オンラインリカバリを実装した pgpool-II 新バージョンのご紹介

SRA OSS, Inc. 日本支社
技術部
浅羽 義之

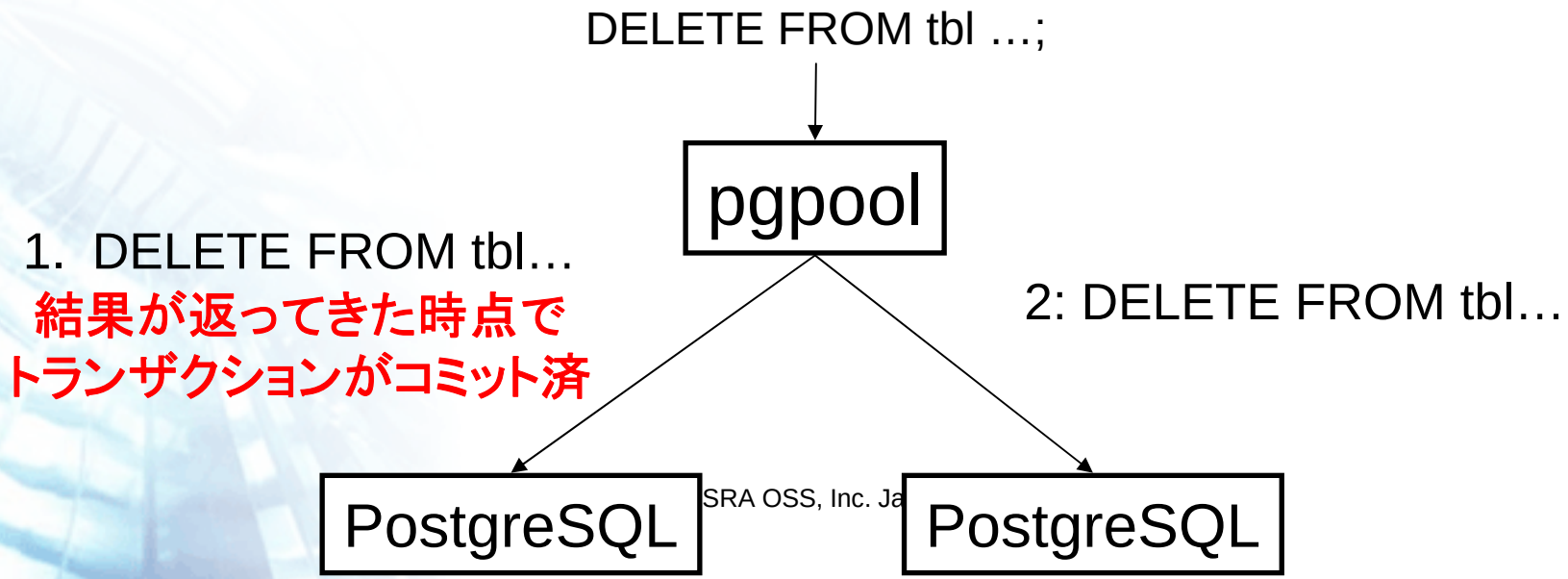
pgpool-II 2.0概要

- 11/16 リリース
  - http://pgfoundry.org/frs/download.php/1521/pgpool-II-2.0.1.tar.gz
- 様々な改良を追加
  - レプリケーション
    - 信頼性向上、高速化、オンラインリカバリ
  - パラレルクエリ
    - 高速化
  - 全般
    - 他のシステムとの連携

レプリケーション

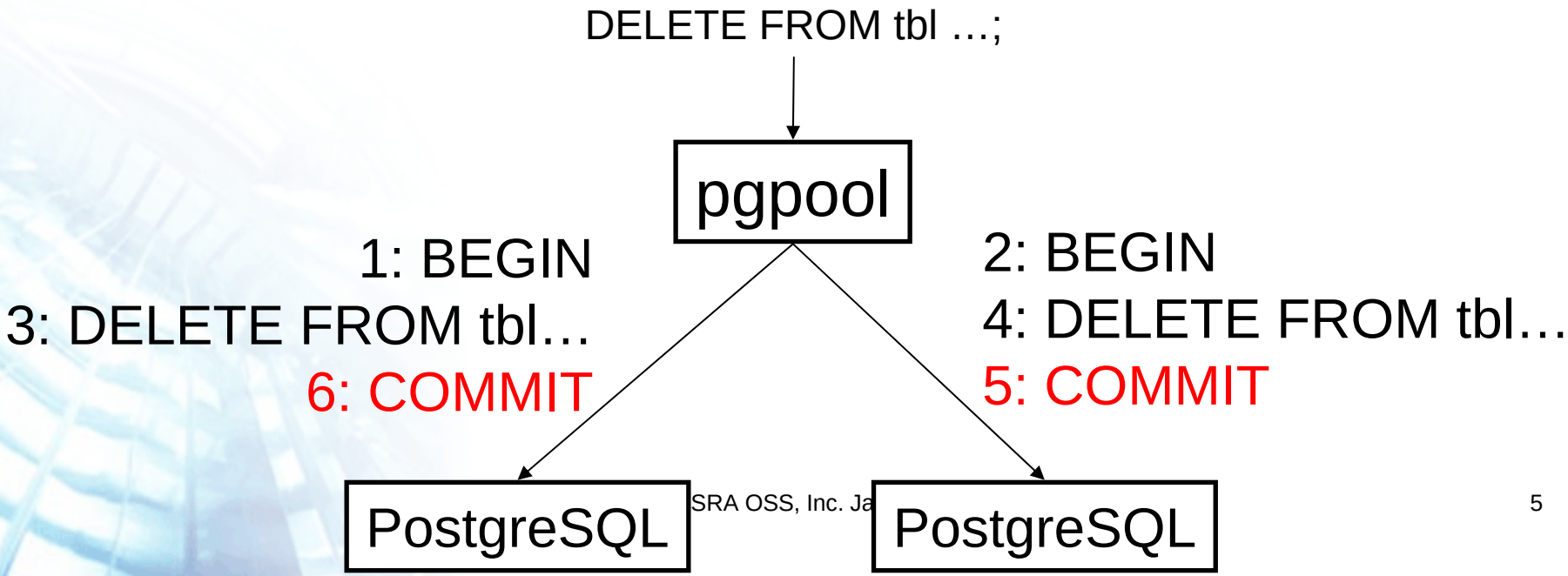
レプリケーション高信頼化(1)

- pgpool-II 1.xでの問題
  - autocommit の場合、セカンダリに送信する前にテーブルロックや行ロックが解放されてしまう
    - 非常に短い時間であるが、セカンダリのロック獲得を別のユーザが割り込める可能性がある



レプリケーション高信頼化(2)

- 解決策
  - 内部的にトランザクションを開始させ、マスタを最後にコミットさせることで、セカンダリに先にロックを解放させる
    - セカンダリのロック獲得の順番がずれないことを保証



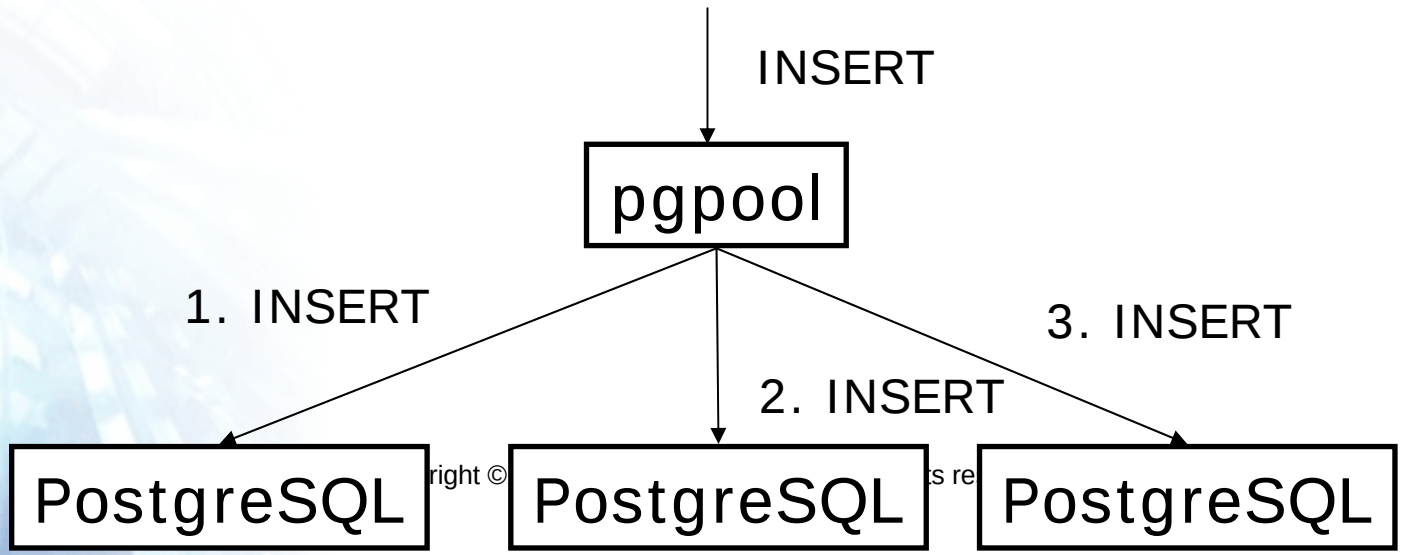
レプリケーション高信頼化(2)

- pgpool-II 1.x までは、更新の成功・失敗のみでデータの整合性を判断
  - 更新数が異なっているとデータの不整合に気付かない
- 更新数の確認(pgpool-II 2.0)
  - INSERT,UPDATE,DELETEの件数が一致していない場合はトランザクションをアボート
  - 不一致が発生した場合にはオンラインリカバリで復旧可能

```
x=# update t set a = a + 1;
ERROR:  pgpool detected difference of the number of update tuples
HINT:  check data consistency between master and other db node
```

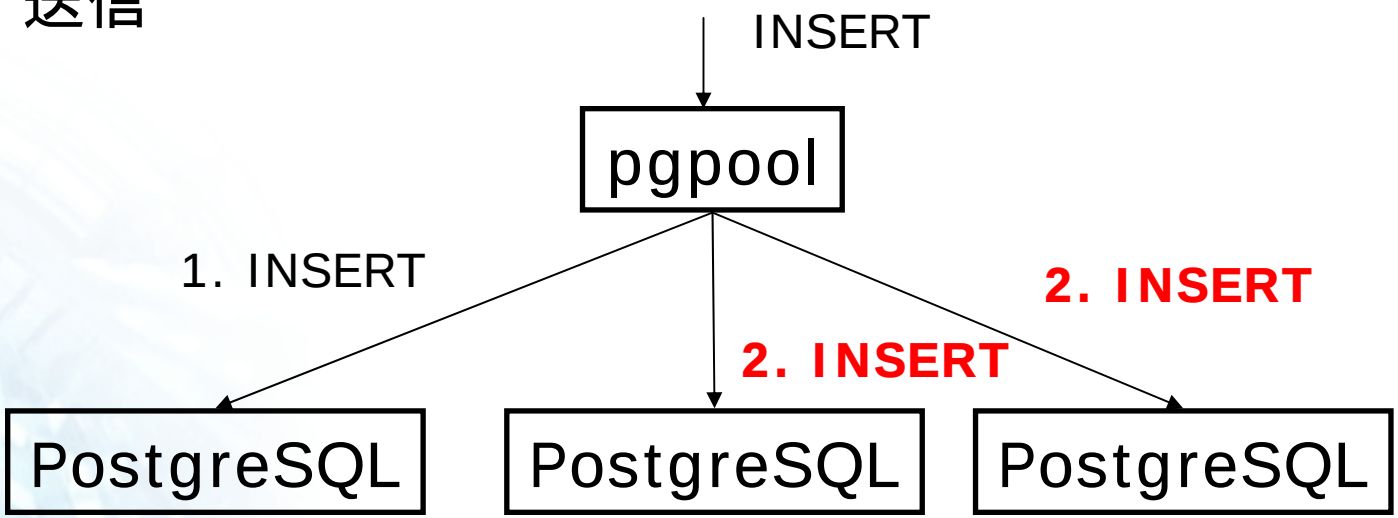
レプリケーション高速化(1)

- pgpool-II 1.xのレプリケーション
  - 順番にクエリを実行(replicate_strict = true)
    - 応答が返ってきたら次のノードへクエリを送信する仕組み
    - 応答が返ってくる前にクエリを送信するとデッドロックする可能性がある
  - ノード数が増えるだけ時間がかかる問題がある



レプリケーション高速化(2)

- pgpool-II 2.0のレプリケーション
  - マスタとそれ以外で分類
  - マスタのクエリ完了後にその他のノードに並列にクエリを送信



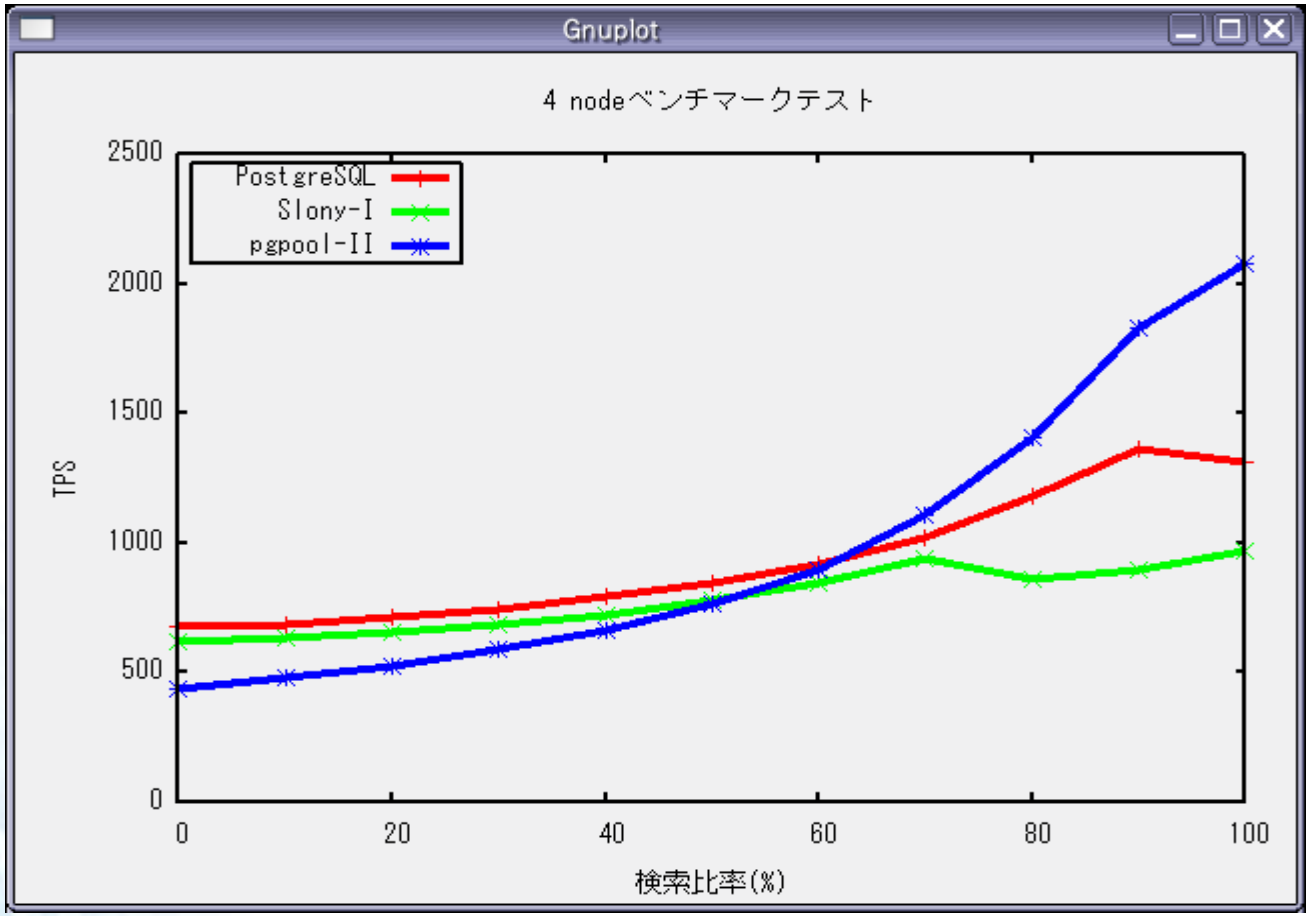
ノード数が増えてもPostgreSQL単体の
約2倍でレプリケーションが完了

レプリケーションのベンチマーク

- pgpool-II,PostgreSQL1台,Slony-Iで比較
- データ件数: 10万件, fill factor 50(pgbench -i -s 1 -F 50 で初期化)
- ノード数は4, レプリケーションモード, 負荷分散あり
- pgbenchパラメータ
  - pgbench -s 1 -c 80 -t 1000 -f ファイル
  - スクリプトファイルは, s00w10-s10w00を順に実行

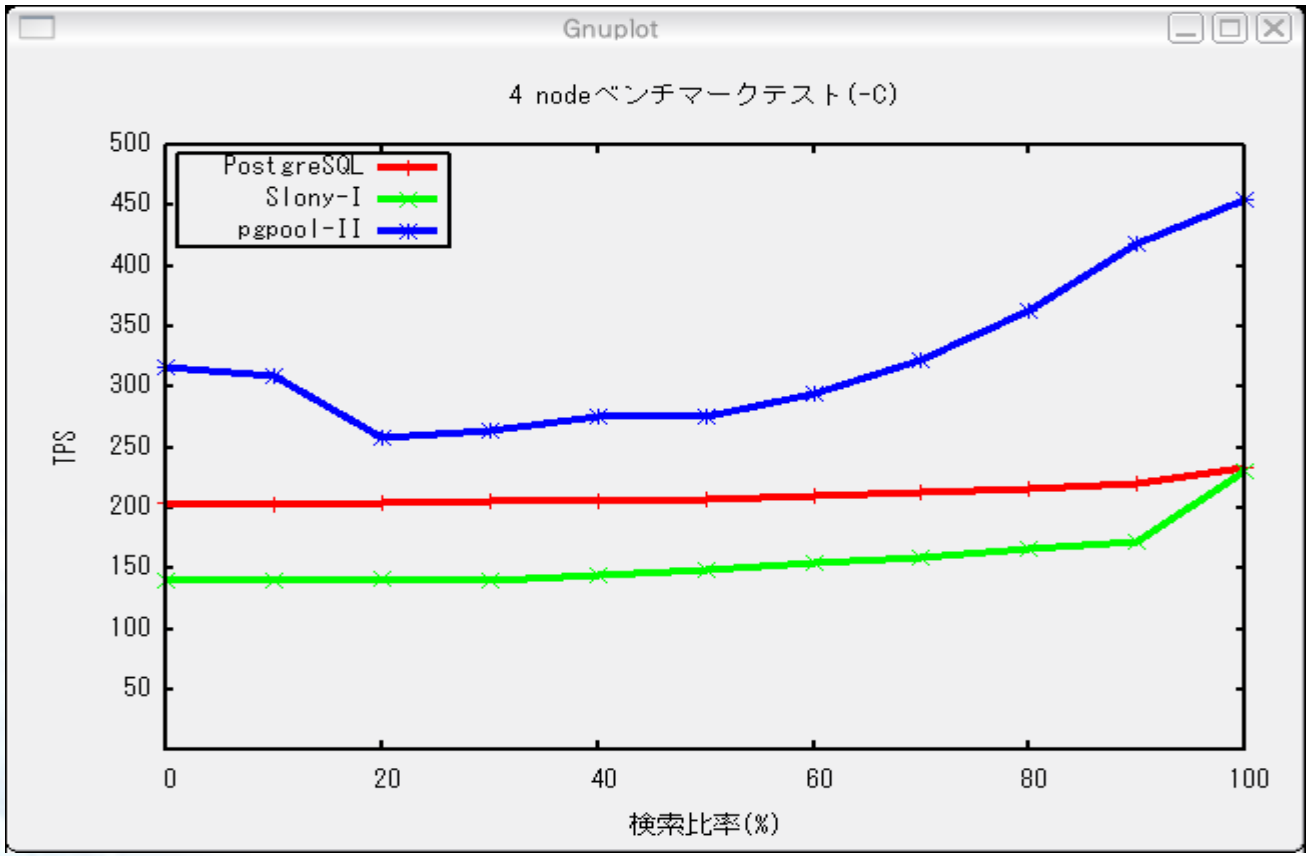
```
¥set naccounts 100000 * :scale
¥setrandom aid 1 :naccounts
SELECT abalance FROM accounts WHERE aid = :aid;
¥setrandom delta -5000 5000
¥setrandom aid 1 :naccounts
UPDATE accounts SET abalance = abalance + :delta WHERE aid = :aid;
```

pgbench によるベンチマーク結果



検索比率が60%越えると
pgpool-II が速い。

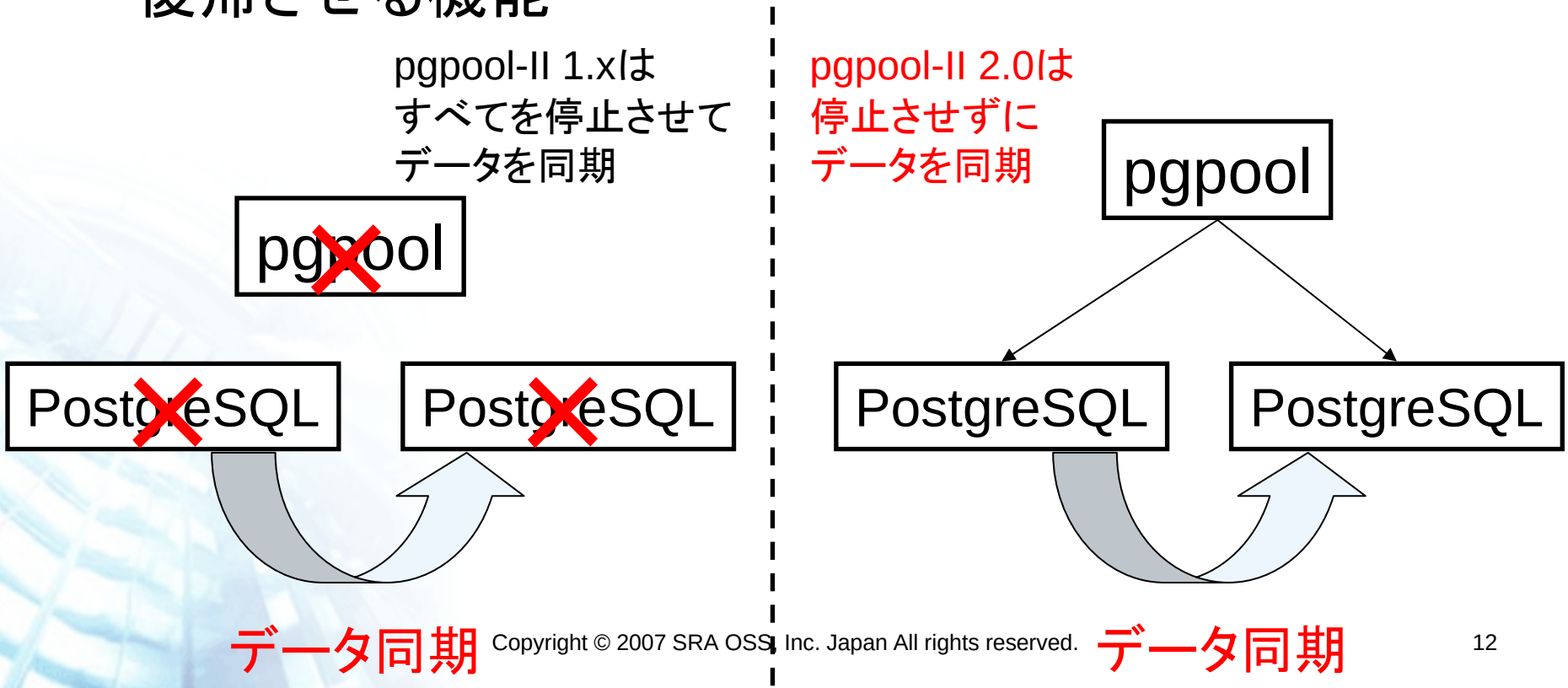
pgbench(-C オプション付き)のベンチマーク結果



コネクションプーリングの効果もあり、
常にpgpool-IIが速い。

オンラインリカバリとは？

- pgpoolを停止させずにデータを同期させてノードを復帰させる機能



オンラインリカバリの仕組み

- 2段階に分けてデータを同期
  - 1段階を**ファーストステージ**と呼ぶ
    - 並行して DB の参照・更新が**可能**
  - 2段階を**セカンドステージ**と呼ぶ
    - すべての接続が終了するまで待機
    - 接続リクエストをすべてブロック
  - 最後にPostgreSQLを起動
    - PostgreSQLが接続を受け付ける状態になった時点でブロックを解除
- リカバリ方法
  - ユーザ自身でどのようにリカバリするかを決めることができる
  - pgpool-IIのソースコードにサンプルスクリプトを同梱
  - リカバリスクリプトはマスタのデータベースクラスタに配置し、C言語関数内からpostgresプロセスがコマンドを実行

オンラインリカバリの準備

- リカバリのためのC言語関数をインストール
  - pgpool-IIから指定したコマンドをSQL経由で実行するためのC言語関数
    - template1データベースにインストール
  - 権限の設定に注意！

```
% cd pgpool-II-2.0/sql/pgpool-recovery/
% make install
% psql –h master-node –f pgpool-recovery.sql template1
```

オンラインリカバリ設定

- pcp.conf
  - pcp_recovery_node コマンドを使うので設定は必須
- pgpool.conf
  - recovery_user
    - リカバリ中にPostgreSQL(template1データベース)へ接続するためのユーザ名
  - recovery_password
    - recovery_userのパスワード
  - recovery_1st_stage_command
    - ファーストステージで実行するコマンド
    - コマンドは$PGDATA以下におく必要がある(セキュリティ上の問題)
  - recovery_2nd_stage_command
    - セカンドステージで実行するコマンド
    - コマンドは$PGDATA以下におく必要がある(セキュリティ上の問題)
  - backend_data_directoryN(Nは数字)
    - データベースクラスタディレクトリ名

オンラインリカバリ実行方法

- pcp_recovery_node コマンド
- 管理ツール上からも実施可能

オンラインリカバリ内部処理

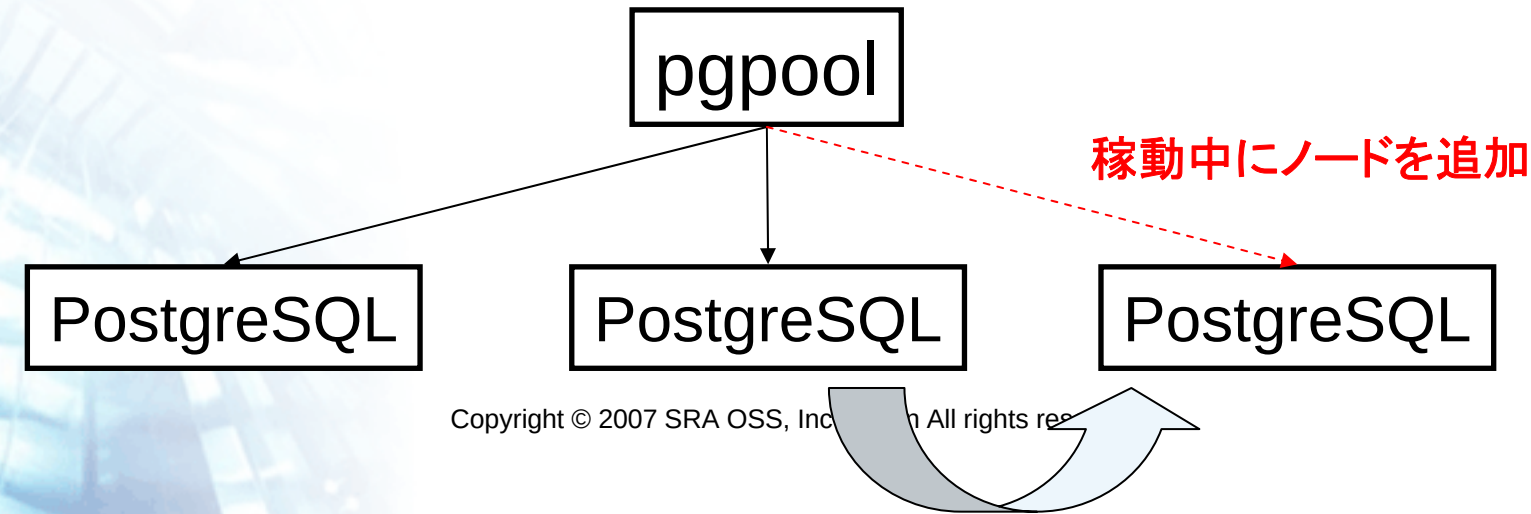
1. バッファ上のデータをディスクへ同期
2. recovery_1st_stage_commandの実行
3. バッファ上のデータをディスクへ同期
   - 同期する前に接続中のクライアントがいなくなるまで待機
   - 接続はブロックされる(クライアント側からは接続に時間がかかるように見える)
4. recovery_2nd_stage_commandの実行
5. postmasterの起動(pgpool_remote_startの実行)
   - 起動スクリプトは$PGDATA/pgpool_remote_startというファイル名
6. リカバリしたPostgreSQLへ接続できるか確認
7. 接続ブロックの解除

オンラインリカバリの実行例

- PITRを使ったオンラインリカバリ
  - 差分バックアップにより、接続をブロックする時間を短時間にすることが可能
  - ブロックする時間はpostmasterがアーカイブログからリカバリする時間
- ファーストステージ
  - ベースバックアップを取得し、ダウンしたホストへコピー
- セカンドステージ
  - SELECT pg_switch_xlog() を実行し、最新のWALログをアーカイブさせる

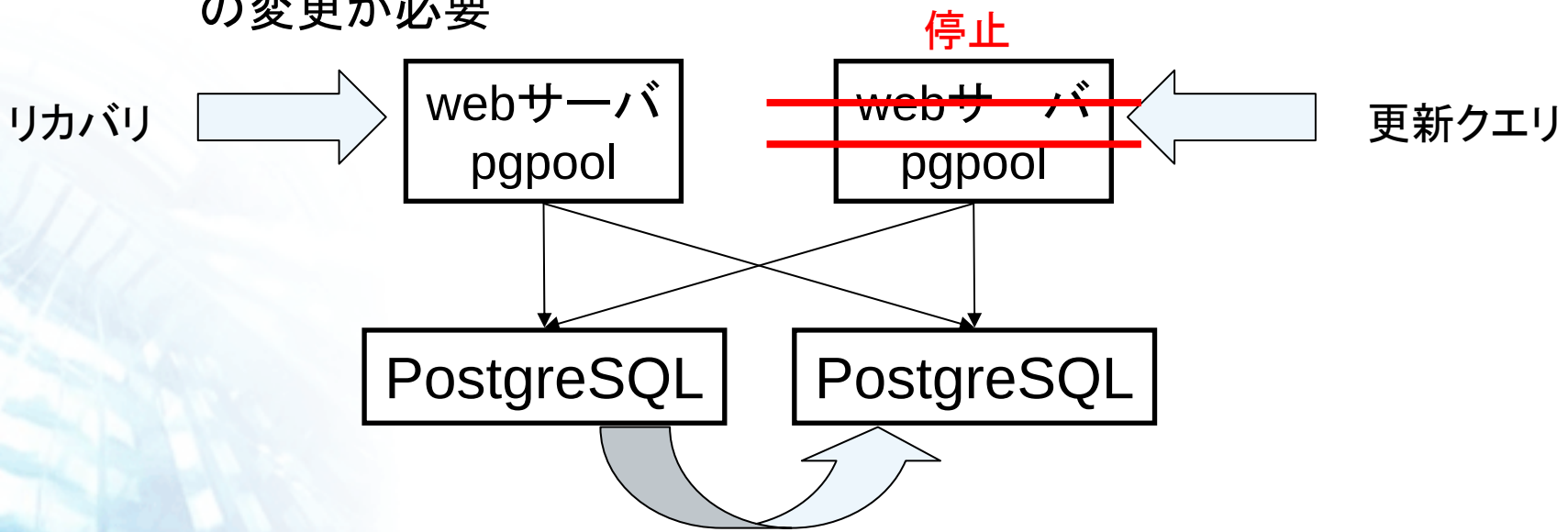
オンラインリカバリの応用

- ノードの動的追加
  - 2.0 から pgpool.conf のリロードが可能
  - backend_* を追加し、オンラインカバリすることで負荷状況に応じてノードを追加ができる(**スケールアップ**)
    - リロードするとノードは「ダウン」状態で追加される



オンラインリカバリ注意事項

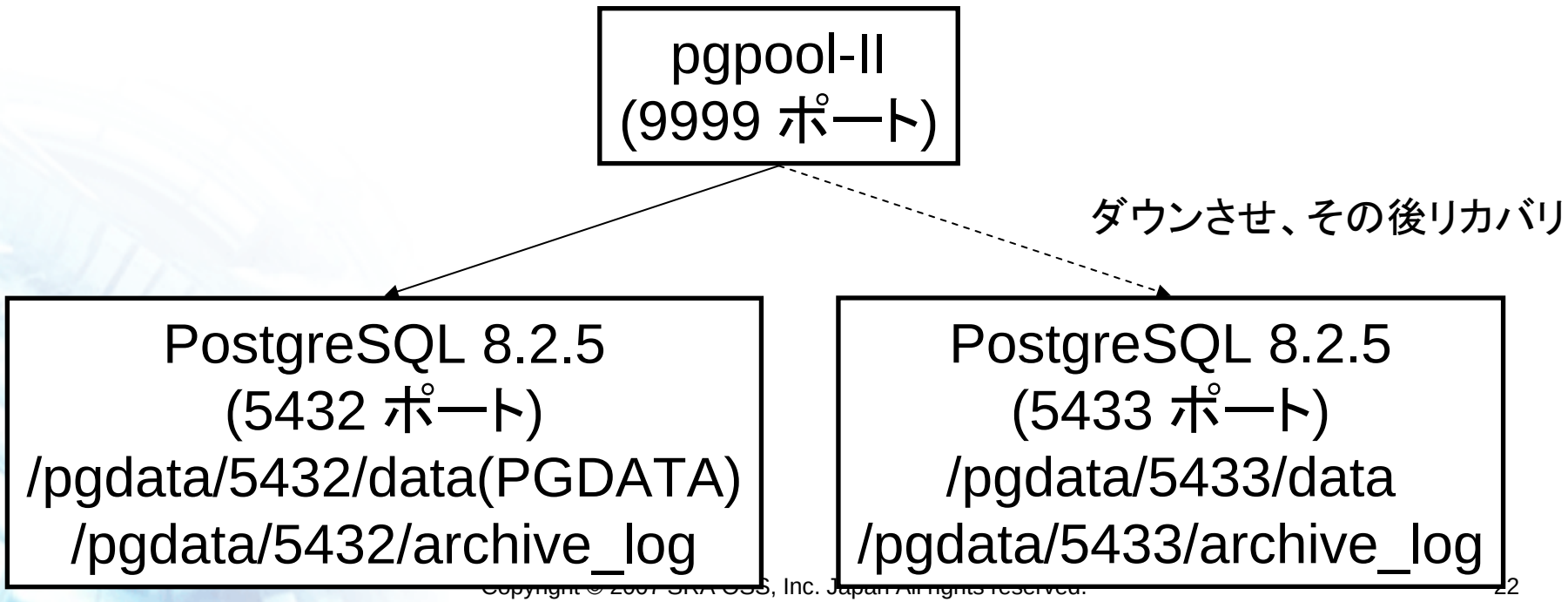
- 一時的にpgpoolを1台での運用
  - リカバリ中に一時的に更新系クエリを止めるが、別ホストで動作しているpgpoolを止めることはできない
    - 例えば、負荷分散装置を一時的に1つのwebサーバへ送るなどの変更が必要



デモ

オンラインリカバリデモ(準備)

- 環境
  - ローカルにpgpoolとPostgreSQLを配置



オンラインリカバリ手順

1. pgpool の設定・起動
2. ファーストステージスクリプト作成
3. セカンドステージスクリプト作成
4. postmaster起動スクリプト作成
5. スクリプトの配置
6. リカバリの実行
7. データの確認

オンラインリカバリデモ(pgpool設定)

- pgpool.conf

```
replication_mode = true
recovery_user = ‘y-asaba’
recovery_1st_stage_command = 'base-backup.sh‘
recovery_2nd_stage_command = 'pgpool_recovery‘
backend_hostname0 = ''
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/pgdata/5432/data'
backend_hostname1 = ''
backend_port1 = 5433
backend_weight1 = 1
backend_data_directory1 = '/pgdata/5433/data'
```

オンラインリカバリ(スクリプト)

- ファーストステージ用のスクリプト
  - スクリプト内で3つの引数を受けることが可能
    1. マスタのデータベースクラスタディレクトリ名
    2. リカバリターゲットのホスト名
    3. リカバリターゲットのデータベースクラスタディレクトリ名
  - ベースバックアップを取得
    1. pg_start_backup()の実行
    2. rsync によるマスタのベースバックアップのコピー
    3. pg_stop_backup()の実行

オンラインリカバリ(スクリプト)

```
#! /bin/sh
DATA=/pgdata
MASTER_DATA=$1       # マスタの PGDATA
RECOVERY_HOST=$2 # リカバリ先のホスト名
RECOVERY_DATA=$3 # リカバリ先の PGDATA

psql -c "select pg_start_backup('pgpool-recovery')" postgres

# recovery.conf の生成
echo "restore_command = 'cp $DATA/5432/archive_log/%f %p'" >
$MASTER_DATA/recovery.conf
# データの退避とベースバックアップのコピー
rm -rf $RECOVERY_DATA.bk
mv -f $RECOVERY_DATA{,.bk}
rsync –az $DATA/5432/data/ $RECOVERY_DATA/
cp -f $RECOVERY_DATA.bk/postgresql.conf $RECOVERY_DATA
rm -f $RECOVERY_DATA/postmaster.pid

psql -c 'select pg_stop_backup()' postgres
```

オンラインリカバリ(スクリプト)

- セカンドステージ用のスクリプト
  - xlogの切り替え

```
#! /bin/sh

psql -c 'select pg_switch_xlog()' postgres
```

オンラインリカバリ(スクリプト)

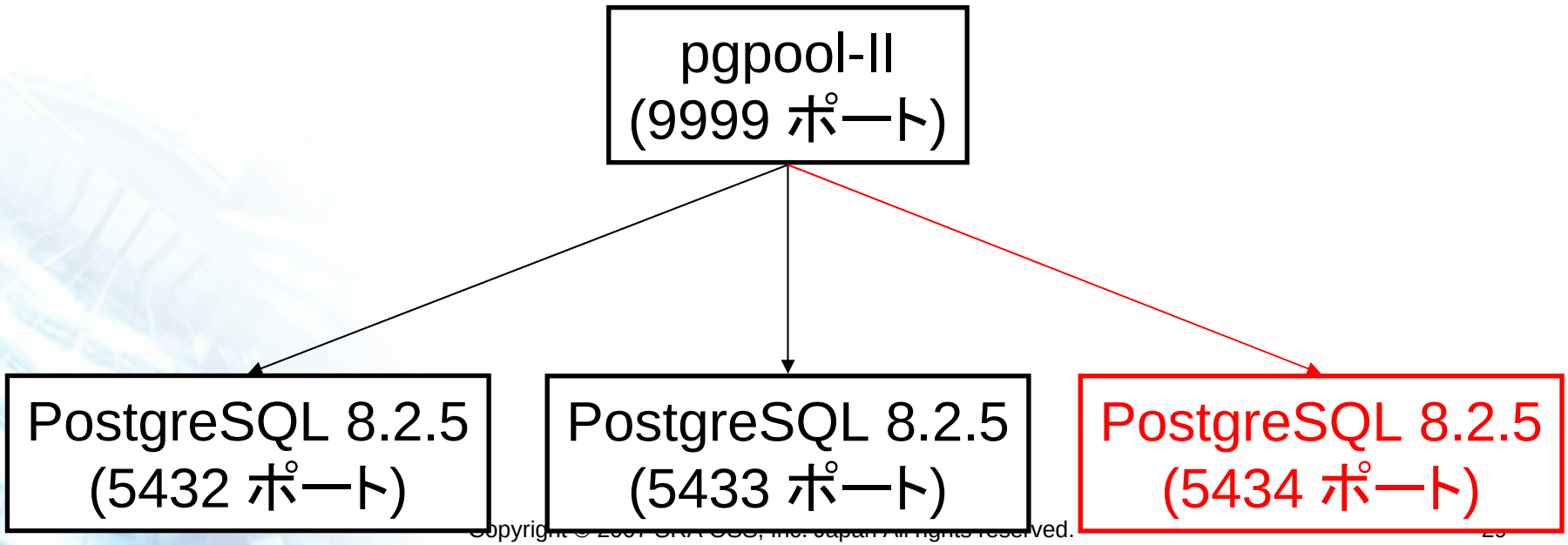
- postmaster 起動スクリプト
  - 1 引数目 : リカバリノードのホスト名
  - 2 引数目 : リカバリノードのデータベースクラスタパス

```
#! /bin/sh
DEST=$1
DESTDIR=$2

pg_ctl -w -D $DESTDIR start
```

ノードの動的追加のデモ

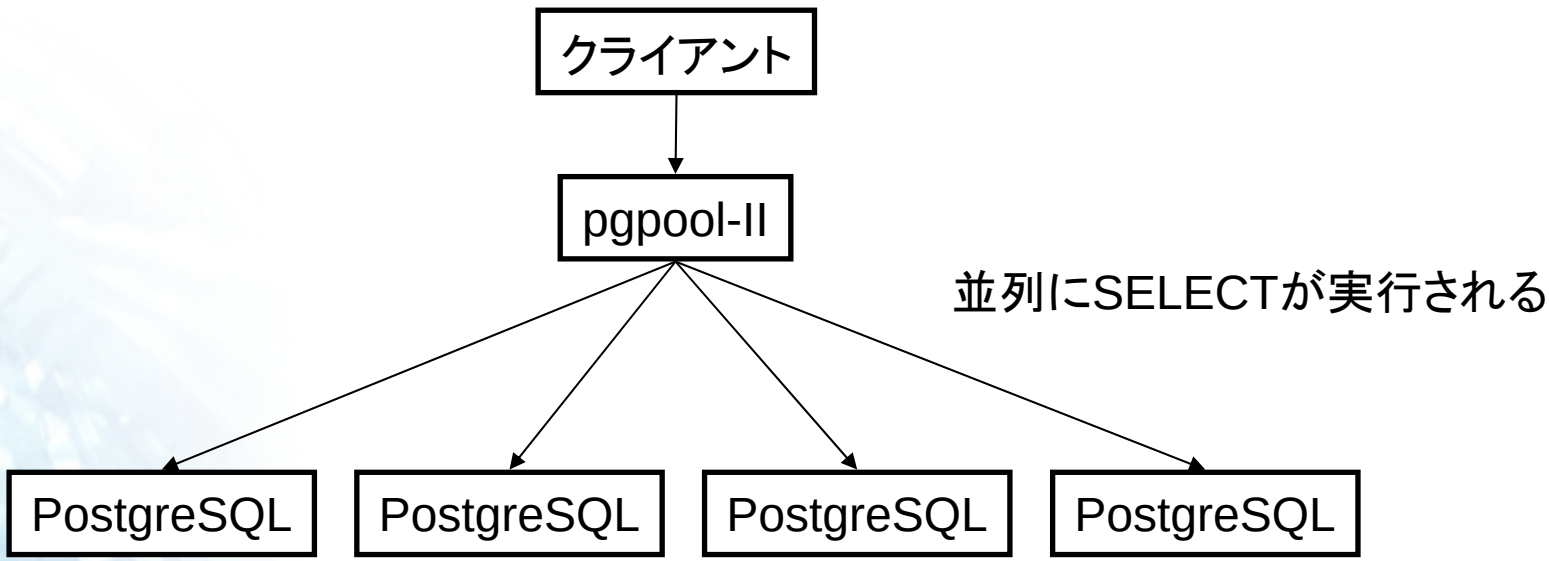
- 新しく5434ポートで動くPostgreSQLを追加
  - 設定ファイルの編集、リロード



パラレルクエリ

パラレルクエリとは？

- 複数のノードでリクエスト(SELECT)を処理
  - 1つのテーブルを複数サーバに分割(テーブルパーティショニング)しておき、SELECTの結果をpgpool-IIでまとめることで、仮想的に1つのテーブルとして扱う

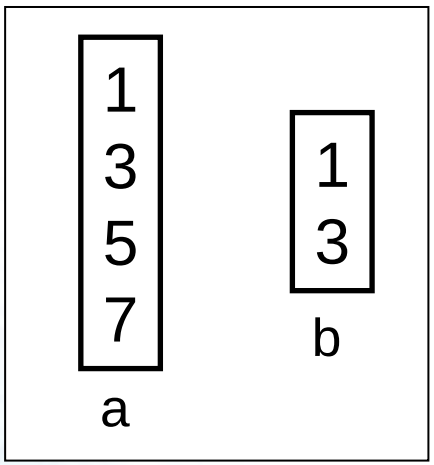


パラレルクエリの変更点

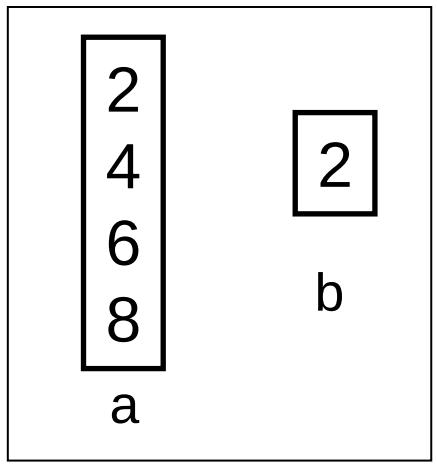
- 1.xの挙動
  - すべてのテーブルを分割する必要がある
- 2.0
  - レプリケーションテーブルとパーティショニングテーブルを混在させることが可能
    - クエリをより並列に動かすことが可能

パラレルクエリの弱点(1)

- パラレルクエリの弱点
  - ノードをまたぐテーブル結合



ノード1



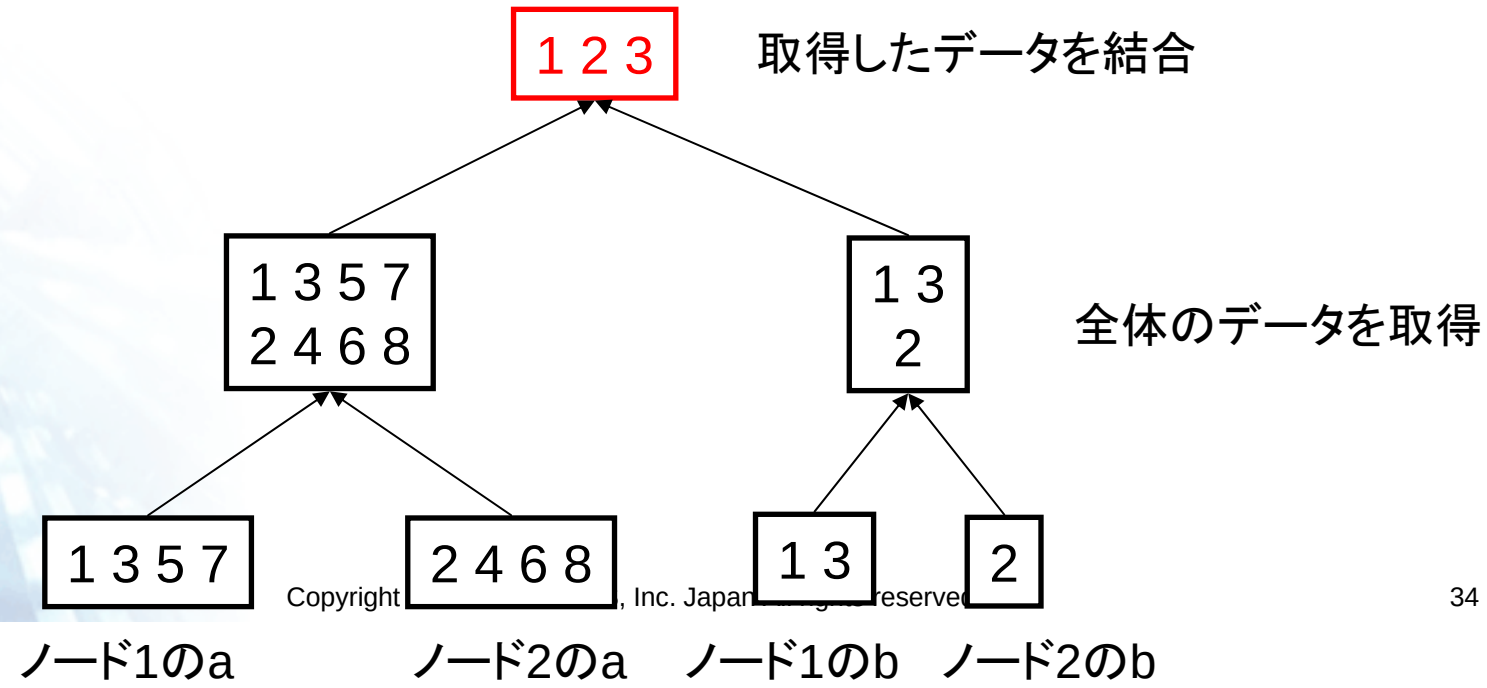
ノード2

a,bそれぞれを
偶数、奇数で
振り分け

SELECT * FROM a,b WHERE a.i = b.i

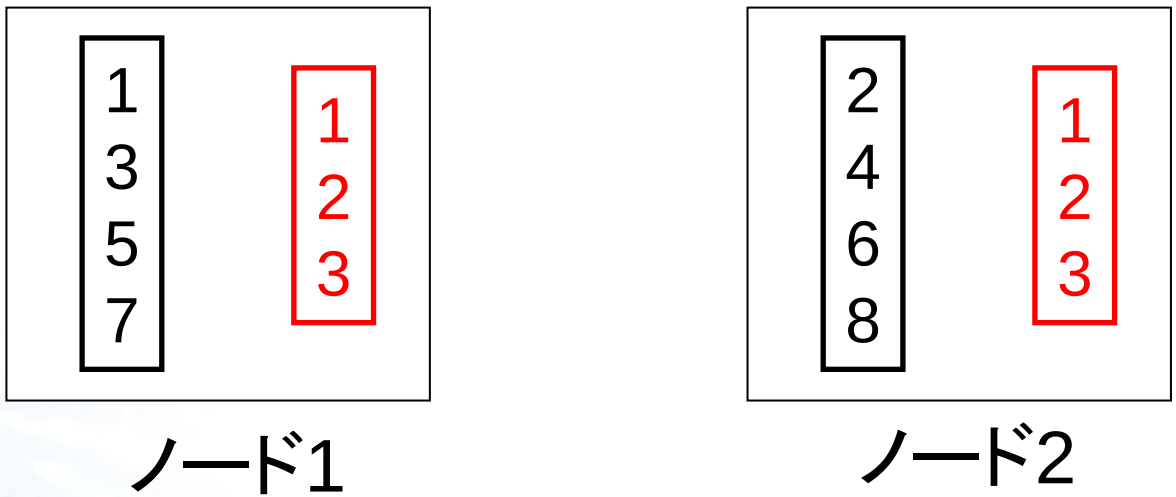
パラレルクエリの弱点(2)

- ノードをまたぐテーブル結合の流れ
  I. テーブルaのデータをノード1、2から取得
  II. テーブルbのデータをノード1、2から取得
  III. SystemDB上でI, IIで取得したデータを結合



パラレルクエリの改良(1)

- 部分レプリケーションテーブルのサポート

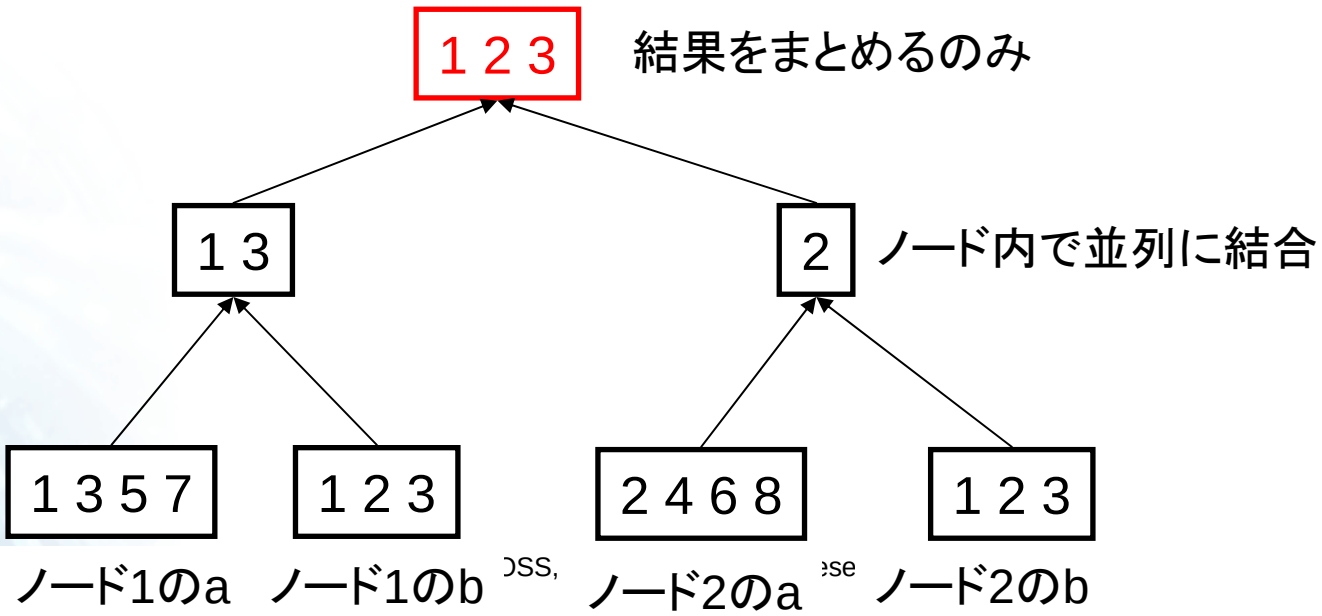


SELECT * FROM a,b WHERE a.i = b.i

部分レプリケーションによって、
ノード内でJOINが完結することが可能

パラレルクエリの改良(2)

- 部分レプリケーションテーブルの流れ
  I. ノード1でaとbをテーブル結合
  II. ノード2でaとbをテーブル結合
  III. I,IIの結果をまとめてクライアントに返却



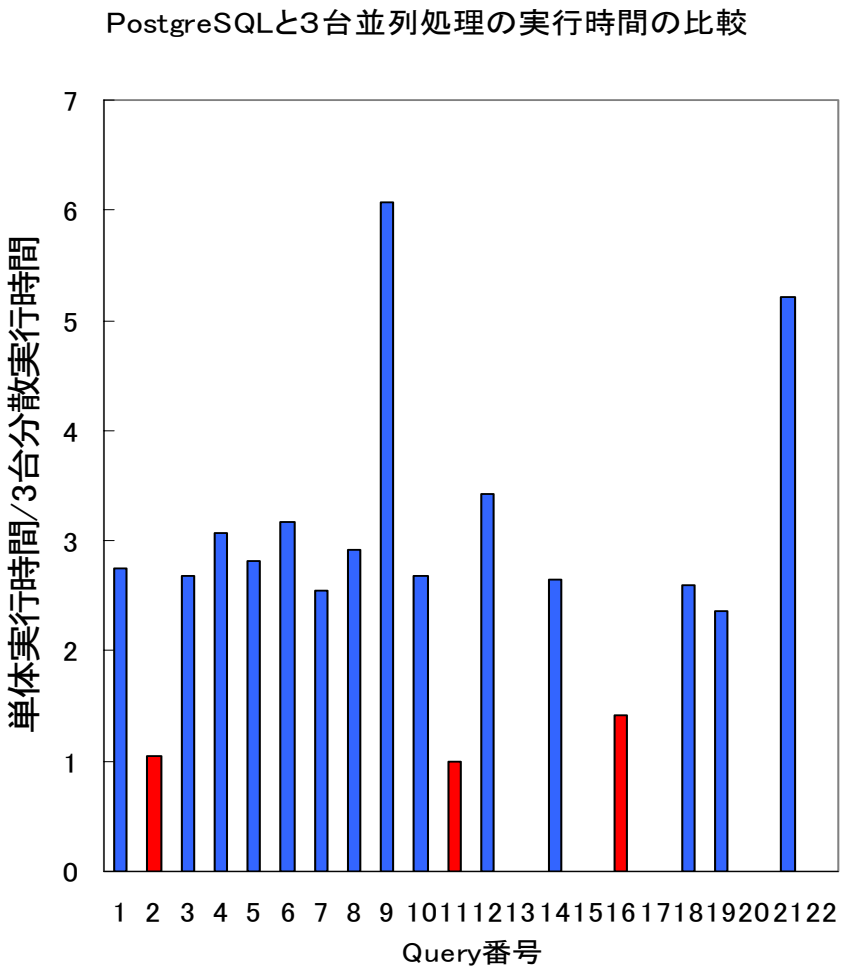
部分レプリケーションのルール登録

- replicate_def テーブルの追加
  - 分散ルールとほぼ同じスキーマ
  - 分散キーとルール定義関数名の登録は必要なし

```
INSERT INTO pgpool_catalog.replicate_def VALUES (
   'pgpool',
   'public',
   'branches',
  ARRAY['bid','bbalance','filler'],
  ARRAY['integer','integer','character(84)'] );
```

pgbenchのbranchesテーブル定義例

DBT-3によるベンチマーク



- line_itemとordersをテーブル分割
  - PostgreSQL 3 ノード(3分割)
  - 青線：パーティションテーブルとレプリケーションテーブルの結合
  - 赤線：レプリケーションテーブル同士の結合
  - パラレルクエリの弱いクエリは未測定(13,15,17,20,22)
- scale factor＝10
  - インデックス含めてトータルで30GB

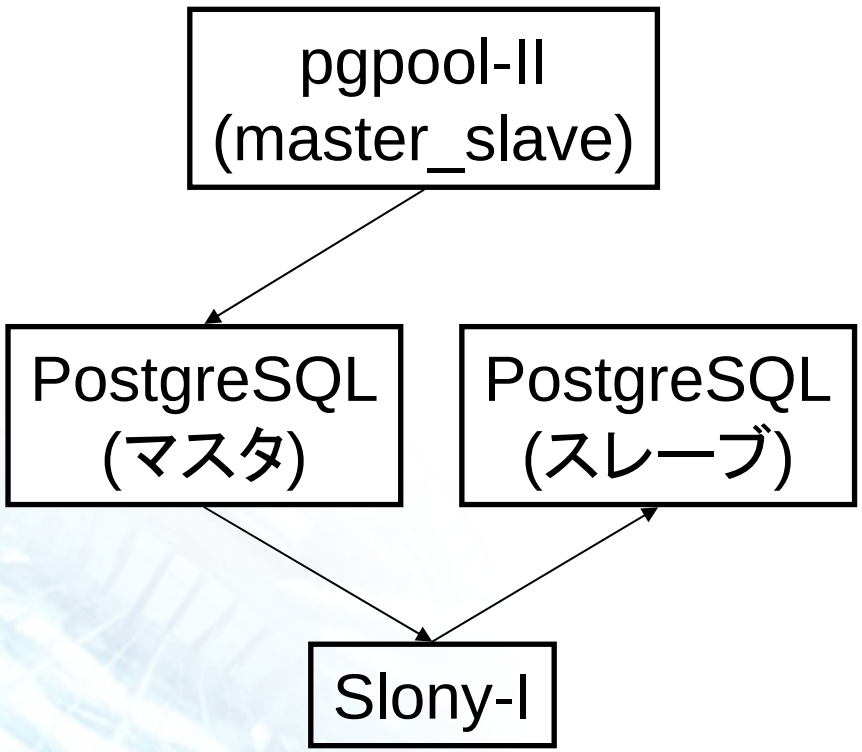
最大で6倍のパフォーマンス改善(クエリ9番)

全般

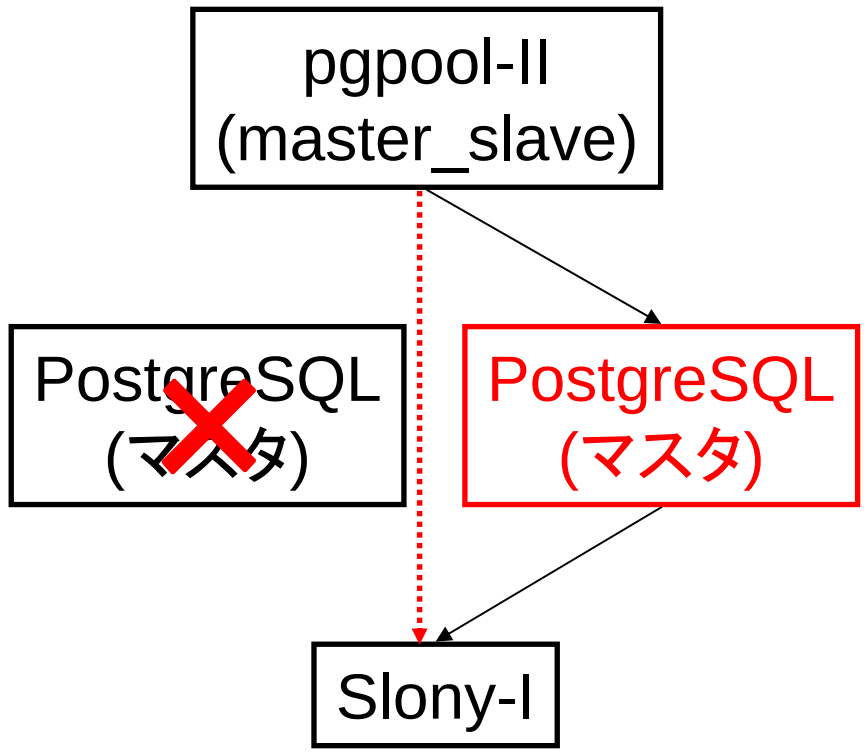
他のシステムとの連携

- failover_command
  - ノードを切り離した時に指定したコマンドを実行
  - Slony-I の連携やアラートメールを送ることが可能
- failback_command
  - ノードを戻した時に指定したコマンドを実行
- pgpool-IIから必要な情報を渡すことも可能(特殊文字を置換)
  - %d: 対象ノード番号
  - %h: 対象ノードのホスト名
  - %p: 対象ノードのポート番号
  - %D: 対象ノードのデータベースクラスタパス

Slony-Iとの連携例



通常時



failover_commandでSlony-Iに対して
マスタを切り替えるように指示

クライアントの強制切断

- client_idle_limit
  - クライアントからクエリが届くまでの最大待ち時間を指定可能
  - 秒単位で指定
  - TCPのタイムアウト待ち防止などに有効

参考URL

- pgpool-II 開発サイト
  - http://pgfoundry.org/projects/pgpool/
- pgpool Wiki
  - http://pgpool.sraoss.jp/
- pgpoolメーリングリスト
  - http://www.sraoss.jp/mailman/listinfo/pgpool-general-jp
- マイコミジャーナルでのpgpool-IIの解説記事
  - http://journal.mycom.co.jp/articles/2007/11/08/pgpool/index.html

ご清聴ありがとうございました。