PGECons
PostgreSQL Enterprise Consortium

なくします！移行での”迷子”を
～ 異種DBMSからPostgreSQLへ ～

2013年度活動成果報告

PostgreSQLエンタープライズ・コンソーシアム
WG2（移行WG）

アジェンダ

- WG2について
- 2012年度の活動内容
- 2013年度の活動について
  - 活動スケジュール
  - 活動体制
- 2013年度活動の成果
  - データ移行
  - ストアドプロシージャ
  - チューニング
  - バージョンアップ
- まとめ＆今後の予定

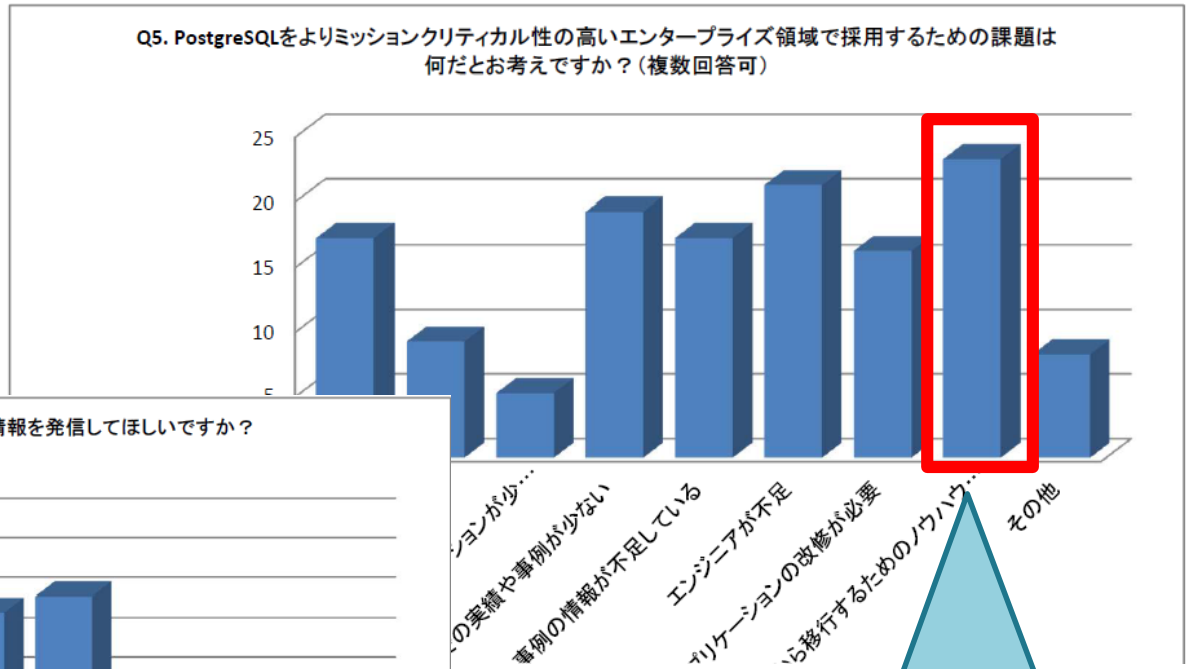
WG2について

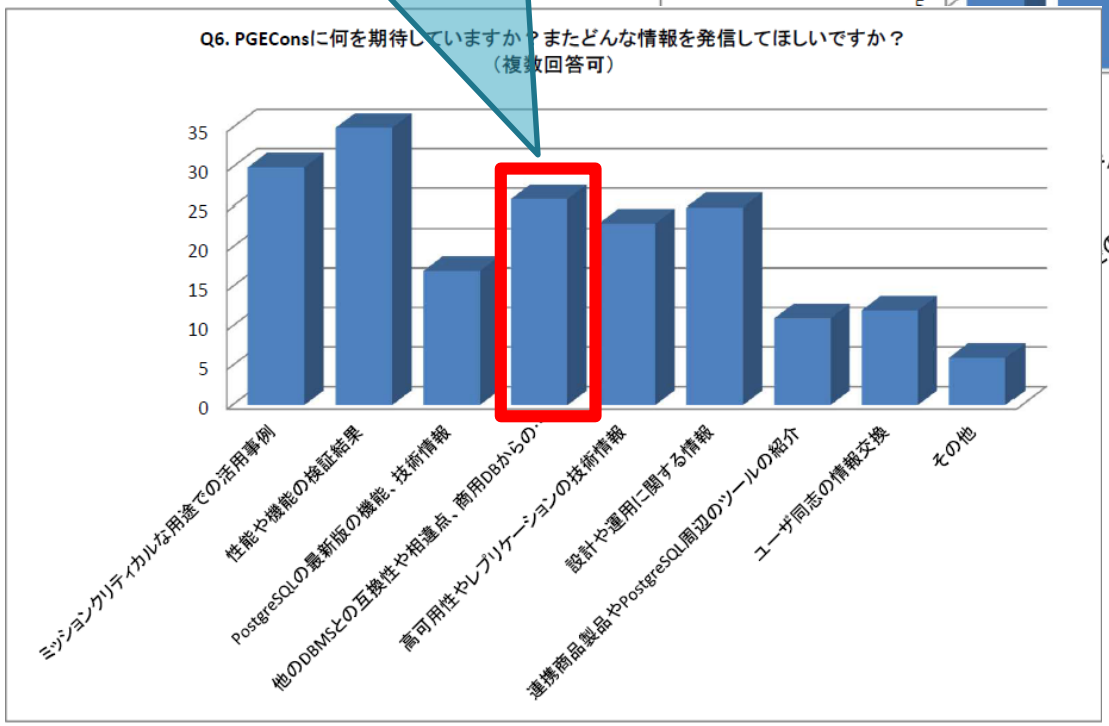
WG2(移行WG)

- 活動方針
  - PostgreSQL利用促進を目的とし、PostgreSQL経験の少ないDB技術者に向けた情報発信を行う
  - 異種DBMSと比べた際のPostgreSQLの優位点や注意事項を明確化する
  - 実システムを更改する場合の検討項目・手順を作成する
    - 例
      - SQL文やストアドプロシージャなどのアプリケーション側の注意点
      - データ定義変更の有無を判定する基準や方法
      - PostgreSQL活用時にかかるコスト要因の分析

中間成果報告(2013/12/3)アンケートより

PGEConsへの
期待
「移行に関する情報発信」





「移行ノウハウ」は、
PostgreSQL採用の
課題

2012年度の成果

2012年度 WG2(設計運用WG)活動内容

活動テーマ:異種DBMSからPostgreSQLへの移行

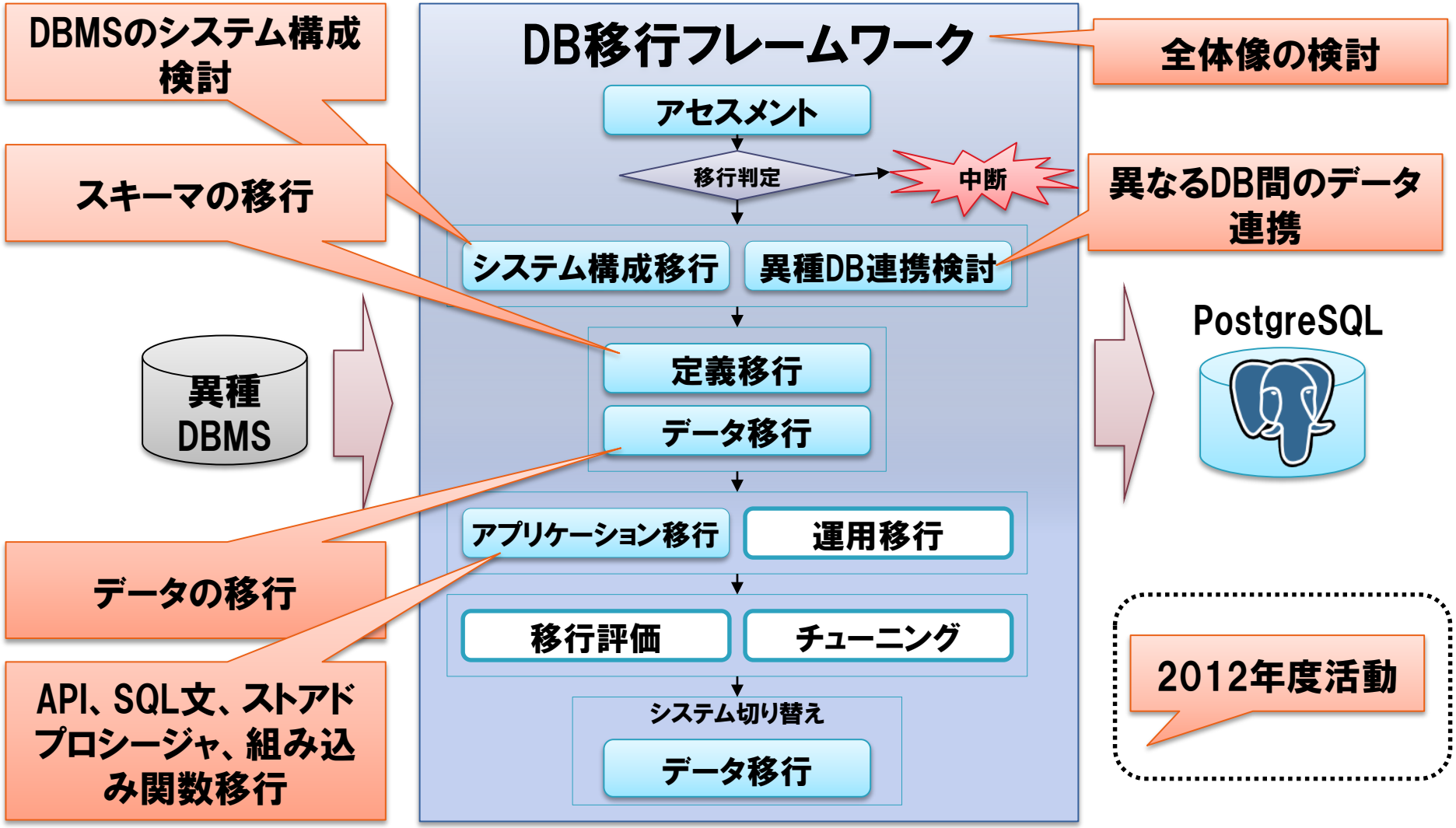
課題認識

- 異種DBMSシステムをPostgreSQLへ移行するプロセスが 確立していないことが、普及を妨げる大きな障壁と認識
  - 移行作業をどのように進めればよいかがわからない。
  - 初期段階で移行に必要なトータルコストを算出できない。
  - 過去の経験則や点在するノウハウに依存しているのが現状

活動目標

- 異種DBMSからPostgreSQLへの移行を検討する際のガイドラインを提示する。
  - 留意すべき事項
  - 難易度判断
  - 調査方法

2012年度活動対象



2012年度の成果

作成文書	内容
DB移行フレームワーク編	移行作業全体の解説
システム構成調査編	DBMSの主なシステム構成とPostgreSQLに移行時の構成検討
異種DB間連携調査編	異種DBMSとPostgreSQLのデータ連携
スキーマ移行調査編	異種DBMSとPostgreSQLのスキーマの違いと書き換えの方針検討
SQL移行調査編	Oracle,SQL Server,PostgreSQLのSQL互換性調査と書き換えの検討
ストアドプロシージャ移行調査編	異種DBMSのストアドプロシージャをPostgreSQLに移行する方法を検討
組み込み関数移行調査編	Oracle, PostgreSQLの組み込み関数互換性を調査
データ移行調査および実践編	異種DBMSからPostgreSQLへデータ移行に関する作業や注意点の調査
アプリケーション移行調査および実践編	APIやトランザクションの差異調査
運用設計	未着手
移行評価	未着手
チューニング	未着手

2013年度の活動について

2013年度WG2(運用設計→移行WG)活動テーマ

未着手項目の解消／テーマの掘り下げ

テーマ	内容
データ移行差分調査	実案件を想定したマルチバイト文字コード移行に関する調査
ストアドプロシージャ	昨年度作成資料への追加記述 Oracle PL/SQL以外のプロシージャ言語の調査
チューニング	DB移行におけるチューニングのフレームワーク検討および作業の具体化
バージョンアップ(←運用)	バージョンアップに関する手順書作成 バックアップ／監視 → WG3

2013年度WG2 活動テーマ別担当企業

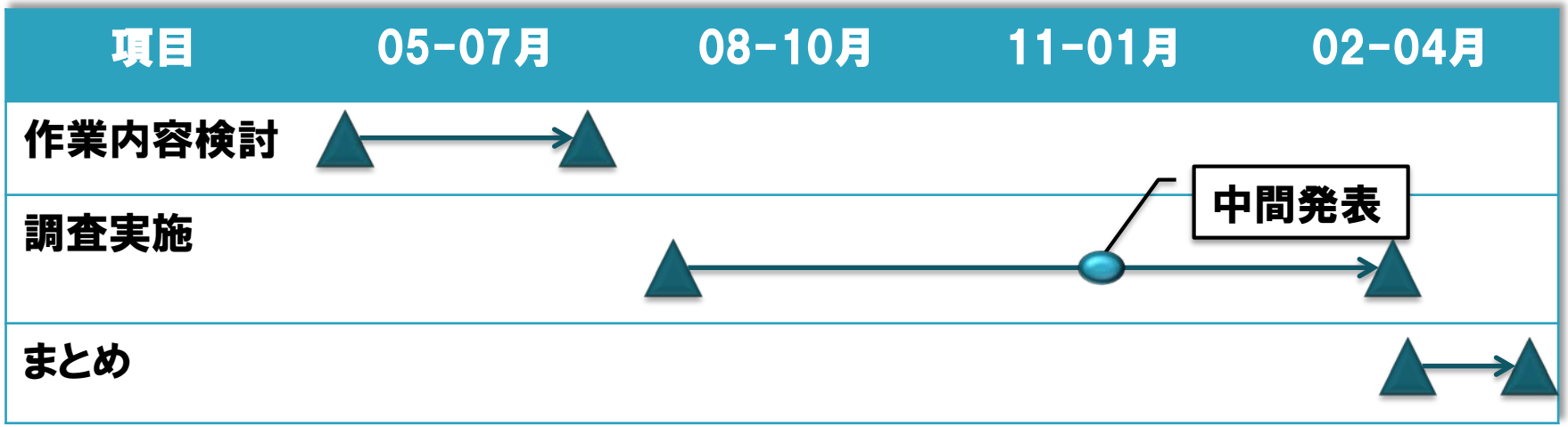
テーマ	参加企業
データ移行差分調査	• （株）アシスト • NECソリューションイノベータ（株）
ストアドプロシージャ	• クオリカ（株） • （株）インフォメーションクリエーティブ • 日本電信電話（株）
チューニング	• NECソリューションイノベータ（株） • 富士通（株） • （株）富士通ソーシアルサイエンスラボラトリ
運用（バージョンアップ）	• （株）富士通ソーシアルサイエンスラボラトリ • 日本電信電話（株） • サイオステクノロジー（株）

2013年度の活動スケジュール

05-07月　今年度の活動内容の検討

08-03月　活動項目の実施

03-04月　まとめ作業（結果をドキュメントとして整備）



2013年度活動の成果

データ移行

成果物の構成

- 移行作業の全体像を解説
  - DB移行フレームワーク編
- 移行作業に含まれる作業内容、手順の調査
  - システム構成調査編
  - 異種DB間連携調査編
  - スキーマ移行調査編
  - データ移行調査編 <UP!> 
  - ストアドプロシージャ移行調査編 <UP!>
  - アプリケーション移行調査編
  - SQL移行調査編
  - 組み込み関数移行調査編
  - チューニング編 <NEW!>
  - バージョンアップ編 <NEW!>
- 移行作業を試行する検証
  - データ移行調査および実践編
  - アプリケーション移行実践編

2012年度の成果

- データ移行調査および実践編

DBMSの諸元の比較	データ型の互換性	移行対象オブジェクトの確認方法
データ抽出方法	サポートするエンコーディング	テキストファイルの文字コード変換
外字	PostgreSQLのデータ投入方法	移行データ確認方法
データ移行後に実施すべき処理	移行手順の確立	

データ移行に関する一連の作業について、机上検証および実機検証を実施

課題

2012年度の残項目

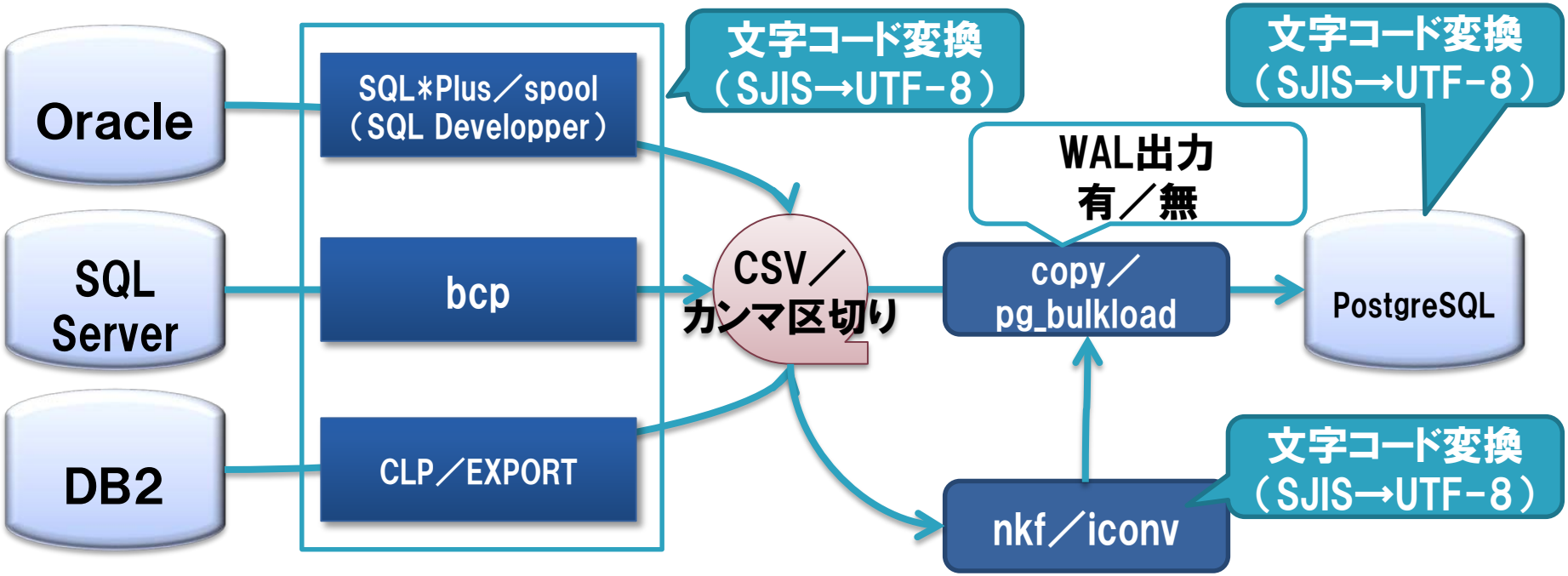
No	課題	2013年度活動内容
1	• マルチバイトデータの移行について実機検証を未実施 • 動作確認・実機検証はOracleが中心	• 各種DBMSからのデータ移行方法 • 文字コード変換 • DBに格納／出力可能な文字コード • 変換のタイミング • 使用するツール
2	LOBなど、移行手順が未確定な属性	2013年度は活動テーマとして取り上げず。

活動方針

1. 机上検討
   - 選択可能な文字コード(格納／出力)
   - 移行ツールおよび移行手順の確認
2. 実機検証

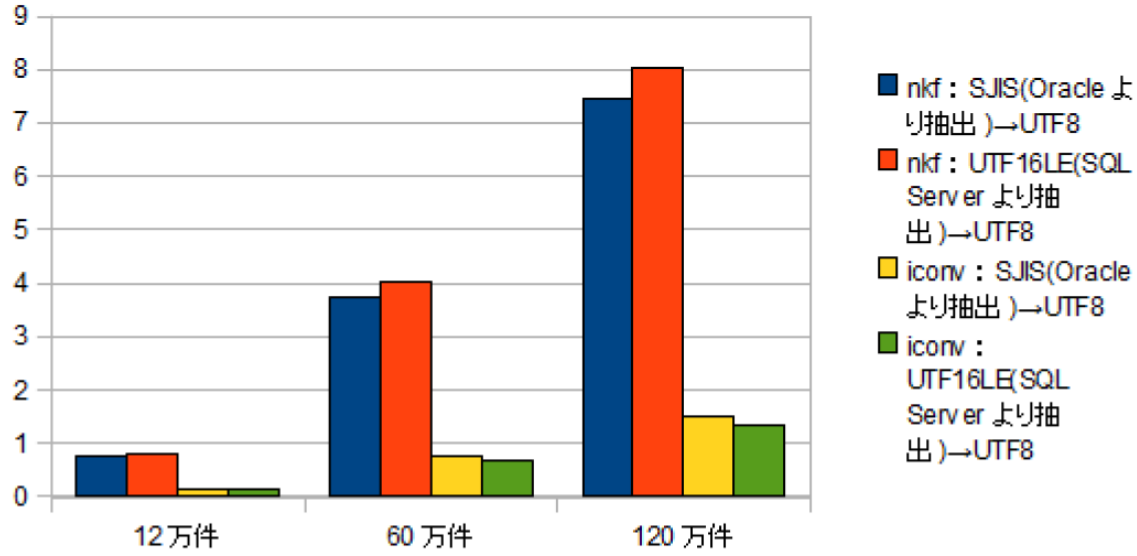
実機検証

- 郵便番号のデータを利用（約１２万件）
- Oracle、SQL Server、DB2からデータを抽出しPostgreSQLに投入
- 各DBMSからは、CSV・カンマ区切りの形式で抽出
- コード変換のタイミング別検証
（データ出力時／出力ファイル／データ投入時）



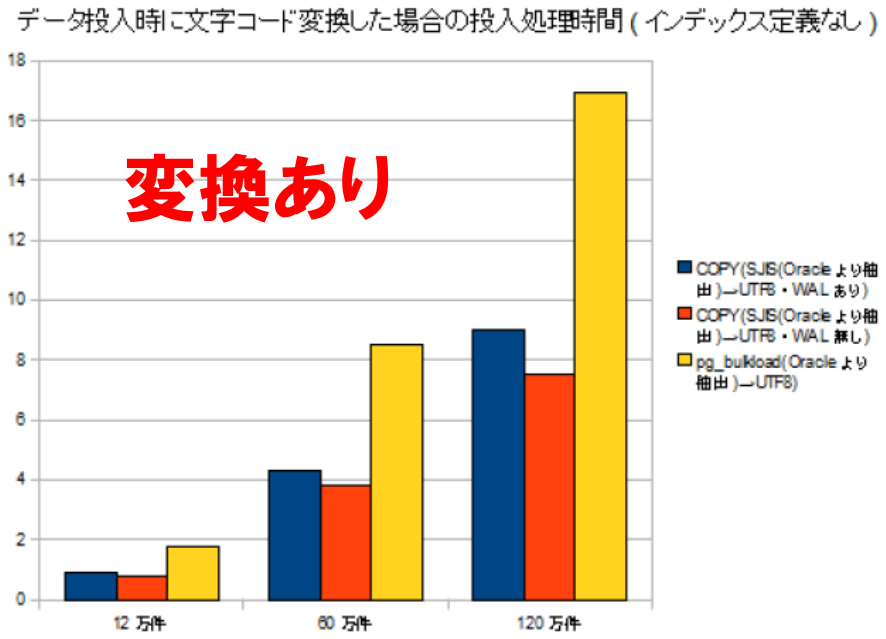
検証結果

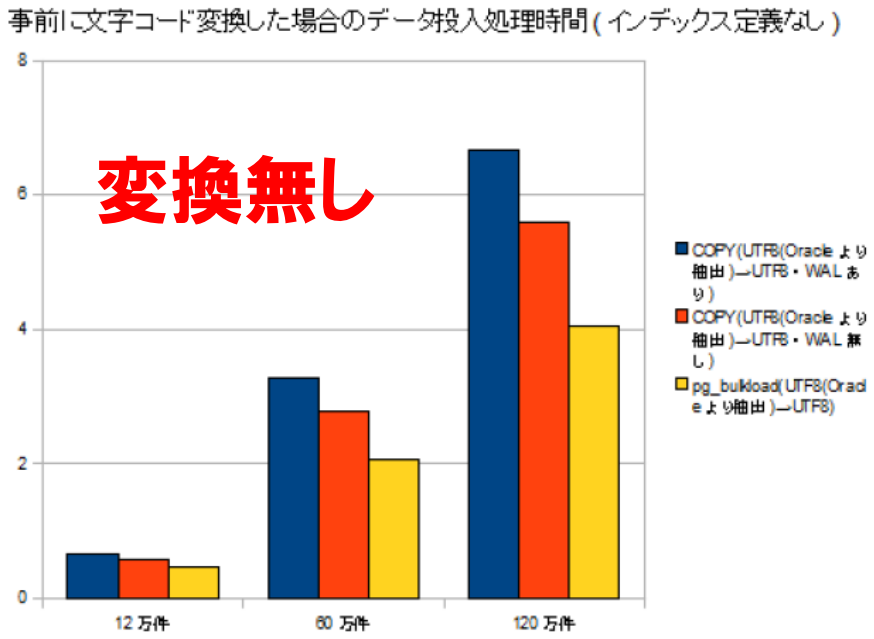
検証項目	検証結果
データの抽出／投入	• DBMS、ツール、文字コード変換のタイミングに関わらず、データ移行に成功 • CSV／カンマ区切りだが、出力時にデータを""でデータを囲うなど、ツールにより出力形式が異なる結果となった → 今回は、形式の違いがデータ投入の成否に影響しなかった • データ抽出時の文字コードはDBMSによる差を確認 • Oracle(SQL*Plus)、DB2(CLP／EXPORT)はUTF-8出力 • bcpはUTF-16出力
ファイルの文字コード変換	nkfとiconvには大きな変換性能差



検証結果

- 文字コード変換によるロード性能への影響
  - COPY(WAL出力あり)
  - COPY(WAL出力無し)
  - pg_bulkload





検証の成果と課題

検証の成果

工程	確認
Export	• データ抽出時のカンマ区切り／CSV出力形式の違い • データ抽出ツールによる出力文字コードの制限
Transform	ツール(nkf/iconv)による文字コード変換性能差
Load	• ツール(copy/pg_bulkload)による性能差 • ツールによる文字コード変換性能差

残課題

- ラージオブジェクトなどDBMSごとに異なるデータの移行手順の確立
- データ移行前後のデータ差異確認方法
- 異種DB連携ツールを利用した場合、“～”問題が発生するケースも考えられる

ストアドプロシージャ

成果物の構成

- 移行作業の全体像を解説
  - DB移行フレームワーク編
- 移行作業に含まれる作業内容、手順の調査
  - システム構成調査編
  - 異種DB間連携調査編
  - スキーマ移行調査編
  - データ移行調査編 <UP!>
  - ストアドプロシージャ移行調査編 <UP!> 
  - アプリケーション移行調査編
  - SQL移行調査編
  - 組み込み関数移行調査編
  - チューニング編 <NEW!>
  - バージョンアップ編 <NEW!>
- 移行作業を試行する検証
  - データ移行調査および実践編
  - アプリケーション移行実践編

2012年度の成果

- ストアドプロシージャ移行調査編
  - Oracle(PL/SQL)→PostgreSQL(PL/pgSQL)移行検証
    - 移行可能な構文の変換方法
    - 移行できない機能(プロシージャ内のCOMMITなど)

Oracle PL/SQL → PostgreSQL PL/pgSQL検証を実施

課題

■ 2012年度の積み残し課題

No	課題	活動内容
1	Oracle(PL/SQL)以外のプロシージャ言語の調査	PL/pgSQLとの差異調査 • SQL Server(Transact-SQL) • DB2
2	移行が困難な構文、機能の移行方法検討	本年度は検討のみ

調査方針

- 観点
  - プロシージャ／関数の定義構文
  - プロシージャの構造の差異
  - 変数定義／操作方法
  - 制御構造
  - カーソル
  - エラーハンドリング

調査内容と結果

- 構文ごとの比較／移行方法の調査

CREATE FUNCTION 文の比較

SQL Server	PostgreSQL
CREATE FUNCTION *ファンクション名* (*@引数名 データ型*) RETURNS *戻り値* AS BEGIN DECLARE *@変数名 データ型* *処理内容* END	CREATE FUNCTION *ファンクション名* (*引数名* IN *データ型*) RETURNS *戻り値* AS $$ DECLARE *変数名 データ型;* BEGIN *処理内容;* END; $$ LANGUAGE plpgsql;

CREATE FUNCTION 文の比較

DB2	PostgreSQL
CREATE OR REPLACE FUNCTION *ファンクション名* (IN *引数名 データ型*) RETURNS *戻り値* LANGUAGE SQL BEGIN DECLARE *変数名 データ型;* *処理内容;* END	CREATE FUNCTION *ファンクション名* (*引数名* IN *データ型*) RETURNS *戻り値* AS $$ DECLARE *変数名 データ型;* BEGIN *処理内容;* END; $$ LANGUAGE plpgsql;

調査内容と結果

DBMS	検討項目	検討結果
SQL Server	プロシージャ／関数の定義構文	変数宣言の場所、記述言語の指定など随所に違い
	変数定義／操作方法	変数の名に@を付け、SET文でデータを代入など、構文に違い
	制御構造	GOTO、WAITFORなど未サポート機能があるが、多くは代替構文あり
	カーソル	カーソルのレコード件数取得などの機能差異および随所の構文差異
	トランザクション	プロシージャ内のCOMMITは実行不可
	エラーハンドリング	多くの機能は代替構文あり
DB2	プロシージャ／関数の定義構文	関数の引数など随所に違い
	変数定義／操作方法	変数宣言の場所や、SET文による代入などの違い
	制御構造	GOTOなど未サポート機能があるが、多くは代替構文あり
	カーソル	カーソル定義の記述箇所やカーソルのステータス確認方法等に違い
	トランザクション	プロシージャ内のCOMMITは実行不可
	エラーハンドリング	エラーハンドラの定義など、PL/pgSQLとの差異が随所に存在

詳細な文法上の違いは、成果物をご覧ください

検証の成果と課題

- 検証の成果
  - 構文別の移行可否および移行方法
  - 構文は似通っているが、随所に細かい差異が存在する
  - 構文の差異は解消できるが、DBMSの機能差異の解消は困難
- 残課題
  - 変換作業に対する難易度の提示
  - 単純変換できない構文や機能についての解決策の提示
  - 公開可能なサンプルソースによる実例の提示

チューニング

成果物の構成

- 移行作業の全体像を解説
  - DB移行フレームワーク編
- 移行作業に含まれる作業内容、手順の調査
  - システム構成調査編
  - 異種DB間連携調査編
  - スキーマ移行調査編
  - データ移行調査編 <UP!>
  - ストアドプロシージャ移行調査編 <UP!>
  - アプリケーション移行調査編
  - SQL移行調査編
  - 組み込み関数移行調査編
  - チューニング編 <NEW!> ◀◀◀
  - バージョンアップ編 <NEW!>
- 移行作業を試行する検証
  - データ移行調査および実践編
  - アプリケーション移行実践編

課題

- DB移行を前提としたチューニング
- 異種DBMSにおいて一般に用いられ、PostgreSQLも提供しているチューニング手法の紹介

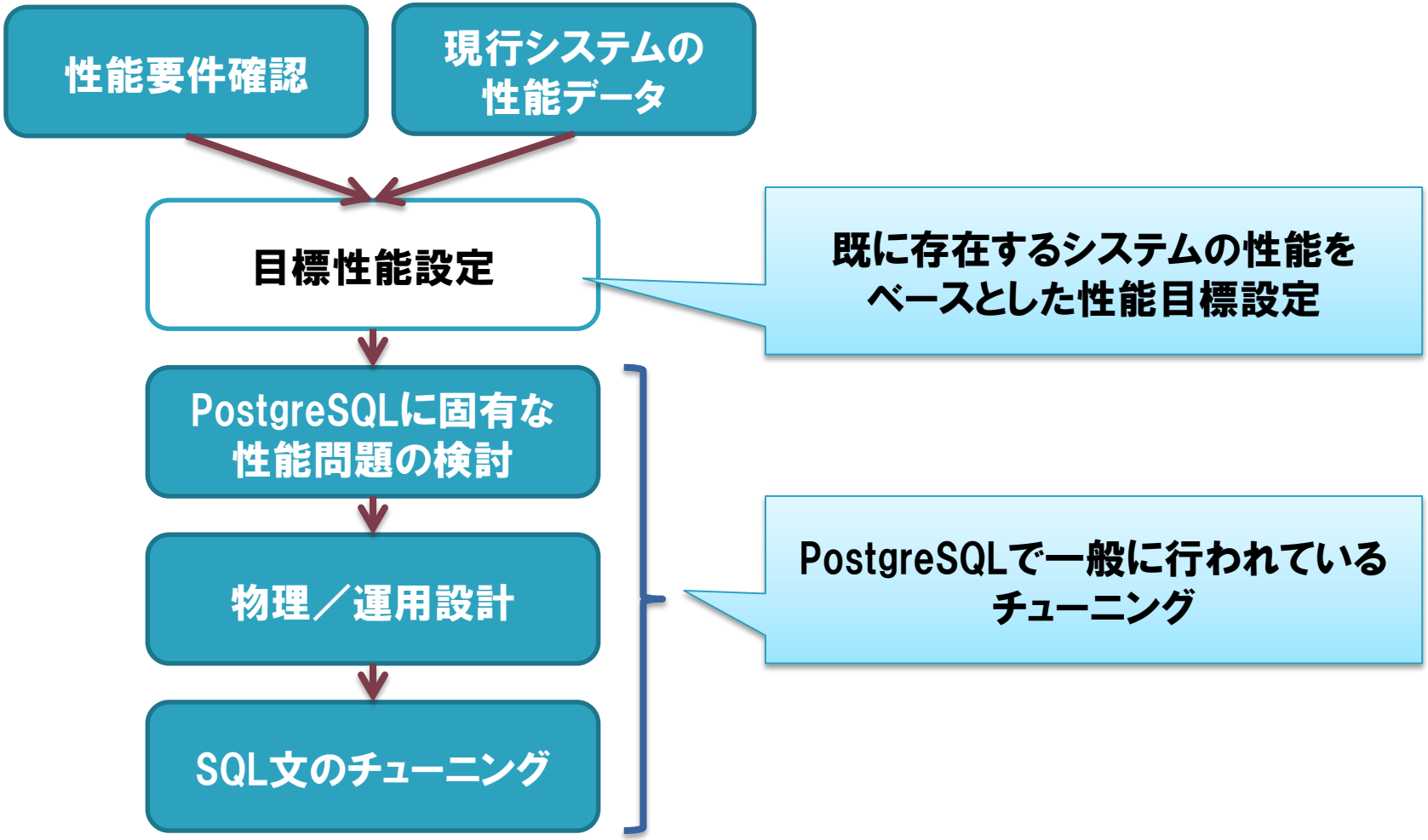
No	課題	活動内容
1	チューニング手順の検討	DBMS移行時のチューニング手順検討
2	PostgreSQL固有な課題	PostgreSQLに固有な性能上の課題の検討
3	物理設計	• パラメータチューニング • システム構成による性能改善
4	SQL文のチューニング	• SQL文の性能確認手段 • チューニング手法

調査方針

- DBMS移行におけるチューニング作業手順検討
- PostgreSQLに固有な性能上の留意点の確認
- 物理設計
  - パラメータ設定
  - 負荷分散など
- SQL文のチューニング手法

調査内容

■ チューニング手順



PostgreSQLに固有な性能上の留意点

- VACUUM運用
- 更新されたレコードの最初の参照で発生するトランザクション状態の反映
- DBMSによるチューニング機能の違い

項番	チューニング機能	PostgreSQL	Oracle	SQL Server	DB2
1	自動パラメータチューニング	△ (pgtune)	○	○	○
2	Hint 情報	△ (pg_hint_plan)	○	○	○ (最適化プロファイル)
3	統計情報のセーブ・ロード	△ (pg_dbms_stats)	○	○	○ (db2look -m)
4	バッファ固定	×	○	×	○

表 2.1: チューニング機能差異 (○…標準機能、△…追加機能により対応、×…非対応)

パラメータ設計

チューニング関連のパラメータおよび設定値の考え方の提示。

- 資源の消費量に関するチューニング
- 更新ログ(WAL)に関するチューニング
- レプリケーション設定
- 自動VACUUMの設定
- ロック関連の設定

パラメータ設計(抜粋)

表 3.3: パラメータ(資源の消費)

No.	パラメータ名	初期値	説明
1	shared_buffers (しぇあーど・ばっふぁーず)	128MB	データベース全体で利用する共有バッファを設定します。本値を大きくすることにより、パフォーマンスは向上しますが「チェックポイント処理」(共有バッファとストレージの同期処理)の処理負荷が向上する可能性があります。このため、むやみに値を大きくすることは推奨できません。 PostgreSQL 文書では 1GB のメモリを搭載したコンピュータを例に取り、25%としていますが、現在のコンピュータスペックでは本例は適応され難いスペックとなっています。OS や PostgreSQL 以外のサービスが消費するメモリを考慮して設定してください。
2	temp_buffers (てんぷ・ばっふぁーず)	8MB	セッション中で作成する一時テーブル用のメモリ量を指定します。本値で設定したメモリは直ちに確保されずに一時テーブル利用時にのみ、設定値を最大として必要な分だけ確保されます。 一時テーブルを利用する場合は、アプリケーション仕様における最大値から検討を始め、ハードウェアのメモリ総量を踏まえて決定してください。 全てのセッションで一時テーブルを作成すると、最大で max_connections * temp_buffers 分のメモリを確保しますので注意してください。

:

SQL文のチューニング

- EXPLAINによるプランの確認

```
$ psql -h serverName]  -p [portNum] testdb
psql (9.3.2)
"help" でヘルプを表示します.

testdb=# EXPLAIN UPDATE pgbench_accounts SET abalance = abalance + 1 WHERE aid = 12;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 Update on pgbench_accounts  (cost=0.00..8.38 rows=1 width=103)
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts  (cost=0.00..8.38 rows=1 width=103)
         Index Cond: (aid = 12)
(3 行)
```

- パラメータ設定によるプランの変更
- HINT、統計情報の固定(アドインツール)

PostgreSQLは
他DBMSが提供する一般的なチューニング手段を備える

成果と課題

- 成果
  - 大まかな手順およびチューニング手法の提示
- 課題
  - 移行を前提とした具体的なチューニング作業の紹介
  - ノウハウ

参加企業の増加による事例やノウハウの充実が課題

バージョンアップ

成果物の構成

- 移行作業の全体像を解説
  - DB移行フレームワーク編
- 移行作業に含まれる作業内容、手順の調査
  - システム構成調査編
  - 異種DB間連携調査編
  - スキーマ移行調査編
  - データ移行調査編 <UP!>
  - ストアドプロシージャ移行調査編 <UP!>
  - アプリケーション移行調査編
  - SQL移行調査編
  - 組み込み関数移行調査編
  - チューニング編 <NEW!>
  - バージョンアップ編 <NEW!> ◀◀◀
- 移行作業を試行する検証
  - データ移行調査および実践編
  - アプリケーション移行実践編

課題

バグフィックス版(リビジョンアップ)や新機能の利用のためのバージョンアップ手段が複数存在し、作業のリスクや比較した情報が存在しない。

No	課題	活動内容
1	バージョンアップ手法の整理	・バージョンアップ手法の机上調査および、手順書作成 ・手法別のメリット・デメリット
2	データ量とバージョンアップ時間の相関関係	データ量とバージョンアップ作業時間との関係を確認

活動の方針

- **机上調査**
  - PostgreSQLで採用可能なバージョンアップ手法を調査
- **実機検証**

※ バージョンアップ元：既存PostgreSQL、バージョンアップ先：新規PostgreSQL

No.	手法（使用ツール）	説明	対応バージョンアップ
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そのまま残し、PostgreSQLのバイナリのみ新規バージョンに置き換える。	マイナー
2	バックアップ・リストア（pg_dump/pg_restore）	既存PostgreSQLからpg_dumpで取得したバックアップを、psqlまたはpg_restoreを用いて新規PostgreSQLにリストアする。	メジャー
3	バージョンアップツールの利用（pg_upgrade）	バージョンアップ用のプログラムであるpg_upgradeを用いて、バージョンアップを行う。pg_upgradeの以下のモードを検証する。 1.　コピーモード 既存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 2.　リンクモード 既存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	メジャー
4	レプリケーション（Slony-I）	異なるメジャーバージョン間でも、PostgreSQLのレプリケーション構成を実現できるSlony-Iを用いて、バージョンアップを行う。	メジャー
5	ベースバックアップとアーカイブログ（pg_receivexlog、pg_basebackup、pg_upgrade）	以下のように各手法を組み合わせ、移行日に大規模なデータコピーが発生することを回避した手法。（ベースバックアップを利用して、バージョンアップの事前試行も可能） 事前：ベースバックアップ（pg_basebackup） 都度：アーカイブログ転送（pg_receivexlog） 移行日：PITR、バージョンアップ（pg_upgrade）	メジャー

検証の内容と結果

■ 各手法の実機検証の結果

No.	手法（使用ツール）	実施バージョンアップ	サービスの停止	サービス停止時間	手順の難易度	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	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	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	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	必要	19秒	易	不可	△
5	レプリケーション（Slony-I）	メジャー	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（pg_receivexlog、pg_basebackup、pg_upgrade）	メジャー	必要	28秒	並	可能	△

※ 主キーがないテーブルは、 Slony-Iではレプリケーションできない。
本検証では、主キーが無いテーブルには、pg_dump/pg_restoreを利用。

■ バージョンアップ時のデータを生成には、JdbcRunnerの付属テーブル構造の1つである「Tiny TPC-C」を利用

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

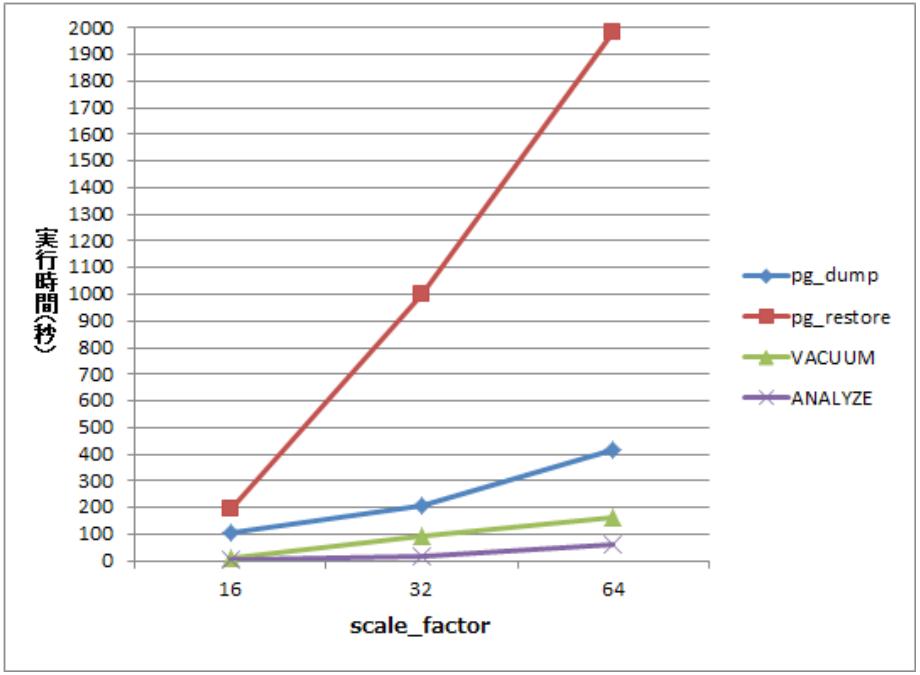
検証の成果 1/2

- 各バージョン手法のメリット、デメリット

項番	手法（使用ツール）	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マイナー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。	メジャーバージョンアップでは利用できない。 ・PostgreSQLの停止を伴う
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。	・バックアップ・リストア中にデータ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） ・データ量が増えると、実行時間も長くなる。 （次頁を参照）
3	バージョンアップツールの利用 （pg_upgrade/コピーモード）	・「バックアップ・リストア（pg_dump/pg_restore」と比較すると、高速にバージョンアップが可能。 ・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。	・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	バージョンアップツールの利用 （pg_upgrad/リンクモード）	「バージョンアップツールの利用（pg_upgrad/コピーモード）」よりも、高速にバージョンアップが可能。	・古いデータベースクラスタも保持が必要なため、運用管理が複雑になる
5	レプリケーション（Slony-I）	稼働中のPostgreSQLのデータを移行できる（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。 ・レプリケーションできないデータが存在する （ラージオブジェクトや主キーがないテーブルのデータ等）
6	ベースバックアップとアーカイブログ（pg_receivexlog、pg_basebackup、pg_upgrade）	・ダウンタイムを削減することが可能。 ・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。

検証の成果 2/2

- データ量による所要時間の増加を確認 （pg_dump/pg_restoreを利用）
- JdbcRunnerのscale_factor（データ量生成因子）のパターン（16、32、64）によりデータ量を変更
- 参考としてバージョンアップ後の、VACUUMおよびANALYZE時間も測定

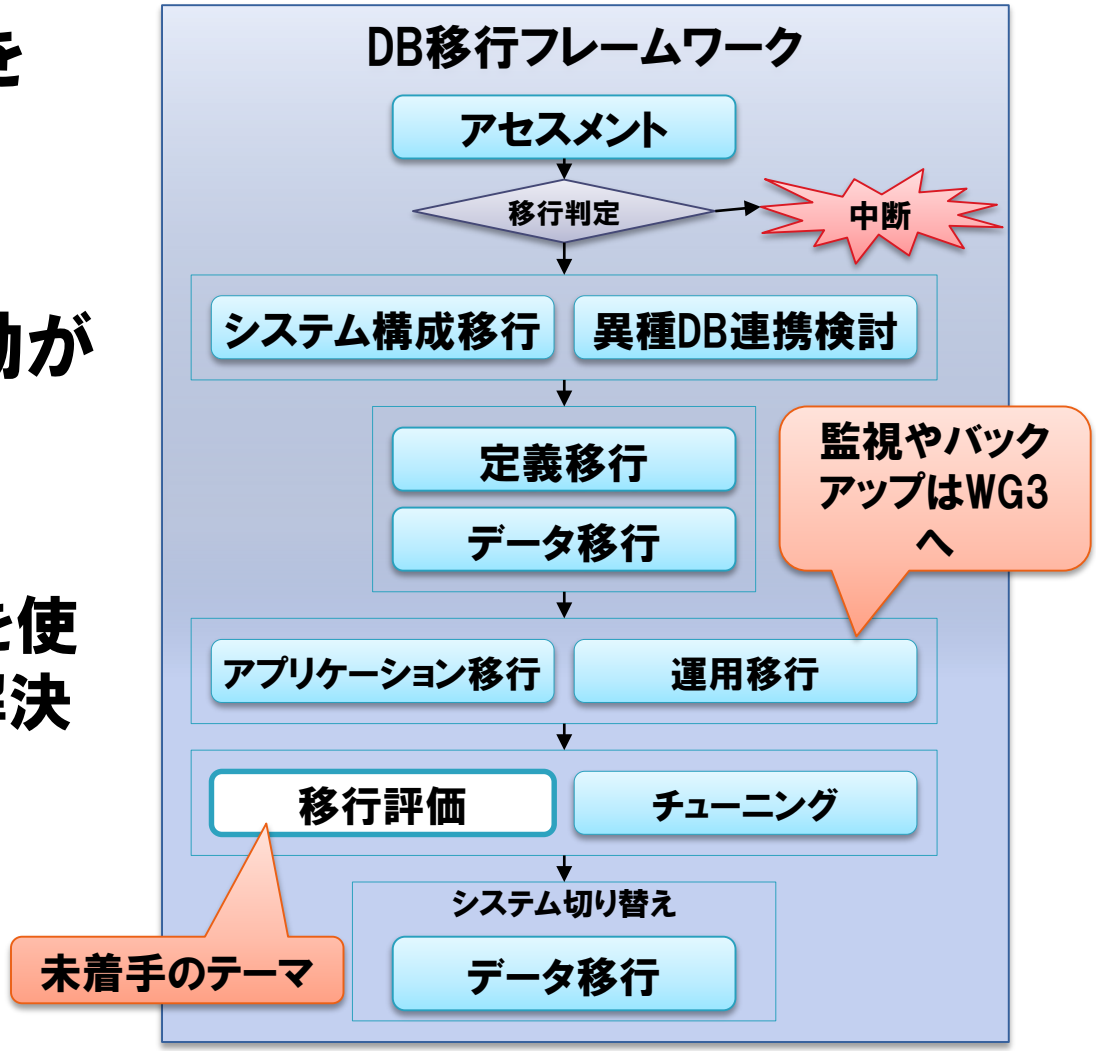


- データ量とバージョンアップ作業時間は概ね比例関係
- 事前に1/nのデータ量で検証を行い、実際の作業時間が予測可能
（データ型等によって大きく変化する可能性がある）

活動に対する課題と今後の予定

WG2の成果

- テーマとしては全体をカバーしつつある
  - 移行評価のみ残
- 今後も継続的な活動が必用
  - テーマ別の完成度
  - 具体的な移行対象を使用した検証や課題解決



活動の課題と反省

- 参加企業からの意見
  - サンプルソースの提示が必用（ストアドプロシージャ）
  - 参加企業数に見合った分量のテーマの選定
  - 仲間を増やすような活動ができたら良い
  - 参加企業間での議論が少なかった

2014年度は、具体的な移行作業や移行方法の議論主体の活動になる（かもしれません）

PGECons
PostgreSQL Enterprise Consortium

PGEConsで
一緒に活動しませんか？