SRA OSS,INC.

PostgreSQL 8.2の概要とpgpool-IIについて

SRA OSS Inc.日本支社
石井達夫

自己紹介

- 石井 達夫
- SRA OSS, Inc.日本支社
  - 2005年7月設立
  - 本社 米国 サンタクララ
  - 資本金 100万ドル
  - 事業内容
    - オープンソースソフトウェアの
      - サポート
      - 教育
      - コンサルティング
      - パッケージ開発，販売

SRA OSSの扱っているOSS

- PostgreSQL
- Linux
- Sylpheed
- そのほか
  - tomcat, sendmail, postfix etc...

アジェンダ

- PostgreSQL 8.2の概要
- pgpool-IIについて

PostgreSQL 8.2

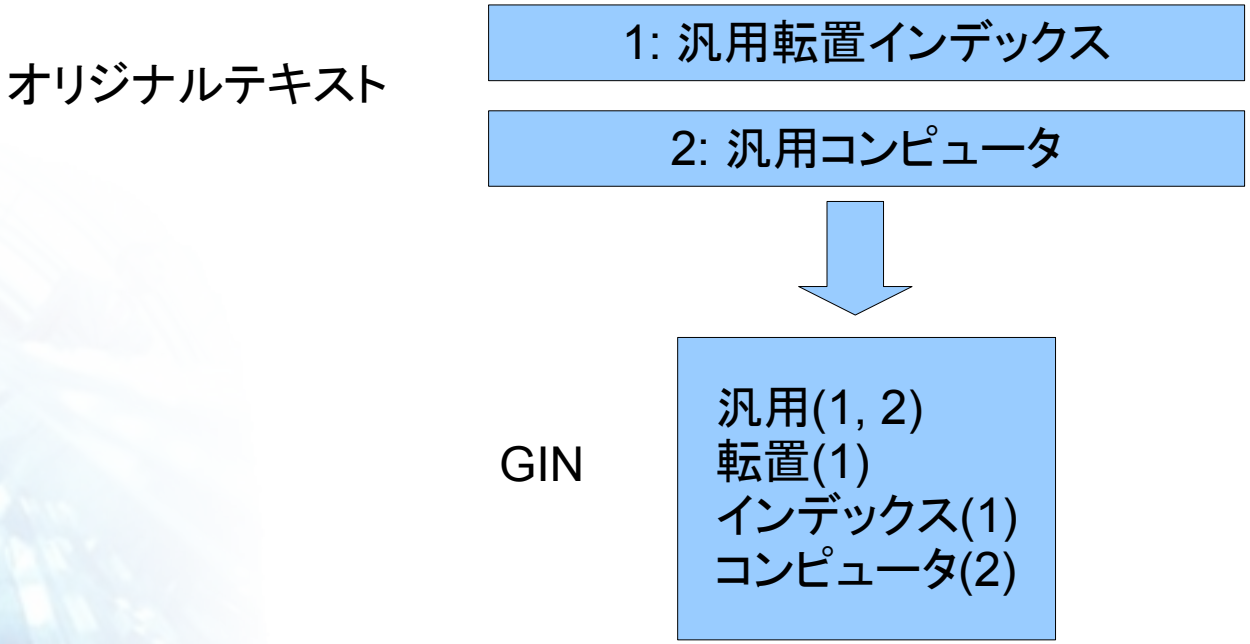
- 2006/12/5リリース？
- 2005/11リリースの8.1からちょうど1年目
- 過去のリリースを振り返ると...
  - 8.0: 大きな機能追加(PITRなど)
  - 8.1: 大きな性能向上(バッファ管理の改良)
  - 8.2: そこそこの機能追加とそこそこの性能向上

機能追加

- GIN
- INSERT/UPDATE/DELETEで更新行情報の返却
- INSERTで複数行の挿入
- 行コンストラクタ
- COPY TOの拡張
- CEの改良
- FILLFACTOR
- contrib
- ウォームスタンバイ

GIN

- Generalized Inverted Index
- 日本語では「汎用転置インデックス」



GINの特徴

- インデックスに対する演算子を自由に定義できる
  - Btreeでは，<, <=, = , >, >= のみ
- データ型に依存しない
  - ユーザ定義データ型にも適用可能
  - デフォルトで，整数，テキスト用のインデックスを作成可能
- Teodor Sigaev氏やOleg Bartunov氏が開発
  - ロシア人，GiST，Tsearch2の開発で有名

使用例(整数型)

```
CREATE TABLE t1(i INTEGER[]);
INSERT INTO t1 VALUES(ARRAY[1,2,3]);
INSERT INTO t1 VALUES(ARRAY[4,5,6]);
INSERT INTO t1 VALUES(ARRAY[7,8,9]);

test=# SELECT * FROM t1 WHERE 5 =ANY(i);
    i
---------
 {4,5,6}
(1 row)

CREATE INDEX t1index ON t1 USING GIN(i);

test=# SELECT * FROM t1 WHERE ARRAY[5] <@  i;
    i
---------
 {4,5,6}
(1 row)
```

使用例(整数型, cont.)

- 演算子
  - <@
    - nを含む行
  - &&
    - 配列要素が重なっている
  - @>
    - 行がnに含まれる

全文検索への応用

- 基本的には整数型のときと同じで，TEXT配列にGINインデックスを作る
- 日本語は分かち書きにしてからstring_to_arrayで配列に変換する
- シングルクォートとバックスラッシュに注意
  - エスケープが必要

わかち書きにする方法

- kakasi, chasen, MeCabを使い，テキストを加工してからSQL文に変換する
- それだと面倒&遅いので...
  - デモではMeCabを呼び出すC関数を作成して利用
  - 実行例

```
test=# SELECT wakachi('日本語のテキスト');
        wakachi
----------------------
 日本語 の テキスト
(1 row)
```

データの登録の実際

```
psql -c "DROP TABLE gin;" $DB
psql -c "CREATE TABLE gin (fname TEXT, orig TEXT, wakachi TEXT[]);" $DB
psql -c "CREATE INDEX ginindex ON gin USING GIN(wakachi);" $DB
cd testdata
ls [0-9]*|sort -n|while read i
do
        echo $i
        sed -e s@\'@\'\'@g -e s@\\\\@\\\\\\\\@g $i > $tmp
        psql -c "INSERT INTO gin VALUES ('$i',`cat $tmp`',
      string_to_array(wakachi(`cat $tmp`), ' '))" $DB
done
```

GINによる検索の例

```
test=# SELECT fname FROM gin WHERE ARRAY['配列'] <@ wakachi;
 fname
--------
 05.txt
 08.txt
(2 rows)

test=# EXPLAIN SELECT fname FROM gin WHERE ARRAY['配列'] <@ wakachi;
                          QUERY PLAN
---------------------------------------------------------------
 Index Scan using ginindex on gin  (cost=0.00..8.02 rows=1 width=32)
   Index Cond: ('{配列}'::text[] <@ wakachi)
(2 rows)
```

フレーズ検索の例

```
test=EXPLAIN SELECT fname FROM gin WHERE ARRAY['配列', '型'] <@ wakachi AND
orig ~ '配列型';
                          QUERY PLAN
---------------------------------------------------------------
 Index Scan using ginindex on gin  (cost=0.00..8.02 rows=1 width=32)
   Index Cond: ('{配列,型}'::text[] <@ wakachi)
   Filter: (orig ~ '配列型'::text)
(3 rows)
```

GINの制限事項

- 「検索語の一部を含む」という条件での検索ができない
  - 例:「日本」で「日本人」も「日本語」も引っかけたい
  - 開発者によれば，今後改善の予定

GINの実例

- pgsql-jpの35485から37636までの3071件のメール
- 合計21MB(du)，合計13.4MB，平均4572バイト
- テーブルサイズ：2.4MB
  - ただし，オリジナル＋わかち書きテキスト
  - EUC_JPに変換かつ圧縮がかかっているのでかなり効率的
- GINインデックスサイズ：20MB

検索速度

- CPU: Pentium M 1.1 GHz, Mem: 768MB
- 「石井」を含む行を検索: 1.5ms(271件)
  - SELECT count(*) FROM gin WHERE ARRAY['石井'] <@ wakachi;(namazu: 14ms)
- 「石井」と「達夫」を含む行を検索: 1.1ms(34件)
  - SELECT count(*) FROM gin WHERE ARRAY['石井','達夫'] <@ wakachi;(namazu: 14ms)
- OR検索: 1.7ms(271件)(namazu: 15ms)
  - SELECT count(*) FROM gin WHERE ARRAY['石井'] <@ wakachi OR ARRAY['達夫'] <@ wakachi;

検索速度(cont.)

- フレーズ検索: 7ms(30件)(namazu: 14ms)
  - SELECT count(*) FROM gin WHERE ARRAY['石井','達夫'] <@ wakachi AND orig ~ '石井達夫';
- フレーズ検索(インデックスなし): 650ms(30件)
  - SELECT count(*) FROM gin WHERE orig ~ '石井達夫';

GINまとめ(namazuと比較しつつ)

- PostgreSQLで日本語が通る全文検索システムが作れる
- インデックスサイズもまあまあ許せる
  - オリジナル: 21MB, GIN: 24MB, namazu: 27MB
- すべてがRDBMS内にあるので，データの整合性維持，管理が容易．部分更新も容易
- 高速検索．namazuよりも1桁速い
- インデックス作成は遅い
- RDBMSの技術が使える
  - パーティショニングやパラレル検索

INSERT/UPDATE/DELETEで更新行情報の返却

- RETURNINGオプション

```
test=# INSERT INTO t1 VALUES(1, 'aaa') RETURNING *;
 i |  j
---+-----
 1 | aaa
(1 row)

test=# UPDATE t1 SET j = 'bbb' WHERE i = 1 RETURNING *;
 i |  j
---+-----
 1 | bbb
(1 row)

test=# DELETE FROM t1 WHERE i = 1 RETURNING *;
 i |  j
---+-----
 1 | bbb
(1 row)
```

INSERTで複数行の挿入

```
test=# INSERT INTO t3 VALUES (1),(2),(3);
INSERT 0 3
test=# SELECT * FROM t3;
 i
---
 1
 2
 3
(3 rows)
```

行コンストラクタ

- SELECT row(a, b) > row(c, d);
- 8.1以前では...
  - a > c かつ b > d のときにtrue
- 8.2(とSQL標準では)
  - a > c か, a = c かつ b> d
- 8.1と8.2で結果が違う例
  - SELECT row(2,5) > row(1,10);

COPY TOの拡張

- SELECT結果のCOPY
  - COPY (SELECT * FROM t3 WHERE i IN (1, 3)) TO '/tmp/test.dat';
- テーブルがなくてもCOPY可能
  - COPY (SELECT 1) TO '/tmp/test.dat';

CE(Constraint Exclusion)の改良

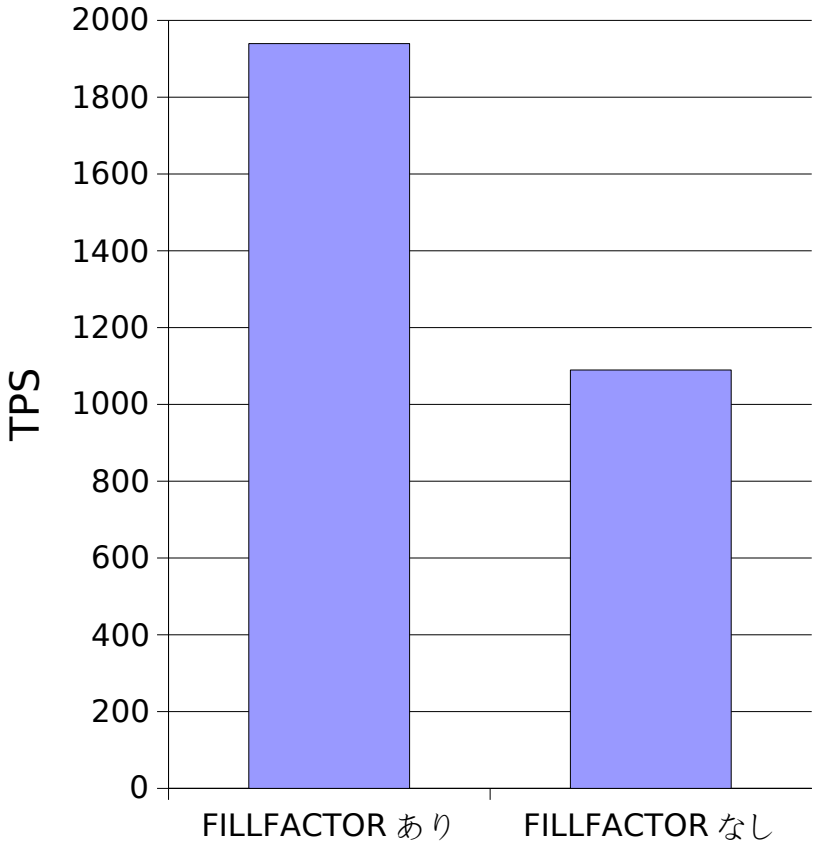
- CEとは
  - 継承を利用し，テーブルの水平パーティショニングを行う機能
- 8.1では，SELECTのみが対応
  - 例: sales01, sales02, sales03にパーティショニングしたときに，SELECTはsales01のみを検索できる
  - UPDATE/DELETEはsales01,sales02, sales03もアクセスしてしまう
- 8.2ではSELECT同様，sales01のみをアクセスすることによって性能向上

FILLFACTOR

- 8.1までは...
  - テーブルやインデックスになるべく隙間ができないようにデータを追加
- FILLFACTOR
  - CREATE TABLE t1(i INTEGER) WITH (FILLFACTOR = 70);で，INSERT，COPY時に30%隙間を残す
  - 更新が多いテーブルではFILLFACTORを小さめにすることによって，後々の更新でディスクアクセスを減らすことができる

FILLFACTOR(cont.)

- FILLFACTORの効果
  - pgbenchを改造してFILLFACTOR=70でaccountsテーブル(10万件)を作成
  - pgbenchで同時30ユーザ，それぞれ1000トランザクション，計3万回 accounts テーブルを更新する



contribモジュール

- pgbench
  - スクリプトファイルで四則演算が可能
  - 変数”scale”の導入
    - \set ntellers 10 * :scale
  - -D name=valueで変数をコマンド引数で定義可能

contribモジュール(cont.)

- pgrowlocksで行ロック情報の取得
  -

```
test=# SELECT * FROM pgrowlocks('t1');
 locked_row | lock_type | locker | multi |   xids    |     pids
------------+-----------+--------+-------+-----------
+---------------
    (0,1) | Shared    |     19 | t     | {804,805} | {29066,29068}
    (0,2) | Shared    |     19 | t     | {804,805} | {29066,29068}
    (0,3) | Exclusive |    804 | f     | {804}     | {29066}
    (0,4) | Exclusive |    804 | f     | {804}     | {29066}
(4 rows)
```

contribモジュール(cont.)

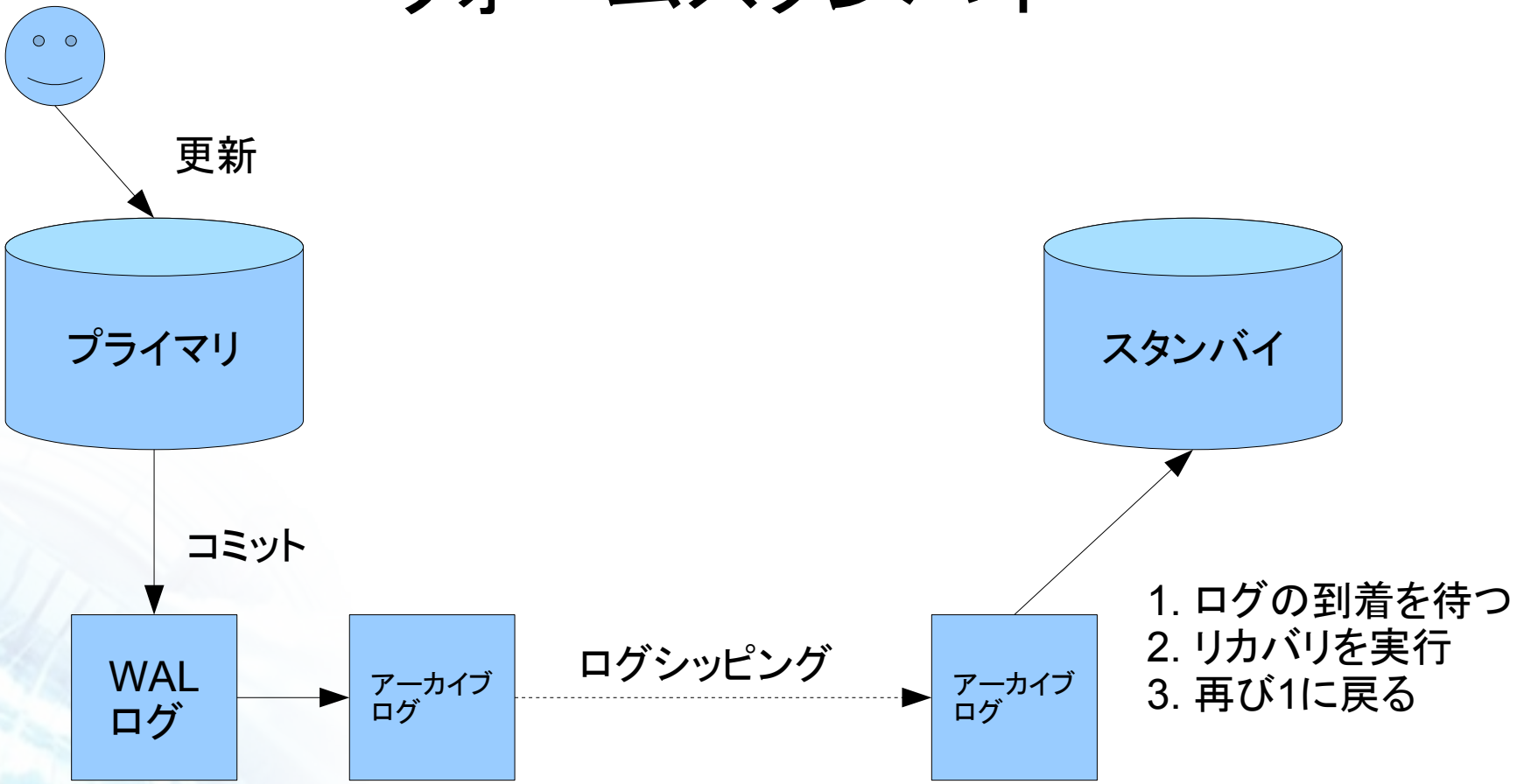
- pg_freespacemap

```
 relname    | pg_relation_size | free_bytes | free_percent
--------------+------------------+------------+--------------
 accounts     |        134299648 |       4604 |         0.00
 pg_depend    |           262144 |       1768 |         0.67
 pg_attribute |           237568 |       5560 |         2.34
 pg_statistic |           139264 |       8600 |         6.18
 pg_class     |            49152 |       6592 |        13.41
 pg_type      |            49152 |       7152 |        14.55
 pg_index     |            16384 |       7280 |        44.43
 pg_constraint |            8192 |       7328 |        89.45
 tellers      |             8192 |       3768 |        46.00
 branches     |             8192 |       7768 |        94.82
(10 rows)
```

ウォームスタンバイとは

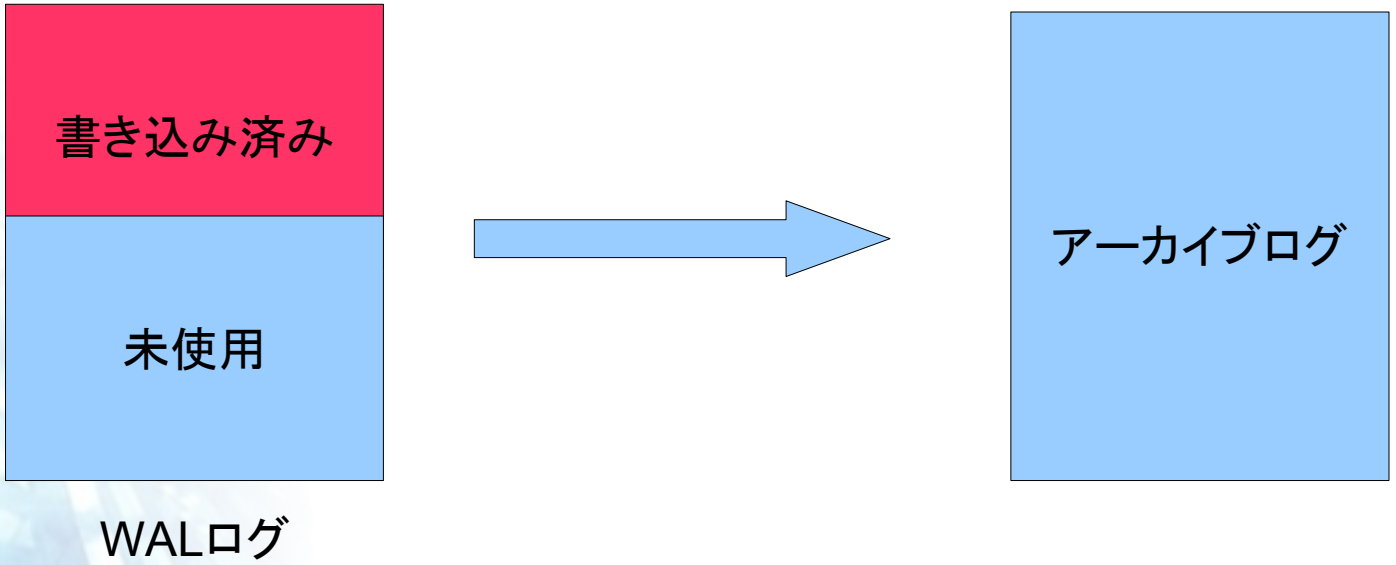
- アーカイブログを別のサーバに転送し，そこから連続的にリカバリをかけることによってデータベースの複製を作る技術
- メリット
  - DB内容の一致が保障される
  - アプリケーションやテーブル定義の変更の必要がない
  - ログを転送さえできればいつでもレプリケーションできる-ディザスタリカバリへの利用
- デメリット
  - スタンバイ側をDBとして利用できない
  - 非同期レプリケーション

ウォームスタンバイ



archive_timeoutの利用

- WALは一杯にならないとアーカイブログにならない
- archive_time秒後に強制的にアーカイブログに



PostgreSQLのクラスタリング レプリケーションソリューション

- クラスタリング/レプリケーションとは
  - 複数のPostgreSQLサーバを使って信頼性や性能を向上させる手法
  - 「クラスタ」と「レプリケーション」は用語の区別が曖昧な場合がある
- 単独のPostgreSQLでは対応できないような高信頼性，高性能が要求される分野でホットな話題
  - 基幹業務
  - 大容量データベース
  - 高負荷

レプリケーションの方式の違い

- 同期(eager)/非同期(lazy)
  - 同期: 複数のDBサーバの内容がリアルタイムに一致する
  - 非同期方式: マスターへの更新が遅れてスレーブに伝播される-複数サーバでデータが一致しない瞬間がある
  - 一般的に同期方式よりも非同期方式の方が性能が高い（はず）

クラスタ構成の違い

- shared nothing方式
  - CPUもディスクも共有しない独立したDBサーバを複数使う方式．もっとも一般的
  - 検索系の性能は期待できる
  - 複数のDBサーバを管理するオーバヘッドがあるので更新系の性能は期待できない
- shared方式
  - CPUやディスクなどの一部のリソースを共有
  - ディスクを共有する方式が多い(Oracle RAC)
  - 排他制御が難しい
  - 更新系の性能も向上する場合がある

並列処理・負荷分散

- 並列処理(parallel query)
  - 複数のDBサーバで問い合わせを並列実行
  - シングルセッションでも性能が向上する
- 負荷分散(load balance)
  - 複数のDBサーバに負荷を分散する
  - 複数セッション時に性能が向上する
  - レプリケーションが前提

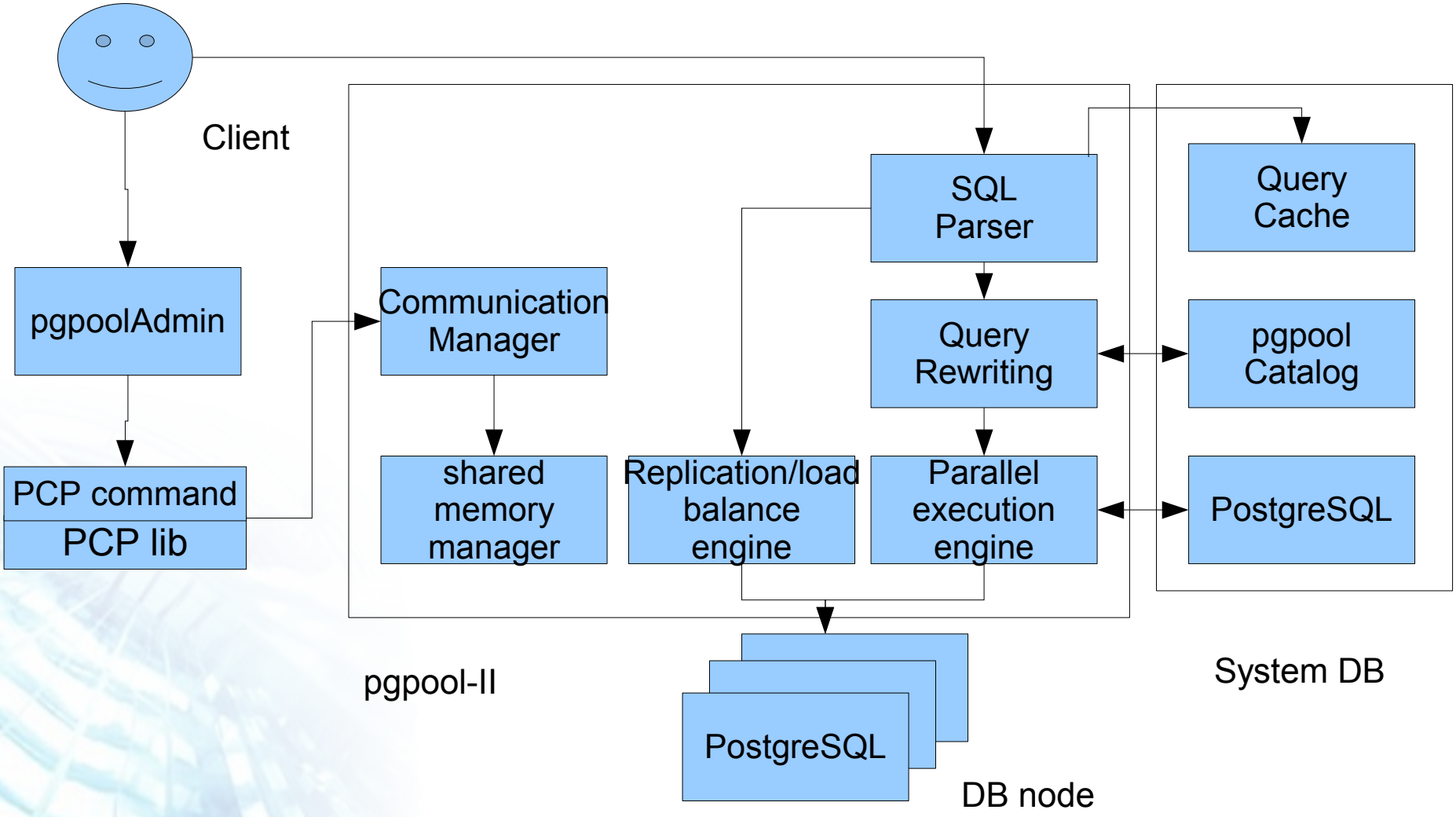
PostgreSQL用のクラスタ・レプリケーションソフトの例

- Slony-I
  - 非同期レプリケーション
- PGCluster
  - 同期レプリケーション，負荷分散
- pgpool
  - 同期レプリケーション，負荷分散，DBサーバ2台まで
- pgpool-II
  - pgpoolの機能に加え，parallel queryをサポート，DBサーバ128台まで

pgpool-IIとは

- 独立行政法人情報処理推進機構(IPA)の補助を受け，SRA OSS, Inc.日本支社が開発
- 現在のpgpoolを置き換えるもの
- 9月に最初のバージョンをオープンソースで公開
- ノード数の増加，GUIツール，parallel queryなど機能もりだくさん(11Kステップから74Kステップに増加)

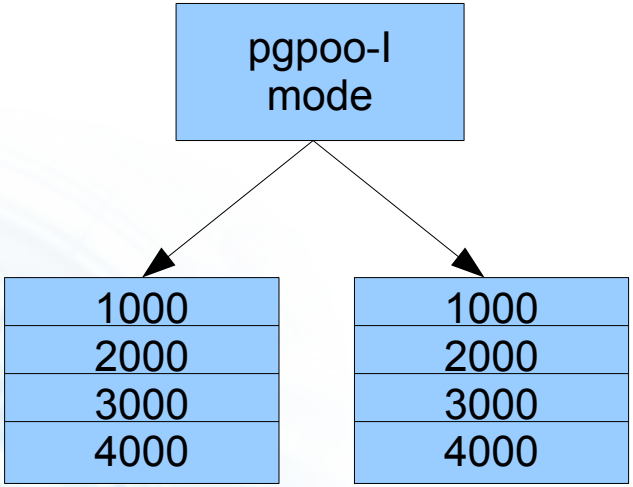
pgpool-II architecture overview

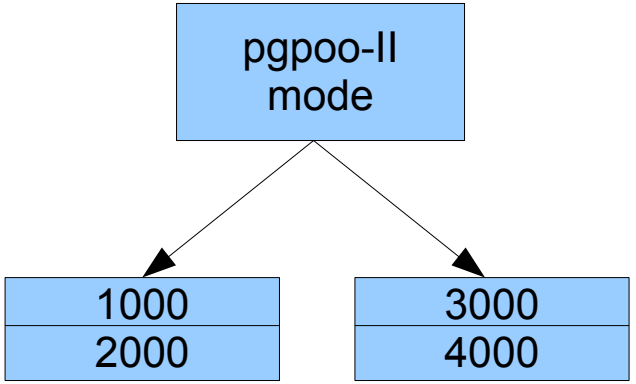


pgpool-IとpgpooI-II mode

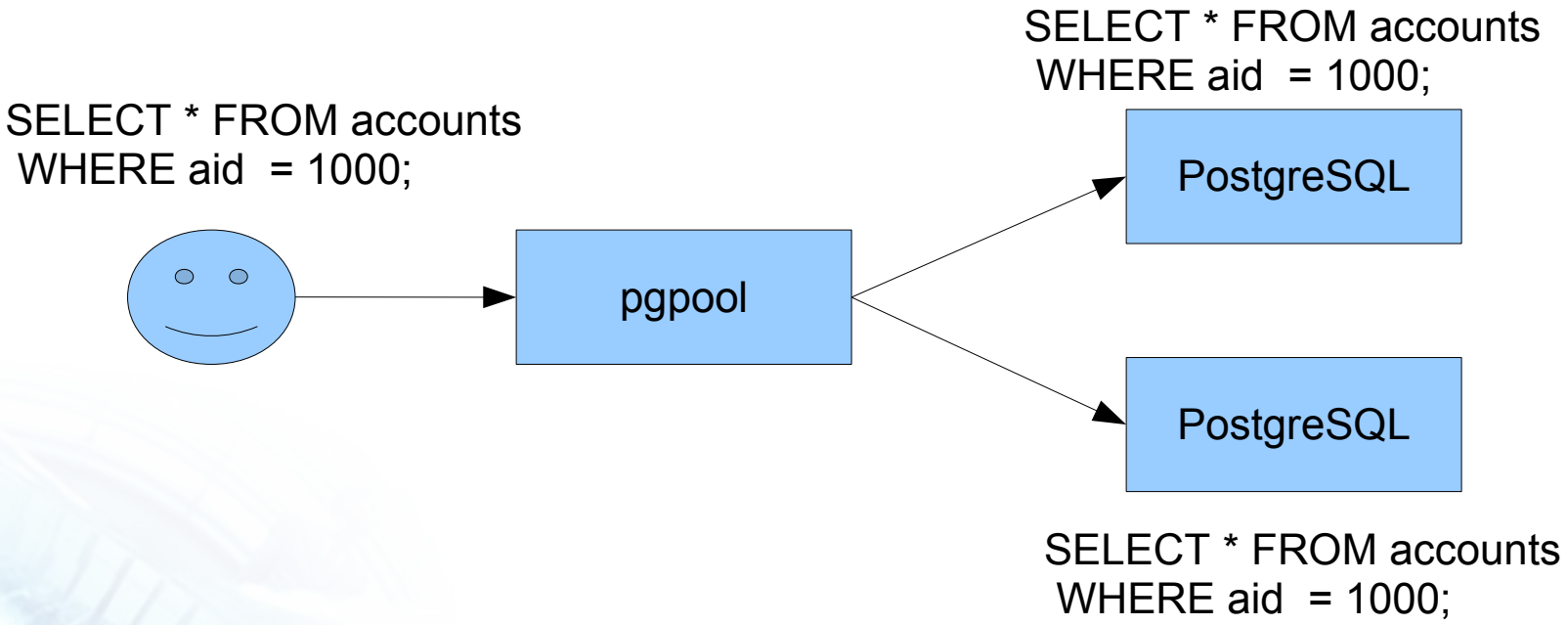
- replication
- load balance
- fail over
- virtually compatible with pgpool

- parallel query





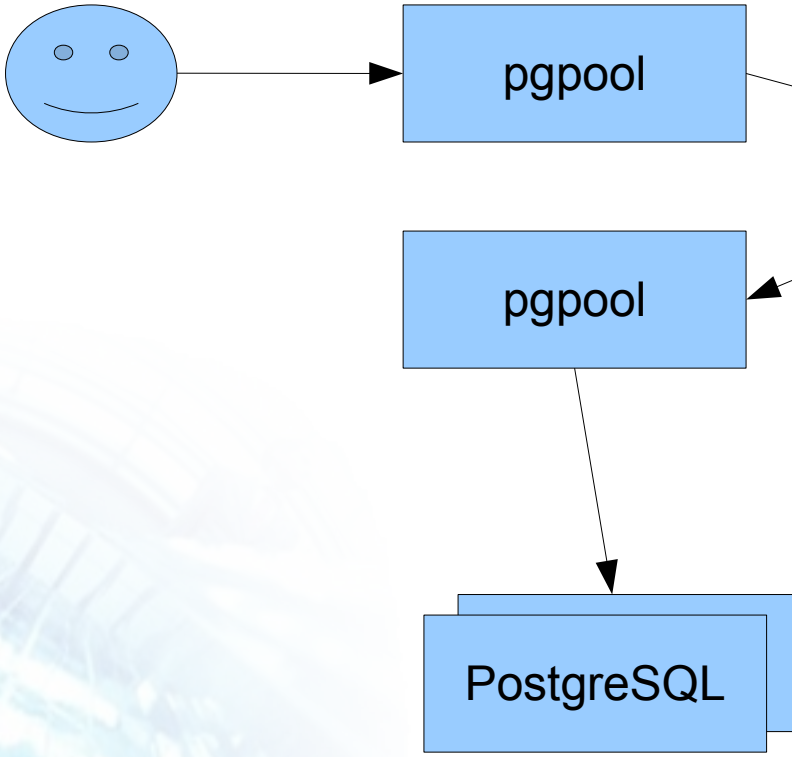
単純な問い合わせ



複雑な問い合わせ

SELECT * FROM accounts
 WHERE aid = 1000
ORDER BY aid;

SELECT * dblink('con',
'SELECT pool_parallel('SELECT * FROM accounts
 WHERE aid = 1000')') AS foo(...)
ORDER BY aid;

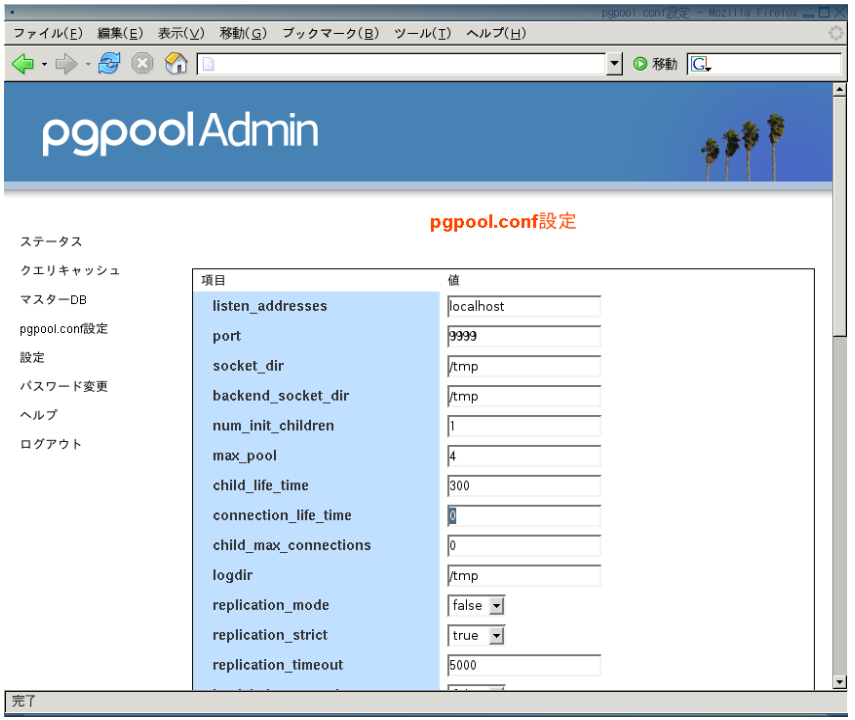


SELECT pool_parallel('SELECT * FROM accounts
WHERE aid = 1000')')

SELECT * FROM accounts
 WHERE aid = 1000;

pgpoolAdmin

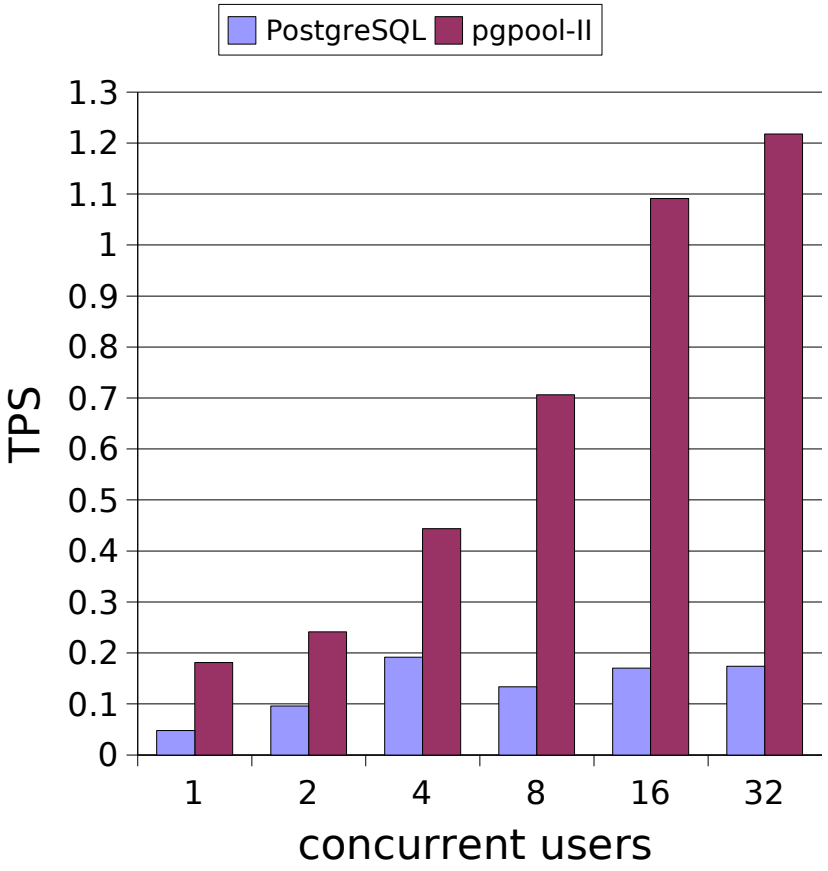
- Webベースのpgpool-II管理ツール
- Apache/PHP/Smarty
- 機能
  - pgpoolの起動，停止
  - スイッチオーバー
  - 各種モニタリング
  - 設定ファイルの編集
  - クエリキャッシュの管理



ベンチマーク結果

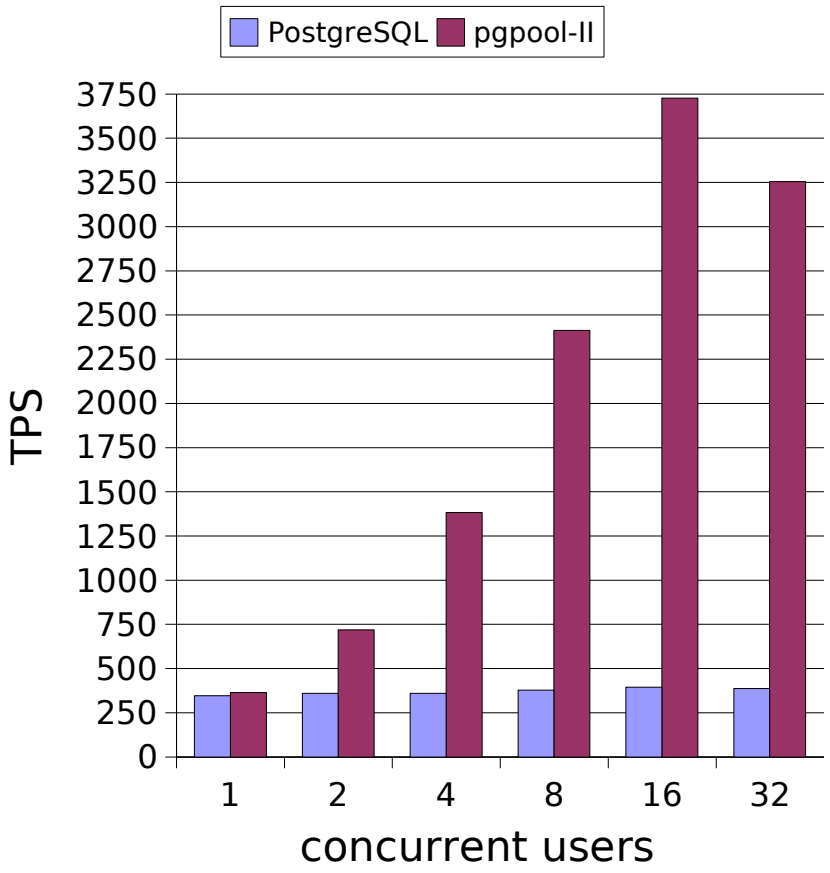
- 日立Blade Symphony BS1000
  - 10 blades
  - Xeon 3.0GHz x 1
  - 1GB Mem
  - UL320 SCSI Disks
- Cent OS 4.3
- PostgreSQL 8.1.4/pgbench

順スキャン(mid size)



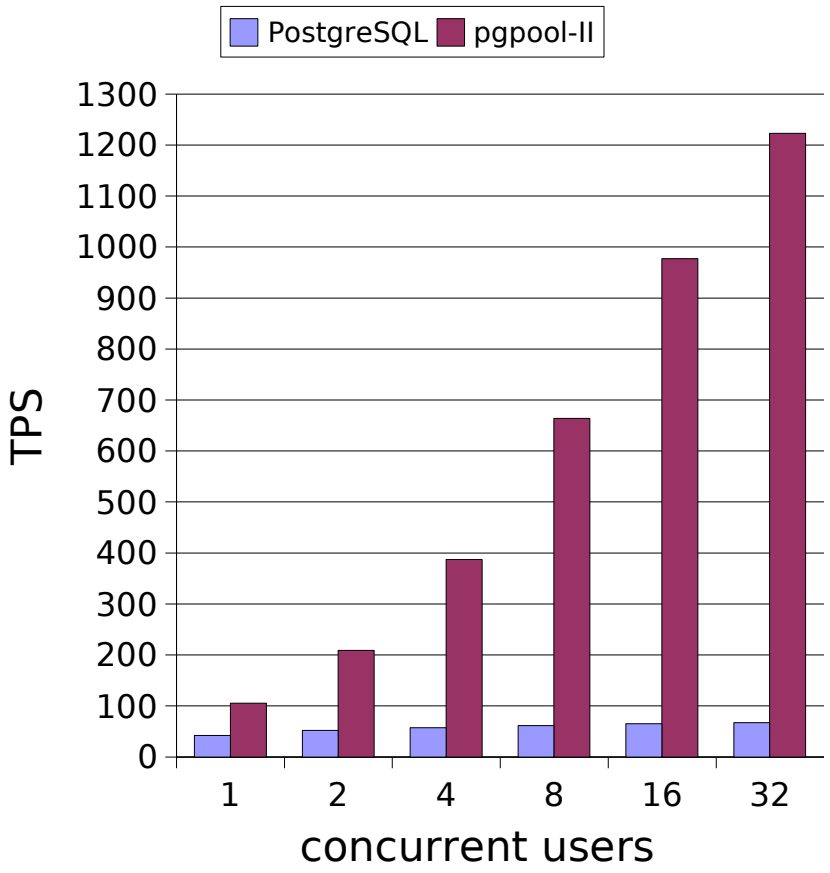
- scale factor = 90 (900万件)
- SELECT * FROM accounts WHERE aid = :aid
- pgpool-II が最高7 倍速い(32 concurrent users)

インデックススキャン(mid size)



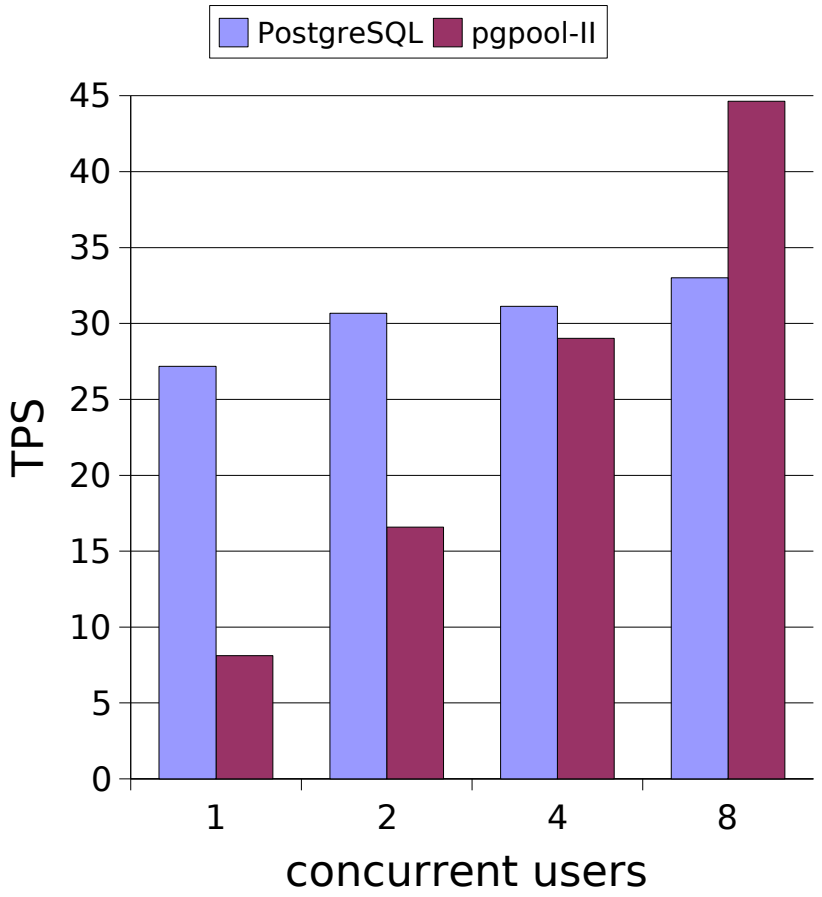
- scale factor = 90 (900万件)
- SELECT * FROM accounts WHERE aid = :aid
- pgpool-II が最高9 倍速い(16 concurrent users)

インデックススキャン(large size)



- scale factor = 900 (9千万件)
- SELECT * FROM accounts WHERE aid = :aid
- pgpool-IIが最高18 倍速い (32 concurrent users)

複雑な問い合わせ



- scale factor = 900 (9000万件)
- SELECT a1.abalance FROM accounts as a1 ,accounts as a2 WHERE a1.aid = :aid1 and a2.aid = :aid2
- pgpool-II は8ユーザ以上でないと速くない

今後の計画

- レプリケーション時の更新性能向上
- オンラインリカバリ
- シーケンスの同期

参考情報

- Version 1.0.0のリリースは2006/9/8
  - http://pgfoundry.org/projects/pgpool/にて配布
- SRAOSSにてサポートサイト開設
  - http://pgpool.sraoss.jp
- 参考URL
  - PostgreSQL Anniversary Summitでのプレゼン資料
    - http://www.sraoss.co.jp/event_seminar/2006/pgpool_feat_and_devel.pdf
  - 日経IT Pro
    - http://itpro.nikkeibp.co.jp/article/COLUMN/20060626/241783/

pgpool-II デモシステムの構成