 秒	1003 秒	90920ms	15677ms
3	64	414 秒	1985 秒	161495ms	61475ms

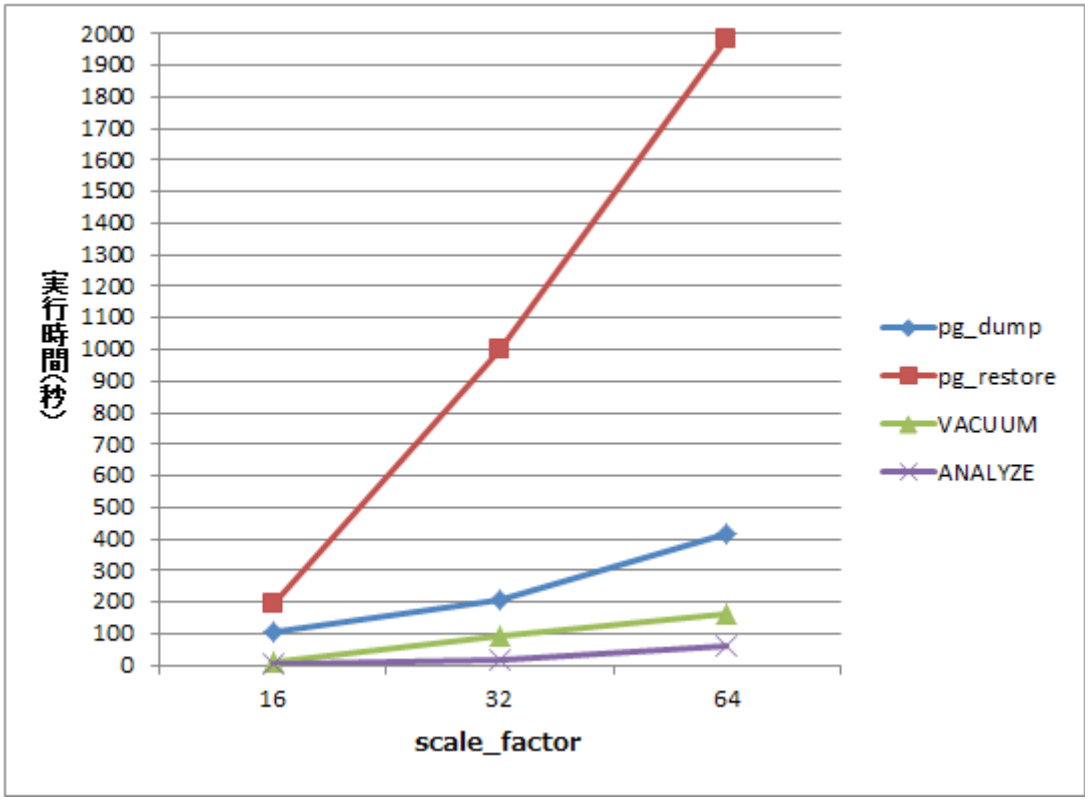

図 3.5.4: scale_factor の変更に対する実行時間の変化

4. 別紙一覧

・別紙 00: PostgreSQL インストール_手順
・別紙 01: 付帯ツールのインストール_手順
・別紙 02: データベースクラスタの継続利用_手順
・別紙 03: バックアップ・リストア（pg_dump,pg_restore）_手順
・別紙 04: バージョンアップツールの利用（pg_upgrade）_手順
・別紙 05: レプリケーション（Slony-I）_手順
・別紙 06: ベースバックアップとアーカイブログ_手順

おわりに

PostgreSQL のバージョンアップの手法の列挙と、手順の確立について記載を行いました。最終的に実際に試行してみることで、手法毎のメリット・デメリットを横断的に評価できたと考えております。本資料を参照いただくことで、実際の PostgreSQL のバージョンアップ作業をイメージできると考えておりますが、本資料はツールの利用方法の確認に主眼を置いている為、下表に示すように実際の運用についての記載やフォローが不足しています。実際に本資料を参考にバージョンアップに臨む場合は留意してください。

表 4.1: 本資料に記載していない課題

No.	課題名	概要・留意点
1	バージョン固有での追加作業	PostgreSQL のリリースノートに、バージョンアップ後に必要な作業や、非互換情報が記載されている場合があります。例えば、8.4.2 のリリースには以下のような記載があります。 「8.4.X からの移行ではダンプ/リストアは不要です。しかしハッシュインデックスが存在する場合、8.4.2 に移行した後に、破損している可能性があるためそれを修復するために REINDEX を行わなければなりません。」 本資料では、バージョン固有の作業に関しては記載していません。バージョンアップを行う際は、必ず PostgreSQL のリリースノートを確認して下さい。
2	サードパーティツールのバージョンアップ	PostgreSQL と組み合わせて、サードパーティツール(pgFoundry[12]等で公開されているツール等)を使用している場合、PostgreSQL のバージョンアップ後、予期せぬエラーが発生する可能があります。PostgreSQL のバージョンアップを行う際には、サードパーティツールの対応バージョン等を個別に確認する必要があり、必要に応じてそれらもバージョンアップを行ってください。
3	外字定義の移行	既存の PostgreSQL に外字が登録されている場合、新規の PostgreSQL にも外字登録が必要となりますが、本書では言及されていません。 PostgreSQL のバージョンによって、外字の登録方法が異なる可能性があるため、個別に調査する必要があります。外字の登録については、「2013 年度 WG2 文書 データ移行調査および実践編」をご確認下さい。
4	サーバ負荷	既存 PostgreSQL からのデータ取得をオンライン（無停止）で行える場合においても I/O 等がボトルネックとなり現行サービスに影響を与える可能性があります。本書の試行ではデータ取得時のサーバ負荷情報は計測していないため、言及できていません。実際のメジャーバージョンアップ時は大量の I/O が発生する可能性があるため、運用状態にあるサーバへの負荷について検討するべきであると考えます。

12 pgFoundry：http://pgfoundry.org/

著者

版	所属企業・団体名	部署名	氏名
バージョンアップ編 第 1 版 （2013 年度 WG2）	株式会社 富士通ソーシアル サイエンスラボラトリ	公共ビジネス本部	青木 俊彦
			杉山 貴洋
			小山田 政紀
			高橋 勝平