商用DBからPostgreSQLへの移行

2012/2/24

株式会社インテリジェンスビジネスソリューションズ
プラットフォームソリューション部
高木 慎太郎

1. 本日の論点・移行に関する前提
2. データベース移行の実態
3. データベース移行方法の整理
4. 検証状況の共有
5. 検証・本日のまとめ

本日の論点:データベースの移行が必要となる背景

▶ データベースを取り巻く環境の変化に対応するため、データベース移行が必要不可欠となっています。

高額なライセンスコスト

- OSSDBの機能向上により、OSSDBを採用しても、商用DBと遜色なく利用可能です。
- 商用DBならではの機能が有効活用されておらず、商用DBの機能が不要な環境があります。

▼

コスト削減のために移行検討

サーバの性能向上

- CPU、メモリ、ディスクの性能向上より、1サーバに1データベースだけではリソースの余剰が発生しています。

▼

ＤＢサーバの集約のために移行検討

データベースサーバの老朽化

- サーバのサポート・リース切れやデータベースサーバの性能劣化に伴い、データベースサーバを移行しなければならない状況が発生します。

▼

DBシステムのリプレースのために移行検討

様々な背景がある中で、移行するためには、適切な『目的』を定めることが重要です。

前提：データベース移行のメリット・デメリット

▶ 商用DBからOSSDBに移行することで、得られるメリットについて主なものを次にあげます。

メリット（移行することで得られるもの）

ライセンスコスト最適化	商用DBよりもライセンスコストが抑えられるOSSDBに移行することで、ライセンスコストが削減可能です。	課題①
ＤＢサーバの集約	余剰リソースがあるサーバに、DB移行する事で、集約が可能です。サーバの利用効率をあげ、メンテナンスコストなどが削減可能です。	その他
ＤＢサーバの性能向上	性能の良いサーバへの移行や、データベースを分割することで、データベースの性能が向上します。	その他

デメリット（移行に関する懸念含む）

移行計画の懸念	1.移行前後のコスト比較が難しい 2.移行先のサーバ選定は、どのようにすればよいか。 3.アプリケーションへの影響はどの程度なのか	課題①②
移行作業の懸念	1.本当に移行できるのか。	検証①
	2.移行にどのくらい時間がかかるのか。	検証②
	3.DBの性能は劣化しないのか。	検証③
製品サポートの懸念	1.サポートは十分なのか。 参考：『2011年度 国内オープンソースソフトウェア(OSS)利用実態調査 IDC調べ』	その他

解決策

検証	①移行可否の確認
	②移行時間の確認
	③性能の確認
課題	①移行前後のコスト確認
	②アプリケーションへの影響検討

▼

次ページから検証・課題を解決していきます。

その他

本日の講演では割愛させていただきますが、いつでもご相談ください。

移行可否を判定するための、検討事項は次の通りです。

1.事前準備　IT資産棚卸 → 移行対象のDB検討 → 移行先の環境検討

移行する事で、目的が達成可能なDBを選択します。
例えば、DBを集約したいのであれば利用率の少ないDBを対象します。本講演では、コスト削減を目的とし、
移行対象を某社DBとして進めます。

商用DBで提供されている機能の網羅性が高いため、
OSSDBの中では、PostgreSQLが最適です

2.運用・移行コスト試算

- 移行前後の各項目を比較
- **移行時だけのコストではなく、数年間運用することを想定して試算します**

DBライセンス費用	ハードウェア購入費用	電気料金
DB保守費用	ハードウェア保守費用	場所・建物費
運用管理費用	移行作業費（詳細は、次ページ）	

検証	①移行可否の確認
	②移行時間の確認
	③性能の確認
課題	①移行前後のコスト確認 ✔
	②アプリケーションへの影響検討

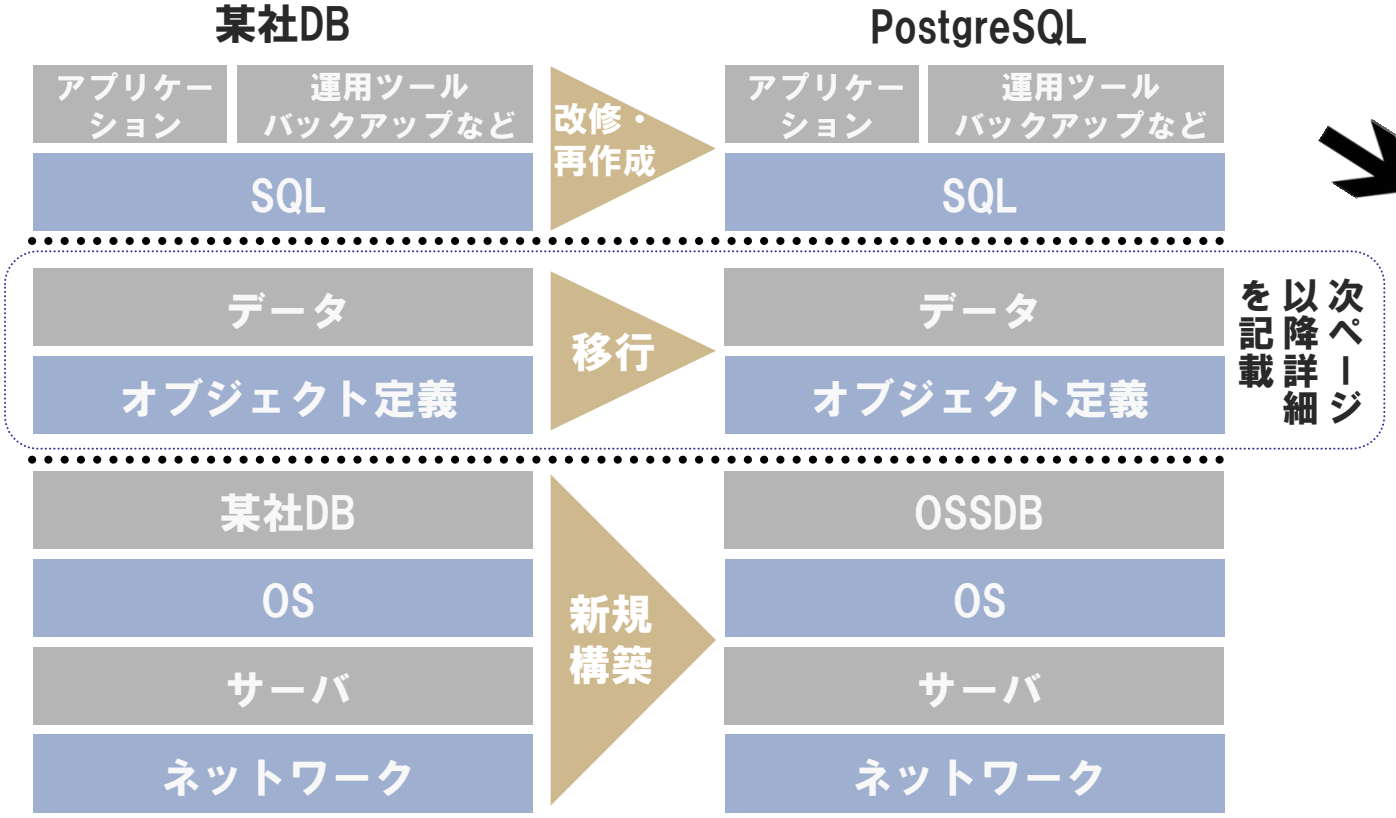
移行先の環境を検討する際は、**スペックも考慮**します。
最低限、次の条件を満たす必要があります。

CPU：
移行先 SpecInt値 ＞ 移行元 SpecInt値 合算値

メモリ・ディスク：
移行先 容量合算 ＞ 移行元 容量合算
※適宜、安全係数を考慮

試算が、『移行前 運用コスト　＞　移行後 運用コスト＋移行コスト』であれば、移行することでコスト削減できると考えられます。

データベース移行の概要を見ながら、移行に必要な作業を洗い出します。



検証
- ①移行可否の確認
- ②移行時間の確認
- ③性能の確認

課題
- ①移行前後のコスト確認 ✔
- ②アプリケーションへの影響検討 ✔

アプリケーションレイヤの影響範囲は環境によってさまざまです。
アプリケーション改修の見積もりについては、環境によって、入念に見積もる必要があります。

そのため、移行元・移行先のDBについて十分なノウハウが必要になります。

弊社では、アプリケーションへの影響を事前に調査し、お見積りさせていただきます。

各レイヤで作業を区切ることで、移行作業費用を細分化して検討する事が出来ます。

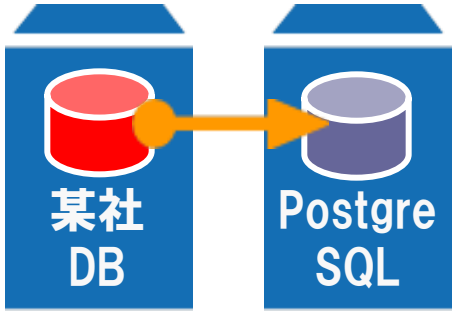
某社DBからPostgreSQLへ移行をするための、ツールを比較し最適なツールを選択します。

比較項目	ミドルウェア付属機能（CSVなどを利用したエクスポート・インポート機能）		某社DB用移行ツール（某社DBからPostgreSQL専用移行ツール）※以下、移行ツールと記載		有償移行ツール	
コスト	○	ミドルウェアに付属のため、無償	○	無償	△	有償
データ移行	×	手作業が多い	○	自動	○	自動
テーブル移行	△	手作業が多い 型を変換するための、規則を調査する必要がある	○	型変換が自動	○	型変換が自動
テーブル以外の移行	×	手作業が多い	△	一部対応のオブジェクトは、手動で移行	△	一部対応のオブジェクトは、手動で移行
インターフェース	△	CUI	△	CUI	○	GUI

コストを抑えながら、手作業が少ない無償の移行ツールを利用して、移行を進めます。

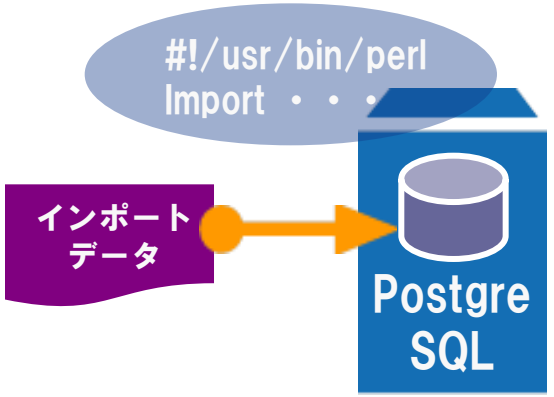
移行ツールの特徴について整理します。移行ツールは某社DBからPostgreSQLに移行できるPerlの移行ツールです。

某社DB -> PostgreSQL



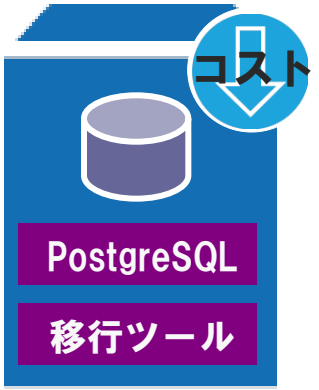
- 本移行ツールは、某社DBClientを利用し、ネットワーク越しに某社DBへ接続します。そのため、既存のサーバに変更不要
- 某社DBで定義されているオブジェクトを一括で抽出可能のため、移行対象のリストアップが不要

CUIを利用した自動移行



- 某社DBで稼働しているDDLをPostgreSQL用のDDLに変換するため、DDLの再作成が不要
- テーブル移行では、某社DBのデータ型をPostgreSQLのデータ型に自動変換

無償ツール



- コストを抑えながら、某社DBからPostgreSQLへの移行が可能
- 検証用に手軽に利用することも可能

データベース移行を検証するにあたって、次の３点を確認することを目的としています。

検証①：移行可否の確認

移行ツールを利用して次の2点について問題がない事を確認します。

1. 主要オブジェクトの移行
2. テーブルデータの移行

検証②：移行時間の確認

移行の中で、一番時間を消費するデータ移行の時間について確認します。次の2点を確認します。

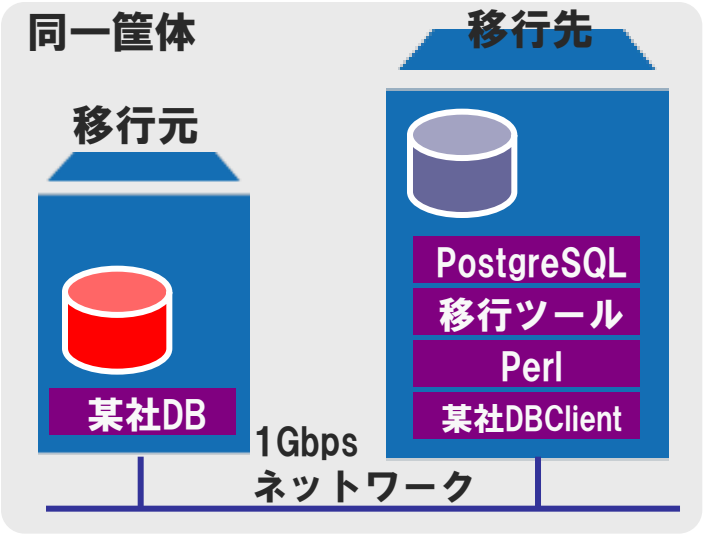
1. 移行ツールを利用したデータのエクスポート
2. PostgreSQLのインポートツールを利用したデータのインポート　※いくつかの方法で時間を計測します

検証③：性能の確認

移行元となる某社DBと移行先となるPostgreSQLの性能を比較します。

データベース移行を検証するにあたって、利用した検証環境について整理します。

- 仮想化ソフトウェアを利用して、移行元・移行先の仮想マシンを用意します。
- 移行元、先の仮想マシンは、同一筐体に配置。同じ仮想マシンスペックとします。



ハードウェア

#	カテゴリ	項目	スペック
1	ハードウェア	サーバ	HP ProLiant BL460c G7
2		CPU	Xeon E5506 2.13Ghz 2CPU/8cores
3		メモリ	22G
4		ディスク	256GB
5		仮想化ソフト	ESXi5.0
6	仮想マシン	CPU	1 CPU/4cores
7		メモリ	4G
8		ディスク	50G

ソフトウェア

移行元

#	項目	スペック
1	ＯＳ	OEL 5.7
2	ＤＢソフトウェア	某社DB

移行先

#	項目	スペック
1	ＯＳ	OEL 5.7
2	ＤＢソフトウェア	PostgreSQL 9.1.2
3	移行ツール	8.9
4	Perl	5.8.8

移行ツールを使用し、移行できるオブジェクトと移行できないオブジェクトを整理します。

#	主要オブジェクト	移行可否	移行できない場合の対策方法
1	テーブル DDL/データ	○	次ページ以降で検証
2	インデックス/ビュー/シーケンス/制約	○	次ページ以降で検証
3	マテリアライズドビュー	×	PostgreSQLにオブジェクトが存在しません。 View作成か、『Create Table AS <SQL>』で対応。
4	シノニム	×	PostgreSQLにオブジェクトが存在しません。 別スキーマのオブジェクト権限を付与し、<スキーマ>.<オブジェクト>でアクセスするなどで、対応。
5	データベースリンク	×	dblinkモジュールの利用。PostgreSQL同士のdblinkのみ可能。 異種DBでのdblinkでは、接続先のオブジェクトの移行も検討。
6	表領域	○	ただし、PostgreSQLと某社DBでは、概念が異なります。 PostgreSQLに適した設計が必要。
7	ユーザ/オブジェクト権限	○	次ページ以降で検証
8	ロール	○	次ページ以降で検証
9	プロファイル	×	PostgreSQLでリソースの制限は、不可。OSのレイヤで制限するか、運用で回避する必要があります。

移行ツールのインストール・使い方について整理します。ここでは、PostgreSQLサーバにインストールします。

1. 移行ツールのインストール

```
[root@host2 tmp]# tar zxf <移行ツール zip>
[root@host2 tmp]# tar xvf <移行ツール tar>
[root@host2 tmp]# cd <移行ツール ディレクトリ>
[root@host2 tmp]# perl Makefile.PL
[root@host2 tmp]# make
[root@host2 tmp]# make test
[root@host2 tmp]# make install
```

2. 実行ファイル準備

※次ページで詳細を説明します

3. 実行

```
[root@host2 <移行ツール ディレクトリ>]# perl export_data.conf
```

ポイント①
2で作成した実行ファイルを指定

■移行ツール インストールの前提条件

1. Perlがバージョン5以上であること
2. 移行ツールで必要となるPerlモジュールがインストール済みであること

   インストール例：

```
[root@host tmp]# perl -MCPAN -e shell
cpan> install DBI
```

3. 某社DBのClientがインストール済みであること

移行ツールで使用する実行ファイルについて整理します。エクスポートするオブジェクトにあわせて作成します。

2. 実行ファイル準備の詳細

```
export_dml.conf
#!/usr/bin/perl
BEGIN {
$ENV{DB_HOME}=〈某社DBホーム〉
$ENV{NLS_LANG}='Japanese_Japan.AL32UTF8';
}
use strict;
use 〈移行ツール用モジュール〉;
my $schema=new 〈移行ツールモジュール〉(
datasource=> 'dbi:〈某社DB〉:host=〈DBサーバIP〉;sid=〈SID〉;port=〈ポート〉',
user=> 〈管理者ユーザ〉,
password=>'password',
schema=>'USER01',
type=>'TABLE',
debug=>1
);
$schema->export_data('/tmp/user01_TABLE.sql');
exit(0);
```

ポイント②
移行元となる
某社DBの情報

ポイント③
TYPE句には、エクスポートする
オブジェクトを指定
詳細は、右の一覧に記載

ポイント④
DDL・データの出力先。TYPE句
で指定された全オブジェクトの
定義が同一ファイルに出力

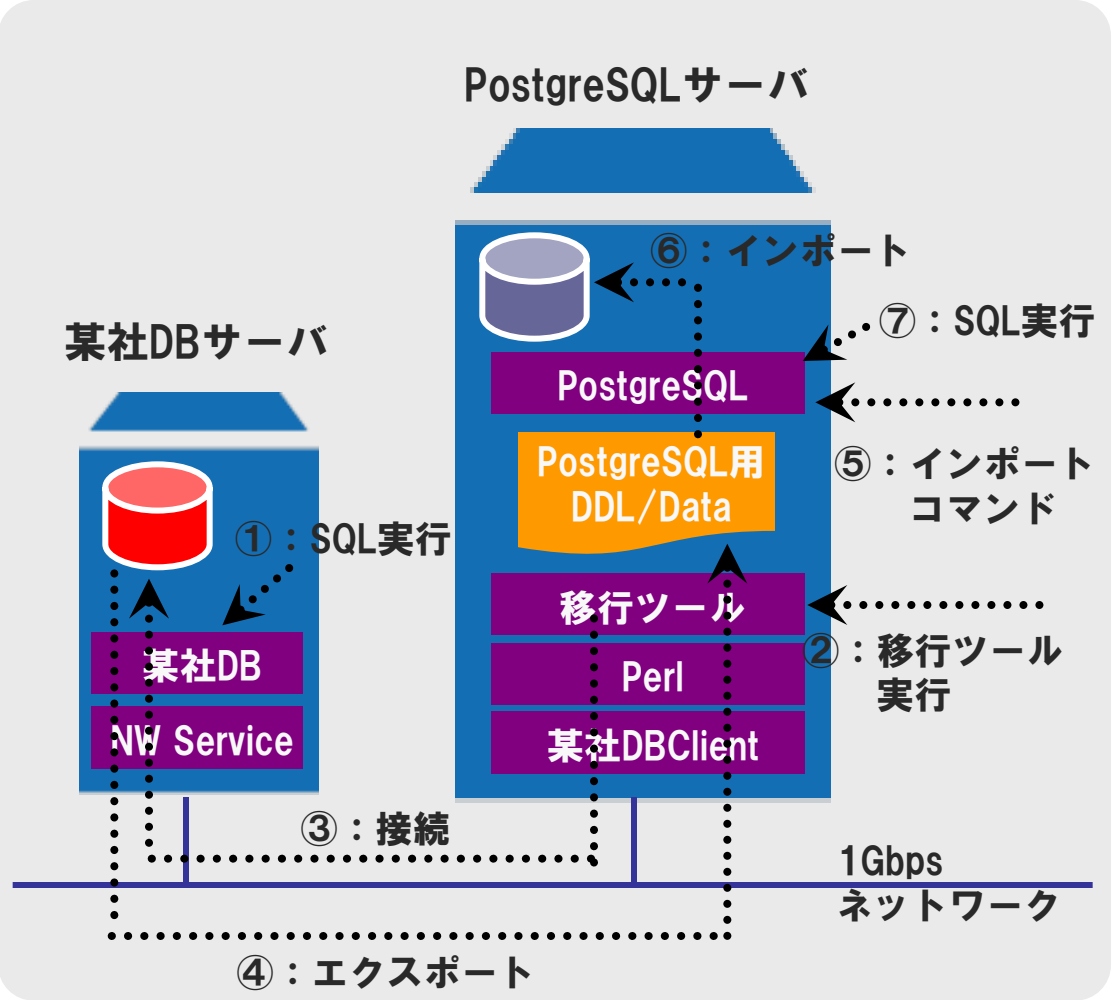
■オブジェクト別のTYPE句

#	主要オブジェクト	TYPE指定
1	テーブルDDL/インデックス/制約	TABLE
2	テーブルデータ	DATA or COPY
3	ビュー	VIEW
4	シーケンス	SEQUENCE
5	表領域	TABLESPACE
6	ユーザ/オブジェクト権限/ロール	GRANT

『file_per_table=>1』を追加することで、
オブジェクト毎にファイルが分割されます。

※ファイルは、export_dataで指定したディレクトリに
『<オブジェクト名>_』という名称で出力されます。
※TYPEをTABLEで指定した場合は、テーブル毎の
出力ファイル分割ができません。

▶ 移行可否を確認するに当たって、『主要オブジェクトの移行』、『テーブルデータの移行』の２点を確認します



主要オブジェクトの移行

1：移行オブジェクト確認　①
2：エクスポート　②、③、④
3：インポート　⑤、⑥
4：結果確認　⑦

#	主要オブジェクト
1	テーブルDDL
2	テーブルデータ
3	インデックス
4	ビュー
5	シーケンス
6	制約
7	表領域
8	ユーザ/オブジェクト権限/ロール

テーブルデータの移行

1：移行前のオブジェクト情報確認　①
2：移行後のオブジェクト情報確認　⑦

『主要オブジェクトの移行』を確認します。最初にテーブルDDLの移行を行い、正常に移行できたことを確認します。

1：移行オブジェクト確認　①

```
SQL> select table_name from <テーブル確認用カタログ>;

TABLE_NAME
--------------------
SALES_H
SALES_I
SALES_M1

3行が選択されました。
```

ポイント①
移行元となるテーブル一覧。
他のオブジェクトの確認方法については後述。

```
SQL> <テーブル確認コマンド> SALES_I
 名前                                      NULL?    型
 ----------------------------------------- -------- ------------
 CNTR_NUM                                  NOT NULL NUMBER(8)
 MONTH                                     NOT NULL NUMBER(2)
 USAGE_FEE                                          NUMBER(6)
```

ポイント②
移行元となるテーブル定義

2：エクスポート　②、③、④

```
[root@host2 <移行ツール ディレクトリ>]# perl export_data.conf
```

- 1の詳細
  移行前、某社DBで次の情報を取得します。
  - 移行オブジェクトの一覧
  - 移行オブジェクトの定義

  移行後に、PostgreSQLの情報と一致していることを確認するために、情報を取得します。
  ここでは、SALES_Iテーブルを移行します。

- 2の詳細
  適宜、TYPE句を指定を修正した実行ファイルを使用して、エクスポートします。

▶ 前ページからの続きで、テーブル DDLが正常に移行できたことを確認します。

3：インポート　⑤、⑥

```
[root@host2 bin]# ./psql test
test=# psql => ¥i /tmp/user01_TABLE.sql
```

4：結果確認　⑦

```
test=# select * from pg_tables where tablename
       like 'sales%' order by tablename;
 schemaname | tablename | tableowner | tablespace | hasindexes | hasrules | hastriggers
------------+-----------+------------+------------+------------+----------+-------------
 public     | sales_i   | postgres   |            | t          | f        | t
(1 rows)
```

ポイント③
テーブルの移行を確認

```
test=# ¥d sales_i
        Table "public.sales_i"
   Column    |     Type      | Modifiers
-------------+---------------+-----------
 cntr_num    | numeric(8,0)  | not null
 month       | numeric(2,0)  | not null
 usage_fee   | numeric(6,0)  |
```

ポイント④
テーブル型の移行を確認。
他のオブジェクトの確認方法については後述。

- 3の詳細
  2で作成されたDDLファイルを指定し、オブジェクトをインポートします。

- 4の詳細
  移行後、PostgreSQLで次の情報を取得します。
  - 移行オブジェクトの一覧
  - 移行オブジェクトの定義

  1で取得した情報と一致することを確認します。

  テーブル DDLが正常に移行できたことが確認できました。

  同様の手順で、全てのオブジェクトを移行します。

テーブル DDLと同様に、他オブジェクトについても確認しました

#	主要オブジェクト	移行可否	某社DB 移行オブジェクトの確認		PostgreSQL 移行の結果確認	
			移行対象一覧	定義	移行対象一覧	定義
1	テーブルDDL	○			pg_tables	¥d <テーブル名>
2	インデックス	○			pg_indexes	pg_indexes index_def列を確認
3	ビュー	○			pg_views	pg_views definition列を確認
4	シーケンス	○			Pg_class relkind列がSを確認	¥d <シーケンス名>
5	制約	○			information_schema.table_constraints	information_schema.table_constraints
6	表領域	△ (※1)			¥db	¥y db+
7	ユーザ	△ (※2)			pg_shadow	pg_user
8	ロール	△ (※3)			pg_roles	¥z
9	ｵﾌﾞｼﾞｪｸﾄ権限	○			¥z	

データベースのカタログ情報から
移行対象一覧・定義を取得。

データベース製品で
カタログ情報の取得方法が
異なります。
該当データベース製品に
あわせてカタログ情報を
取得してください。

再設計

再設計

再定義

(※1) 某社DBの表領域は、ファイルの集合となっています。PostgreSQLの表領域は、ディレクトリとなっています。そのため、PostgreSQLは、テーブルを格納する場合、1つのディレクトリとなってしまいます。ディスクI/Oを分散させるには再設計が必要となります。
(※2) 某社DBでは、クォータ・表領域関連の設定が可能です。PostgreSQLでは、クォータについては、OSで設定し、表領域関連については、クライアントの接続の際に、default_tablespace、temp_tablespacesに値をセットします。
(※3) ロール定義（ロールに割り当てられた権限情報）は移行するが、ロールを割り当てたユーザの情報は移行しません。
そのため、移行後適宜、ユーザにロールの割り当てが必要になります。

『テーブルデータの移行』について確認します。主要なテーブル型のデータが正常に移行できたことを確認します。

1：移行前のオブジェクト情報確認　①

```
SQL> <テーブル確認コマンド> てすと用テーブル;
 名前                                    NULL?    型
 --------------------------------------- -------- ----------------------------
 A                                                DATE
 B                                                TIMESTAMP(6)
 C                                                LONG
 D                                                CHAR(12)
SQL> select * from てすと用テーブル;
A                   B                   C                   D
------------------- ------------------- ------------------- ----------
20120120030405      20110105060708      21                  てすと１
20121223155959      20110108070809      21                  てすと２
```

ポイント①
移行対象テーブル定義を確認

ポイント②
移行対象のデータ確認

2：移行後のオブジェクト情報確認　⑦

```
test-# ¥d てすと用テーブル
        Table "public.てすと用テーブル"
 Column |            Type             | Modifiers
--------+-----------------------------+-----------
 a      | timestamp without time zone |
 b      | timestamp without time zone |
 c      | text                        |
 d      | character(12)               |

test-# select * from てすと用テーブル;
          a          |          b          | c  |      d
---------------------+---------------------+----+--------------
 2012-01-20 03:04:05 | 2011-01-05 06:07:08 | 21 | てすと１
 2012-12-23 15:59:59 | 2011-01-08 07:08:09 | 21 | てすと２
```

ポイント③
テーブルの移行を確認

ポイント④
データの移行を確認

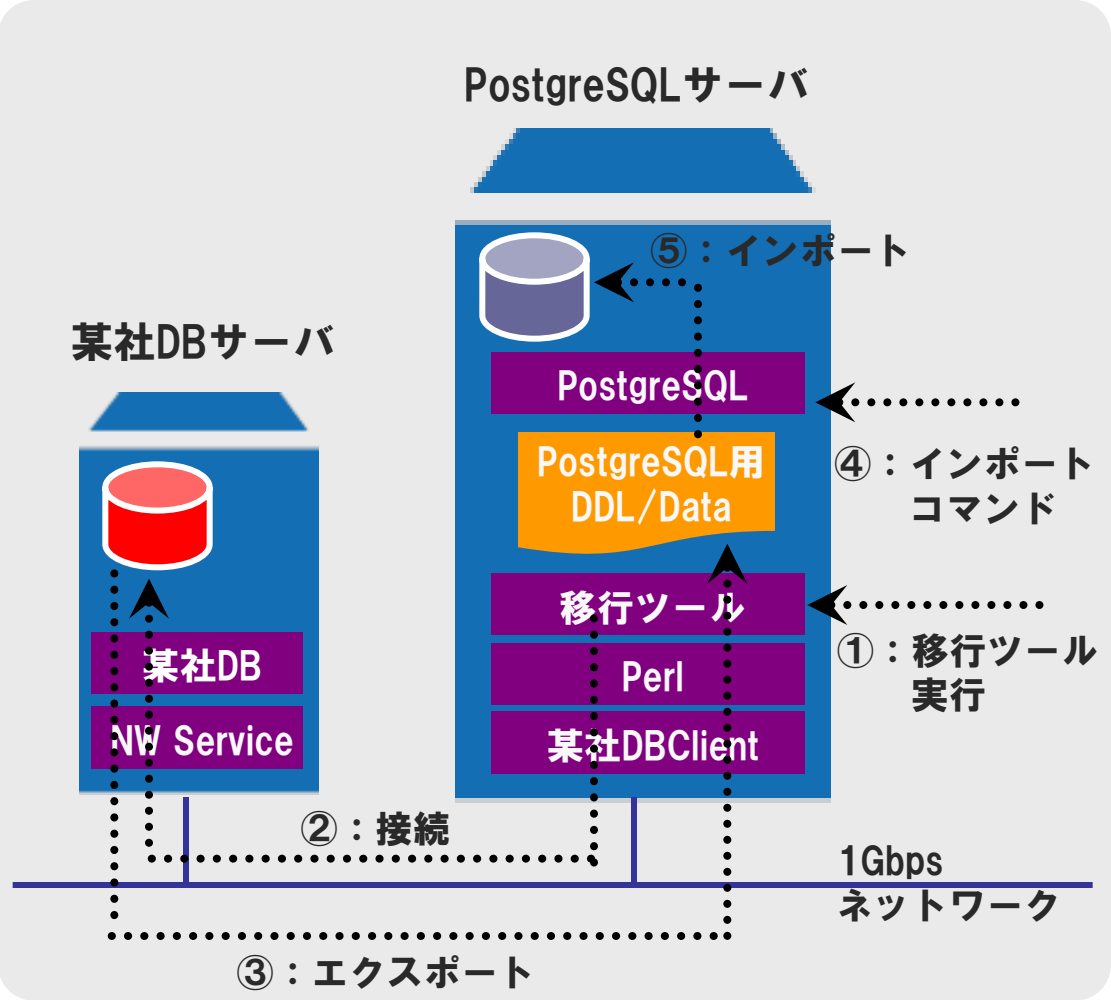
- 1の詳細
  移行前、某社DBで次の情報を取得します。
  - 移行テーブルの定義・データ

  移行後に、PostgreSQLの情報と一致していることを確認するために、情報を取得します。

- 2の詳細
  1で確認した情報と一致することを確認します。

■考察
移行ツールを利用して、効率的な移行が可能であることを確認。
移行ツールで移行できないものや、某社DB、PostgreSQLの違いにより、再度設計しなければならない個所についても整理できました。

データの移行時間を確認するに当たって、検証の手順を整理します。



検証②：移行時間の確認

1：エクスポート時間計測 ①、②、③

2：インポート時間計測　④、⑤

次の2つのテーブルを利用して、時間を計測します。

テーブルA：1000万件
296バイト/レコード
テーブルB：1億件
16バイト/レコード

※いずれも、INDEXあり

PostgreSQLのインポート方法について整理します。主に利用される次の3つについて比較します。

#	移行方法（インポート）	速度	利用する手間	実行時の更新ファイル	制限	エクスポートする際の移行ツールのTYPE句
1	SQL	×	○（※1）	1.データファイル 2.WAL 3.インテックス	なし	TYPE=DATA
2	COPY	△	○（※1）	1.データファイル 2.WAL 3.インテックス	なし	TYPE=COPY
3	pg_bulkload	◎	×（要インストール）	1.データファイル 2.インテックス ※移行後に再構成	1. 実行中のテーブルへの更新が不可 2. 参照整合性制約が適用されない 3. WALを更新しないため、PITRが不可	TYPE=COPY ※COPYと同様

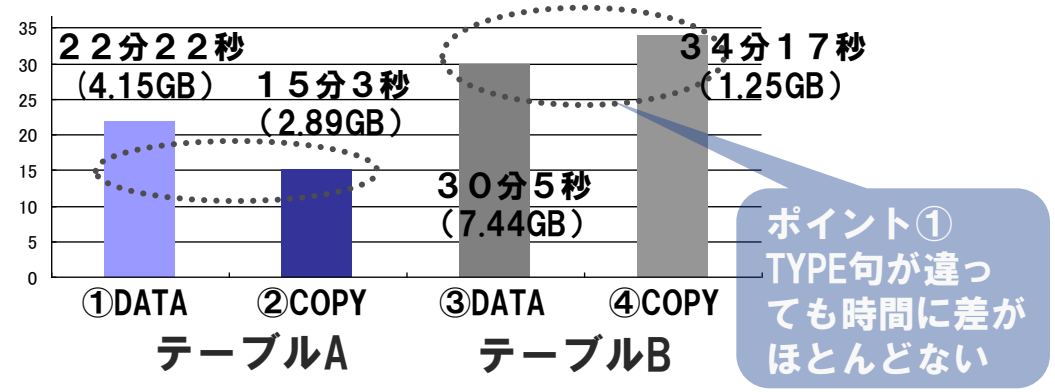
（※1）PotgreSQLをインストールすれば利用可能）

いずれの制限についても、運用中であれば問題になる恐れもありますが、移行作業では、大きな問題になりません

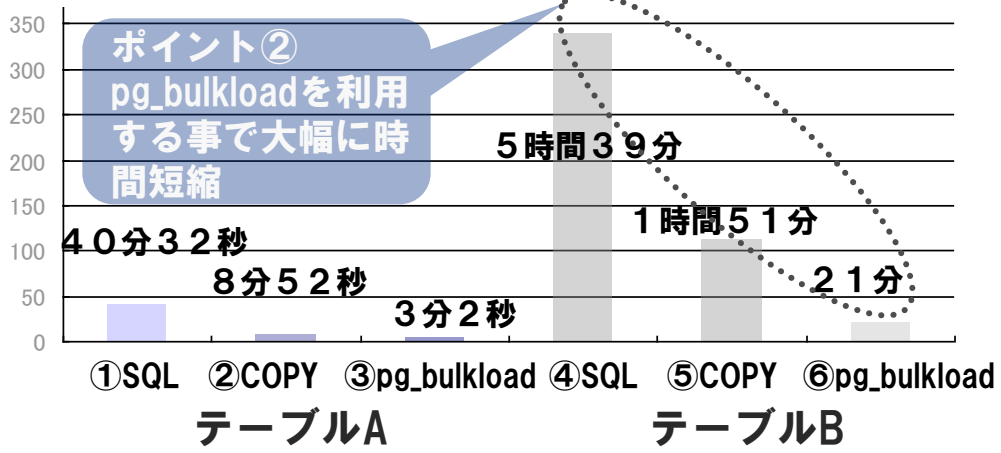
上記の通り、3 > 2 > 1の順で高速にインポートが確認できた場合、移行には、pg_bulkloadが最適

▶ 某社DBからPostgreSQLテーブル・データの移行について実際の手順を見ていきます。

②-1：エクスポート時間計測　①、②、③



②-2：インポート時間計測　④、⑤



■考察

TYPEの違いにより、エクスポート時間に大きな差はありませんでした。

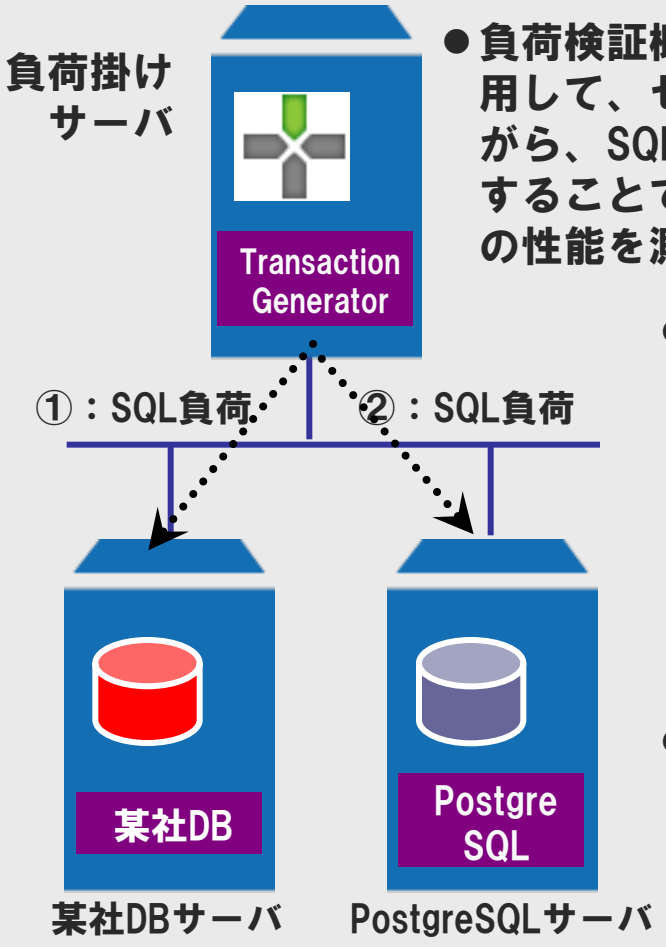
- DATAは、SQL文がそのまま記載されているため、ファイルサイズが大きくなるため、ディスク容量に注意が必要です。
- エクスポート時間に大きな差がないため、インポート方法にあわせて、エクスポート方法を選択する必要があります。

■考察

仮説通り、3 > 2 > 1の順で高速にインポートできました。

そのため、可能な限りpg_bulkloadを利用する事を推奨します。

データベースの性能を確認するに当たって、検証の構成・手順について説明します。



- 負荷検証概要：負荷ツールを利用して、セッションを増やしながら、SQLを実行します。そうすることで、データベース単体の性能を測定します
- 負荷：OLTPを想定した負荷1分間負荷を継続
  - －SELECT　80%
  - －INSERT　10%
  - －UPDATE　5%
  - －DELETE　5%
- 仮想、物理のCPU負荷・TPSを計測します。

検証③：性能の確認

③-1：処理時間の比較　①、②

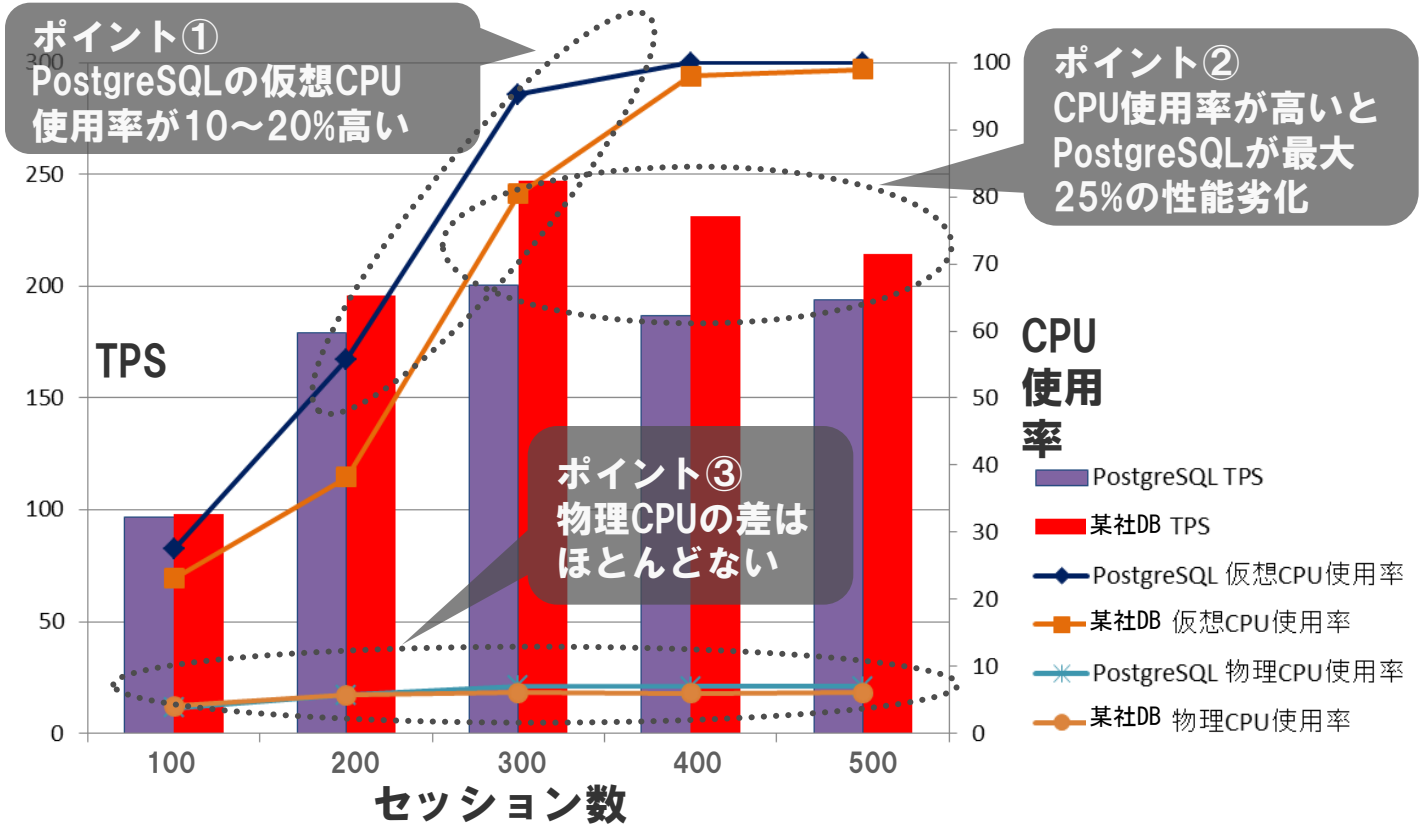
某社DBとPostgreSQL（ソースからインストール）をデフォルトでローカルディスクにインストールしたものを利用。

PostgreSQLの共有メモリ設定を某社DBに併せます。

- PostgreSQL
  shared_buffers = 572MB
- 某社DB
  <共有メモリ> = 572MB（デフォルト）

Transaction Generator：
http://www.transactiongenerator.com/

PostgreSQLと某社DBに負荷をかけ、TPS、CPU使用率を比較します。



■考察

某社DBと比べると、PostgreSQLは、性能劣化が発生します。

セッション200までは、PostgreSQLは某社DBと比べ、CPU使用率は高いものの、TPSはほとんど変わっていないません。

そのため、PostgreSQLは、CPU60%位までなら、某社DBと同等の性能と考えられます。

PostgreSQLに移行することで、CPU使用率が高くなると、性能劣化が発生します。

PostgreSQLの性能劣化の原因を調査し、対策方法を検討します。

某社DB セッション300 vmstat抜粋

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs  us sy id wa st
2 30   6476  32712   6172 2131580    0    0   606   772  193  171   3  1 84 12  0
0 14   6476  31456   6184 2112732    0    0  4916   143 3304 2210   2  2 23 73  0
0 25   6476  31952   6180 2097980    0    0  4970   138 3347 2209   2  2 16 80  0
1 24   6476  33564   6196 2086532    0    0  4627   152 3137 2213   2  2 33 64  0
1 32   6476  31952   6196 2081012    0    0  5439   124 3142 2208   2  2 25 71  0
```

PostgreSQL セッション300 vmstat抜粋

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us  sy id wa st
0 87  27844  31128  62940 2845616    0    0  3981   328 2924 2584  2   2  1 96  0
0 80  27844  31756  62952 2828160    0    0  4387   260 2872 2566  2   2  8 89  0
0 83  27844  31260  62960 2811884    0    0  4315   315 2926 2645  2   2  6 91  0
0 100 27844  30888  62972 2795928    0    0  4391   285 2847 2553  1   1  4 93  0
0 101 27844  31872  62992 2786764    0    0  3941   321 2751 2540  1   1  5 92  0
```

ポイント④
PostgreSQLの方がI/Oウェイトが高い　⇒ディスクI/Oの効率化

CPU使用率が高い場合の性能劣化の原因を調査します。

PostgreSQLは、某社DBに比べて、I/Oウェイトが高い。

キャッシュヒットの悪化によりI/Oが高くなっている可能性があるため、キャッシュヒット率を調べます。

- PostgreSQL　：99%
- 某社DB　　：98.77%

キャッシュヒット率には特に問題がありませんでした。

PostgreSQLは、ディスクI/Oがボトルネックになっています。

前頁からの続きで、PostgreSQLの性能劣化の原因を調査し、対策方法を検討します。

某社DB セッション100 vmstat抜粋

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs  us sy id wa st
 0  2      0  30228   9992 3042632    0    0   966   136 1438 1095  1  1 76 23  0
 0  0      0  31344  10016 3033336    0    0   970   123 1368 1062  1  1 77 21  0
 0  2      0  30980  10044 3032036    0    0   980   134 1356 1065  1  1 76 22  0
 0  1      0  30708  10060 3021720    0    0   958   159 1389 1086  1  1 74 24  0
 1  0      0  31956  10068 3016892    0    0  1108   155 1323 1045  1  1 76 22  0
```

PostgreSQL セッション100 vmstat抜粋

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs  us sy id wa st
 0  0  27844  31268  61360 3300384    0    0  1424   243 1685 1855  1  1 70 29  0
 0  3  27844  32624  61372 3291936    0    0  1527   265 1683 1861  1  1 73 25  0
 0  2  27844  29896  61380 3288824    0    0  1440   224 1631 1865  1  1 74 25  0
 0  0  27844  31764  61392 3281328    0    0  1491   237 1669 1846  1  1 73 26  0
 0  1  27844  30648  61400 3276404    0    0  1468   197 1636 1827  1  1 73 25  0
```

ポイント⑤
PostgreSQLの方がディスクの読み込みが1.5倍以上になっている

PostgreSQLは、某社DBに比べキャッシュヒット率が高いにも関わらず、I/Oウェイトが多い、という事は、某社DBと同様の処理を行っていても、ディスクI/Oが多いと考えられます。

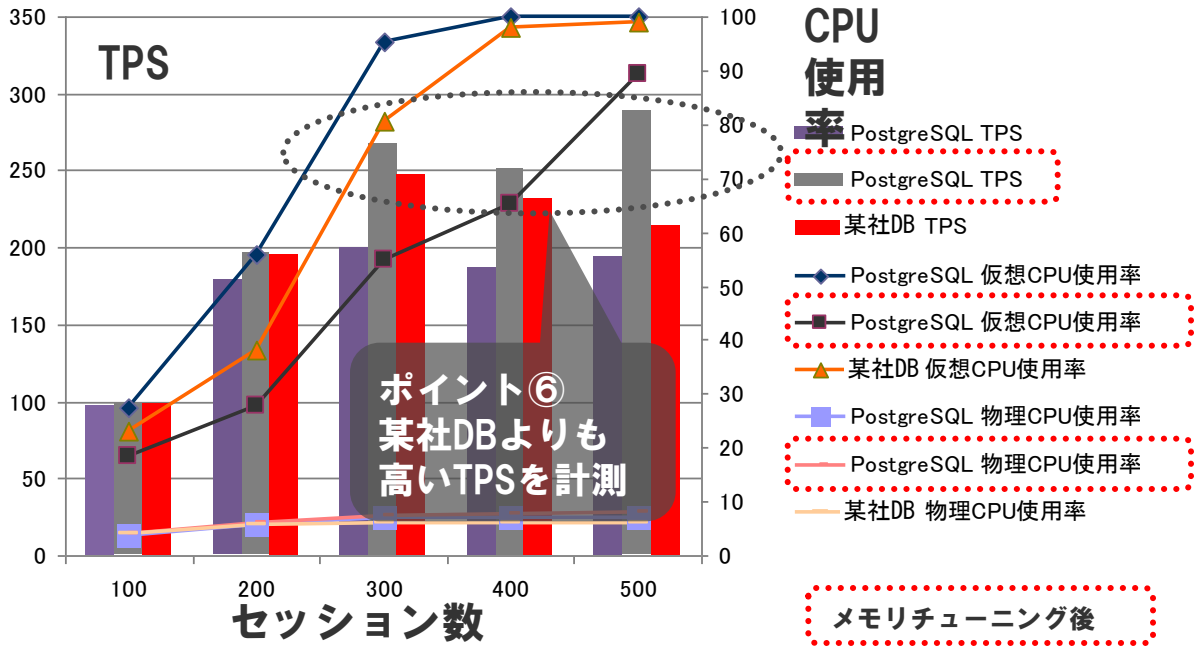
そのため、同程度TPSを計測した、セッション100の情報を確認します。（ポイント⑤）

同じテーブル型、データ件数にも関わらず、テーブルのディスクサイズが大きく異っています。

某社DB　　：2.1GB
PostgreSQL ：5.2GB（2.5倍）

PostgreSQLは、データの格納効率が良くないため、メモリを圧迫し、大量のディスクI/Oが発生。

PostgreSQLで、ディスクI/Oを分散させた状態で性能を比較します。



ディスクI/Oを効率化するために、次の2点が考えられます。

1. ディスクI/O 回数削減
   メモリの増強
2. ディスクI/O 効率化
   高速ディスクの利用・ディスク分散

本検証環境は、仮想化環境のため、2が難しいため、1のメモリの増強を実施します。

PosgreSQLの方がディスク容量が2.5倍増えているため、同程度のTPSを出すためには、メモリについても2.5倍必要と考えられます。

次の通り、メモリを増強し、性能を比較します。
仮想メモリ：10G（当初は、4G）
shared_buffers = 1.25GB　（当初は、572MB）

移行を検討する場合、某社DBサーバで使用していたメモリよりも多くのメモリを搭載したサーバを利用する事を推奨します。
ディスク容量とメモリの割合でメモリサイズを検討します。

PostgreSQLに適切な対策を施すことで、ディスクI/Oのボトルネックが解消されます。結果、性能劣化が抑えることが可能です。

検証状況の共有について、３つの重要なポイントにまとめます。

移行作業の実現性

- 移行ツールを利用した移行で大部分のオブジェクトが移行できました。
- 移行ツールで移行出来ないオブジェクトについても移行の対応策が整理できました

▼

移行ツールで効率的な移行が実現可能

移行時間の見積もり

- エクスポートは、各方式で時間の差は少ないものの、大量データの場合にはその差が許容できなくなることもある。インポート方法とセットで検討する必要があります。
- インポート速度だけを見ると、pg_bulkloadが最速でした。

▼

pg_bulkloadを利用することで時間短縮

性能比較

- PostgreSQLは、某社DBと比べると基本性能が劣る部分がありますが、適切な対策を施す事で、某社DB以上の性能が出せます。
- 移行元の某社DBサーバよりもメモリを多めに搭載する必要があります。

▼

I/Oボトルネックの解消がポイント

本検証環境では、
大きな問題なく移行が完了しました。

補足：株式会社インテリジェンス ビジネスソリューションズ ご紹介

事業内容

✓ システムインテグレーション（SI）事業
→ビジネスコンサルティングからシステム開発、保守、アウトソーシング受託まで

基本情報

✓ 設立	：1982年4月設立
✓ 資本金	：90百万円
✓ 代表取締役社長	：小澤稔弘
✓ 売上	：約25億円（2010年2月末決算）
✓ 従業員数	：315名（2010年8月1日現在）
✓ 事業拠点	：東京、幕張（開発拠点）、ベトナム（オフショア開発センター）
✓ 本社所在地	：東京都台東区柳橋1-4-4 ツイントラスビル3F
✓ 認証取得	：ISO9001、ISO27001
✓ ホームページ	：http://www.ibs.inte.co.jp
✓お問い合わせ先	：https://portal.ibs.inte.co.jp/inquiry/service

主なパートナーシップ

NTTデータ（Hinemos）、 VMware（ソリューション）、オラクル（SOAコンサルティング）、RedHat（JBOSS）、ミラクル・リナックス（ZABBIX）、マイクロソフト（ソリューション）、IIJ（クラウドサービス）

最後に、本日の講演『商用DBからPostgreSQLへの移行』について３つのポイントにまとめます。

コスト・計画

- データベースを移行する事でコスト・性能の向上が実現できる事を事前に確認する必要があります。
- 移行前後のコストを試算するためには、調査しなければならない事項が多く存在します。

▼

移行前の入念な検討が重要

性能

- 移行ツールを利用すれば、効率的な移行が可能です。
- ただし、某社DBとPostgreSQLでは、一部実装が異なり、再設計が必要になる場合があります。
- 性能に関しては、デフォルトの状態では、若干の劣化が発生します。適切な対策により、性能の向上が可能です。

▼

性能劣化、再設計部分を把握し、これらを踏まえたプランが重要

弊社OSS関連サービス

- 商用DBからPostgreSQLへの移行・統合サービスをご用意しております。
- 商用DBからPostgreSQLの移行についてアセスメントしたいなど、お見積もりさせていただきます。
- その他OSSを利用したサービスもご用意しております。

▼

いつでもご相談ください

ご清聴ありがとうございました