

pgpool-II の今と今後の展望

SRA OSS, Inc. 日本支社
浅羽 義之

pgpool-II 開発背景

	PostgreSQL への要求	シングルサーバの限界
可用性	限りなくダウンタイムをゼロにしたい！	ハードウェアの故障による可用性の低下
性能	もっと速く！	リソースの物理的な限界 <ul style="list-style-type: none"> • CPU スケールアップ • 搭載メモリの限界 • I/O の集中

1 PostgreSQL、1 サーバでは超えることのできない壁

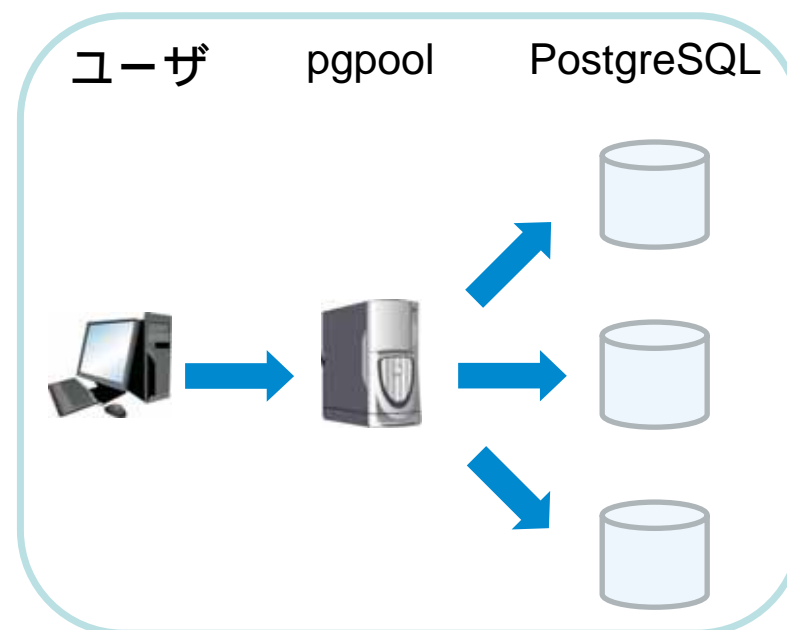
これを解決するために・・・

pgpool-II を開発

可用性と性能の両方を向上させる
オープンソースミドルウェア！！

pgpool-II の特徴

- ユーザと複数台の PostgreSQL の間に
 pgpool-II を設置することで、ユーザからは1台のデータベースに見える



BSDライセンスでソースコードを公開

pgpool-II の歴史

2004年	pgpool 初期版リリース
2006年	IPA の支援により SRA OSS で pgpool-II 開発
2007年	SRA OSS で開発を継続し pgpool-II v2.0 リリース
現在	pgpool Global Development Group で 開発&メンテナンス

pgpool-II の機能

pgpool-II の主な提供機能

- 可用性
 - 同期レプリケーション
- オンラインリカバリ
 - Slony-Iとの連携
 - Warm standby
- 性能
 - 参照クエリの負荷分散
 - パラレルクエリ
 - クエリキャッシュ
 - コネクションプール

運用管理

- Webベース管理ツール
- 独自の管理プロトコルの提供

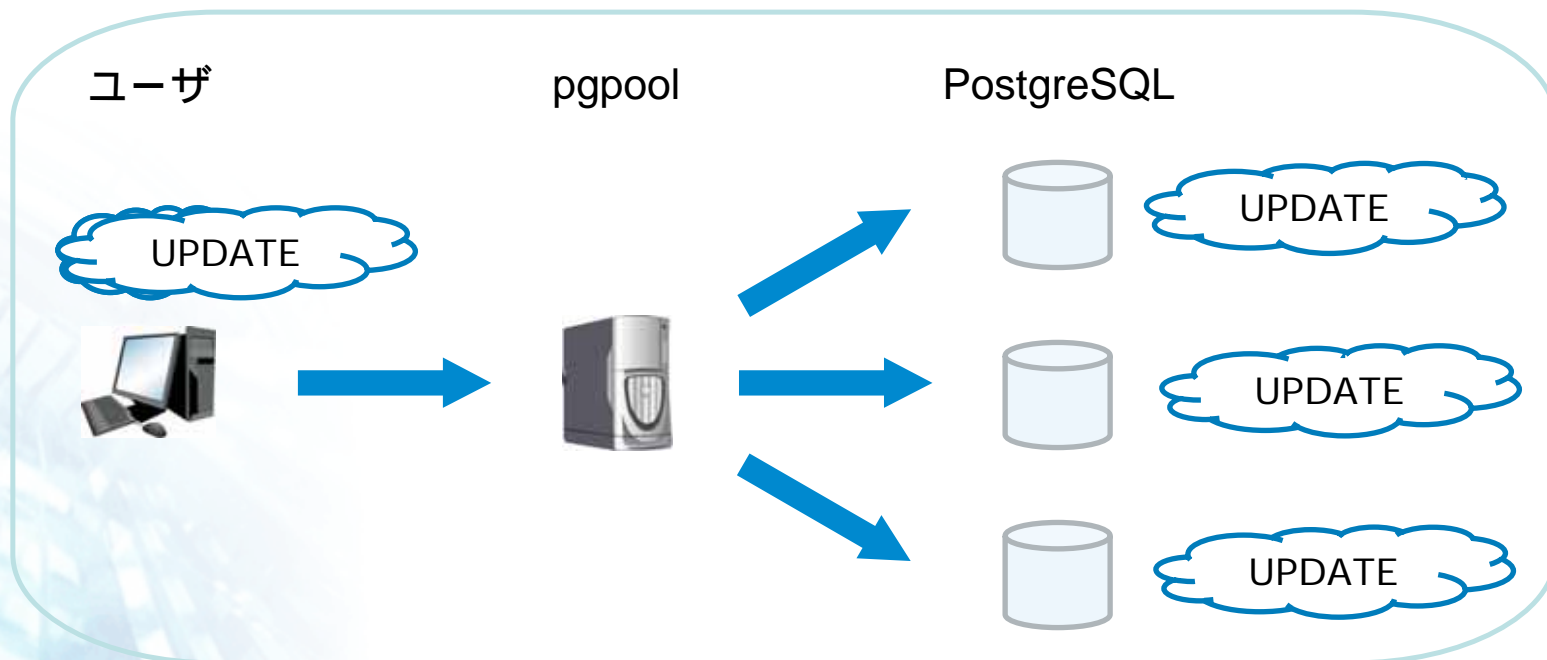


可用性機能 其の一

同期レプリケーション

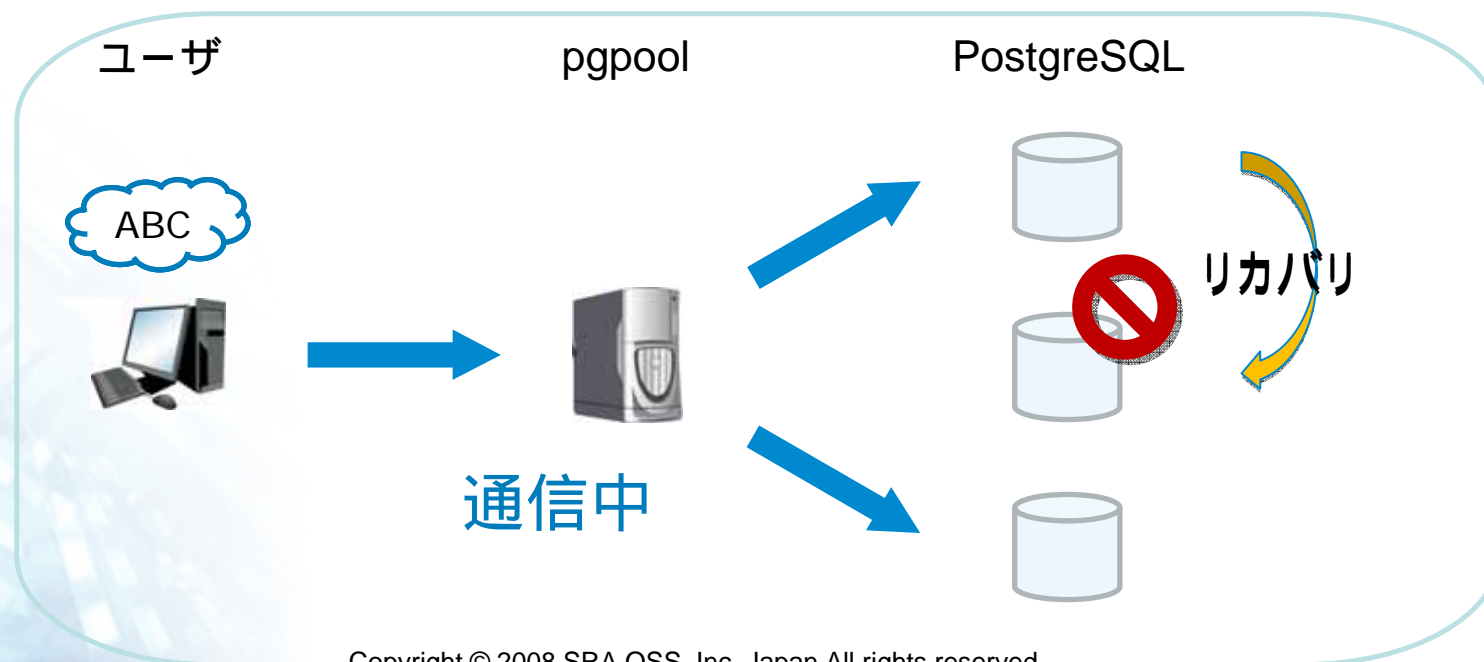
同期レプリケーション

- 更新クエリを複数サーバに送信し同期を行う
- 障害が発生した場合にはフェイルオーバーを行い運用を継続
- 1行の更新性能は約1/2



同期レプリケーション オンラインリカバリ

- サービスを停止することなくノード復旧が可能
 - 新規ノードの追加も可能
 - 無停止でアップデートも可能
- 洗練されたログベースによるリカバリ
- 障害が発生してから、オンラインリカバリの設定ができる!
- 管理ツールからボタン一発操作!

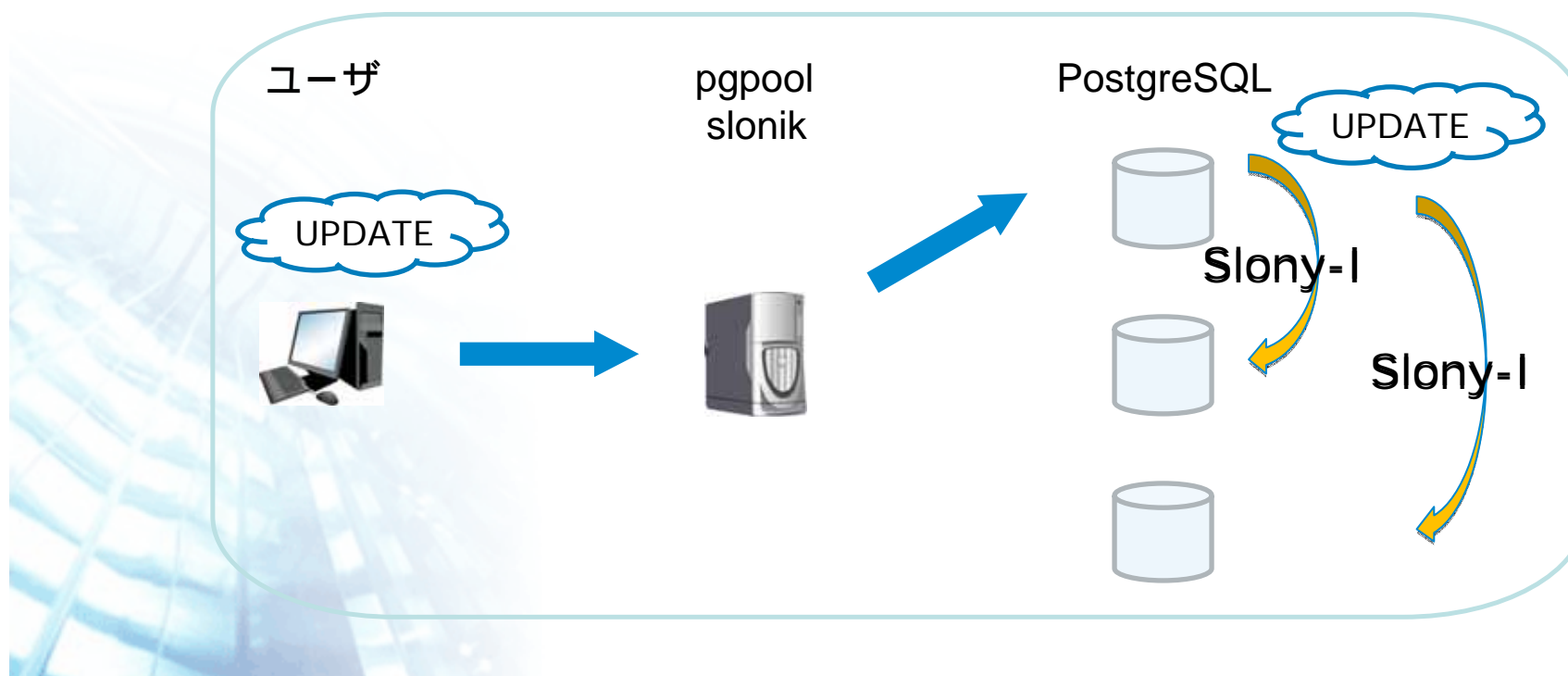


可用性機能 其の二

Slony-I との連携

Slony-I との連携

- Slony-Iによる非同期レプリケーション
- 更新クエリはマスタのみに送信
- 自動フェイルオーバー+ Slony-Iのマスタの切り替え
- 新規ノードの追加も可能
- テーブルの追加などスキーマの変更ができない
- 非同期レプリケーションのため、フェイルオーバー時にデータをロストする可能性がある

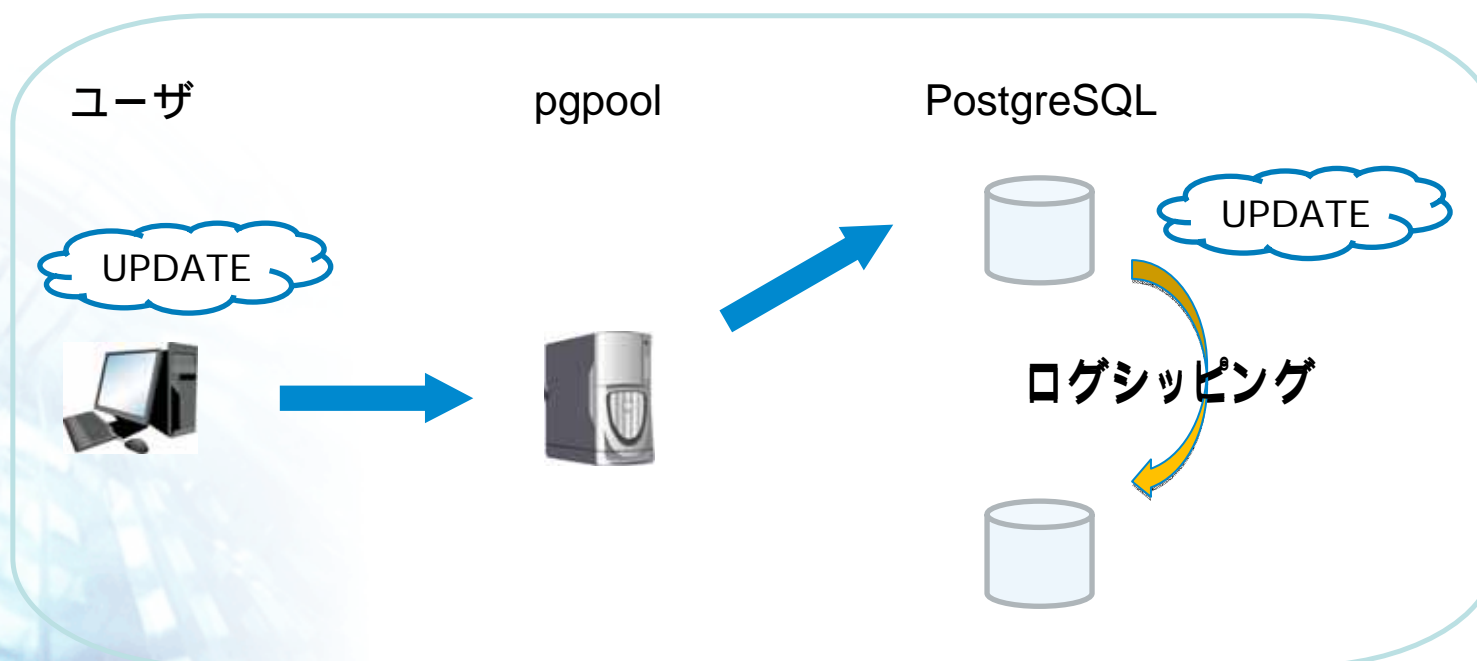


可用性機能 其三

Warm standby

Warm standby

- PostgreSQLのPITRによるログ SHIPPINGを行う
- 更新クエリはマスタのみに送信
- 障害が発生した場合には、自動フェイルオーバー処理を実行
- フェイルオーバー後に、サーバを一台追加してバックアップサーバの構築も可能
- 障害発生時にWAL領域のデータがロストする

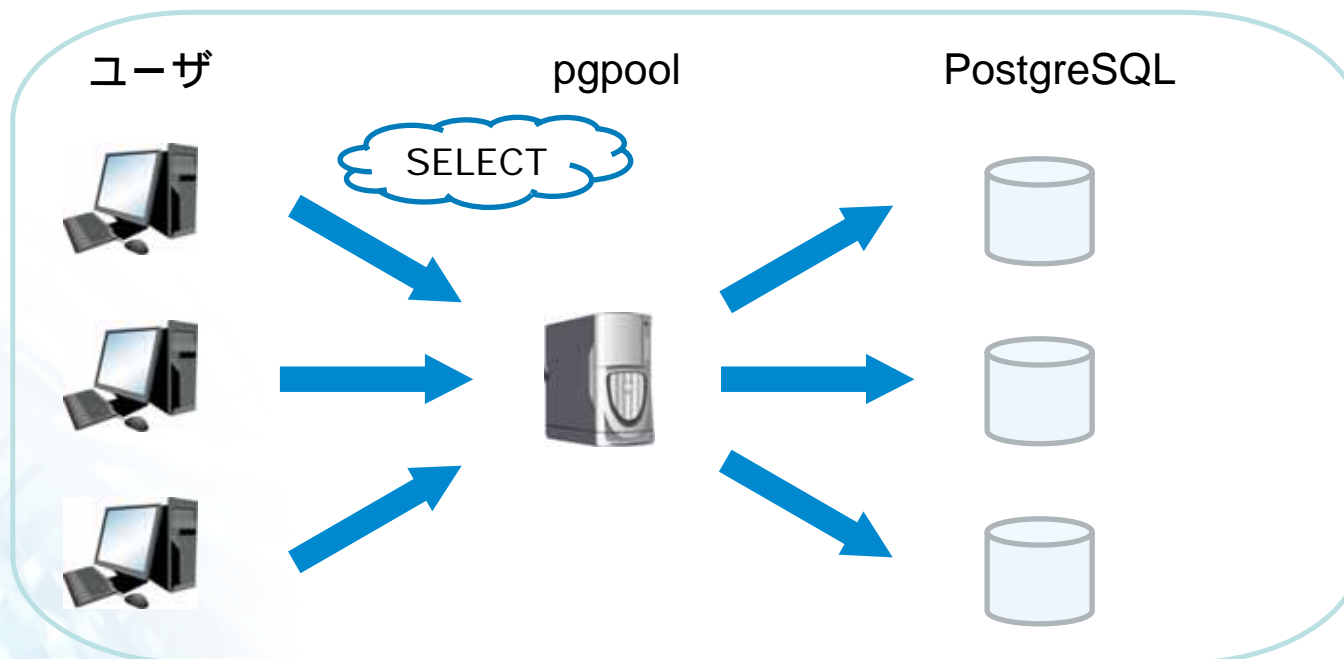


性能機能

負荷分散

負荷分散

- 同期レプリケーション、Slony-Iとの連携を行っている場合のみ有効
- 検索クエリは重みをつけてランダムに振り分ける
- 負荷を PostgreSQL 間で分かち合うので検索性能が向上



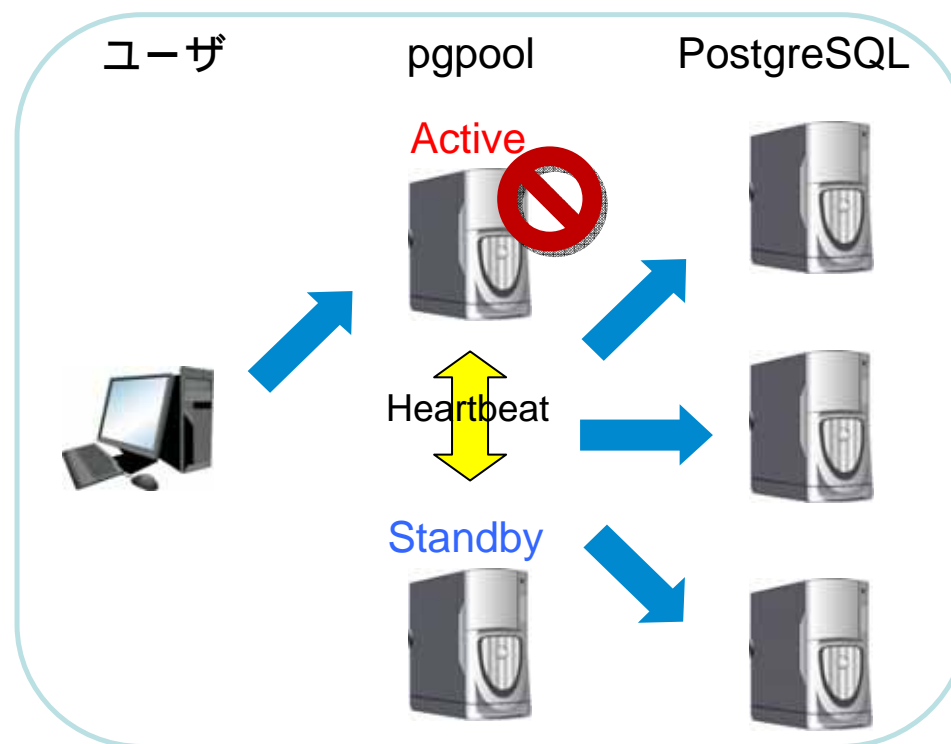
Web システムなどに最適

その他の機能

- パラレルクエリ (性能)
- コネクションプーリング (性能)
- クエリキャッシュ (性能)
- 障害時に指定したコマンドを実行 (運用管理)
 - 管理者へのメール送信など

pgpool-HA

- オープンソースの HeartBeat と組み合わせてさらなる可用性
- Active-Standby 構成で pgpool の死活監視
- 仮想 IP を利用することでクライアントはサーバが切り替わったことを意識する必要がない
- 現状ではオンラインリカバリはできない



デモ 其の一

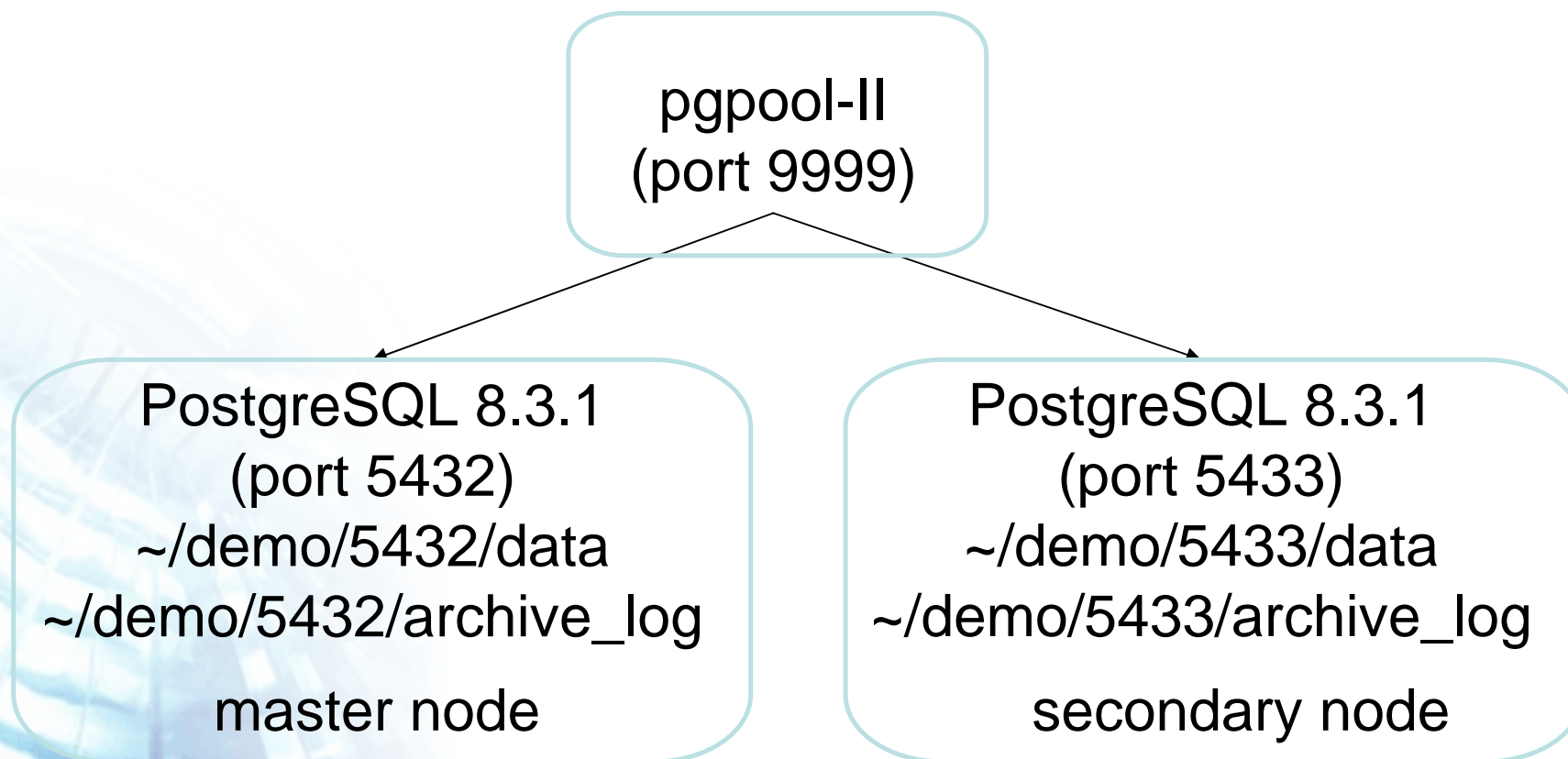
オンラインリカバリ

オンラインリカバリの方法

- 2段階に分けてデータ同期
 - ファーストステージ
 - おおざっぱにデータ同期
 - おすすぬ rsync
 - DBの更新可能
 - セカンドステージ
 - すべての接続が終了するまで待機
 - 新規接続リクエストをブロック
 - 最終同期
 - おすすぬ PITRによるリカバリ
- 最後によりモートサーバのPostgreSQLを起動

デモ環境

- pgpool-IIとPostgreSQLは同じ環境



pgpool-II 設定(1)

pgpool.conf

```
# Master node info
backend_hostname0 = 'localhost'
backend_port0 = 5432
backend_data_directory0 = '/home/y-asaba/demo/5432/data'

# Secondary node info
backend_hostname1 = 'localhost'
backend_port1 = 5433
backend_data_directory1 = '/home/y-asaba/demo/5433/data'
```

pgpool-II 設定(2)

pgpool.conf

```
# Enable replication
replication_mode = true
# Enable load balance
load_balance_mode = true
# Online recovery user
recovery_user = 'y-asaba'
# The first stage script
recovery_1st_stage_command = 'base-backup.sh'
# The second stage script
recovery_2nd_stage_command = 'pgpool-recovery'
```

ファーストステージ

ベースバックアップの作成 base-backup.sh

```
#!/bin/sh
PORT=5432
DATA=$HOME/demo/$PORT
```

\$1 ローカルホストのPGDATA
\$2 リモートサーバのホスト名
\$3 リモートサーバのPGDATA

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

```
# Generate recovery.conf.
```

```
echo "restore_command = 'cp $DATA/archive_log/%f %p'"
>$DATA/data/recovery.conf
```

```
# Copy base backup.
```

```
rsync -az --delete -e ssh --exclude postmaster.pid --exclude postmaster.opts
--exclude pg_log --exclude postgresql.conf $1/ $3/
```

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

セカンドステージ

WALのアーカイブ化 pgpool-recovery.sh

```
#!/bin/sh  
# Archive a current xlog.  
psql -c 'select pg_switch_xlog()' postgres
```

pg_remote_start

- PostgreSQLを起動するためのスクリプト
- PGDATAの下に置いておく

```
#!/bin/sh
DEST=$1
DESTDIR=$2
PGCTL=/home/y-asaba/bin/pg_ctl

$PGCTL -w -D $DESTDIR start
```

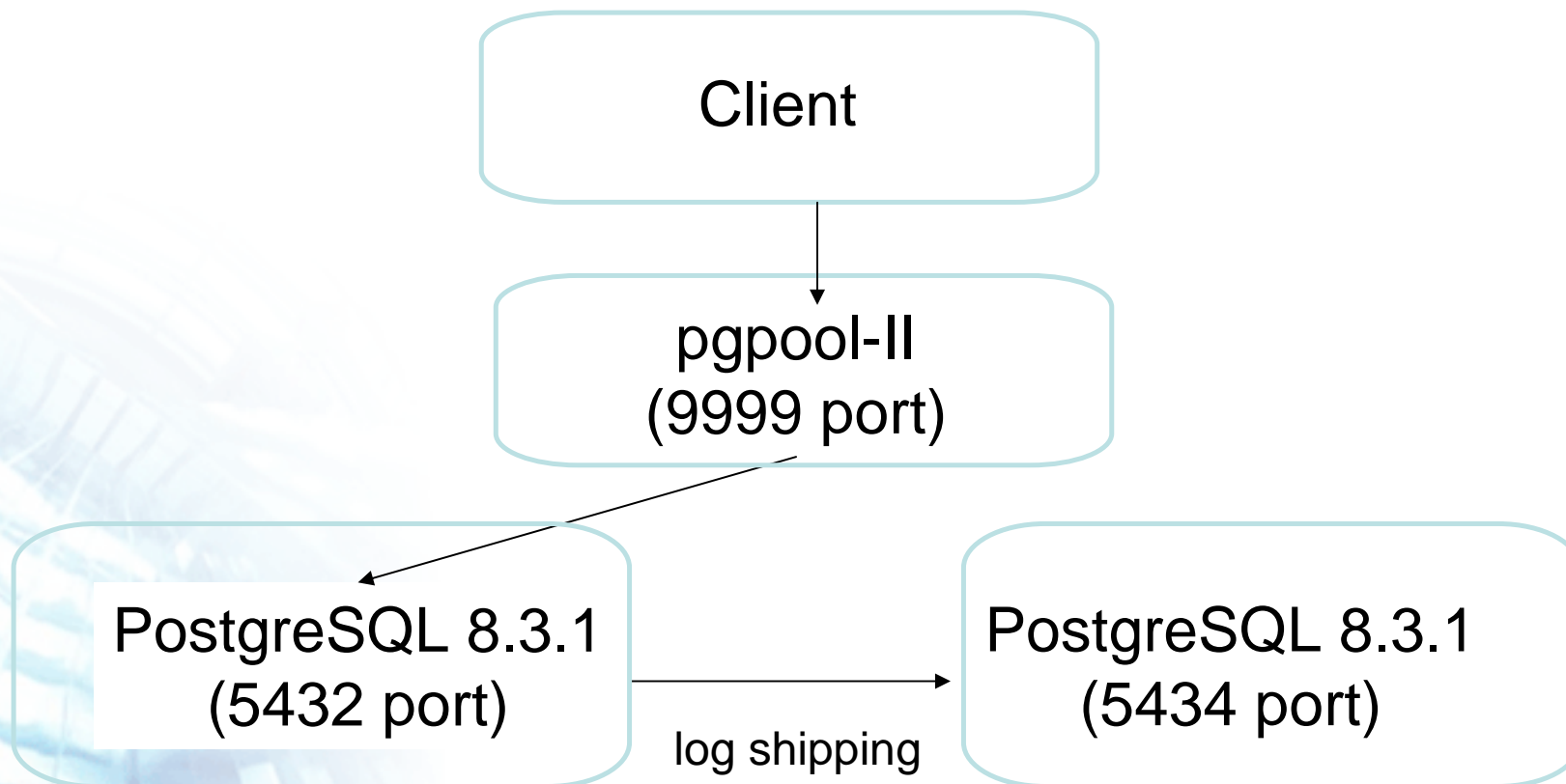
\$1 リモートサーバのホスト名
\$2 リモートサーバのPGDATA

デモ 其の二

Warm standby

pgpool-II & warm standby

- Pgpool-II が障害を検知してフェイルオーバーを行う



pgpool-II & warm standby

1. warm standbyの設定
2. pgpool-IIの設定
3. pgpool-IIの起動
4. `createdb -p 9999 warm`
5. `pgbench -i -p 9999 warm`
 - `SELECT COUNT(*) FROM accounts;`
6. PostgreSQLの停止
7. failover

warm standbyの設定

- recovery.conf

```
restore_command = 'pg_standby -s -t /tmp/pgsql.trigger  
~/demo/5432/archive_log %f %p %r'
```

- postgresql.conf

```
port = 5434  
archive_mode = on  
archive_command = 'cp -f %p ~/y-asaba/demo/5432/archive_log/%f'  
archive_timeout = 60
```

Set up pgpool

- pgpool.conf
 - ssh でパスワード入力なしで接続できること!

```
backend_hostname0 = 'localhost'
```

```
backend_port0 = 5432
```

```
backend_hostname1 = 'localhost'
```

```
backend_port1 = 5434
```

```
failover_command = 'ssh localhost touch /tmp/pgsql.trigger'
```

```
replication_mode = false
```

```
master_slave_mode = false
```

```
parallel_mode = false
```

フェイルオーバー

- pgpool.conf

failover_command = 'ssh localhost touch /tmp/trigger'

pgpoolがfailover_commandを実行する

pgpool-II
(9999 port)

touch /tmp/pgql-trigger

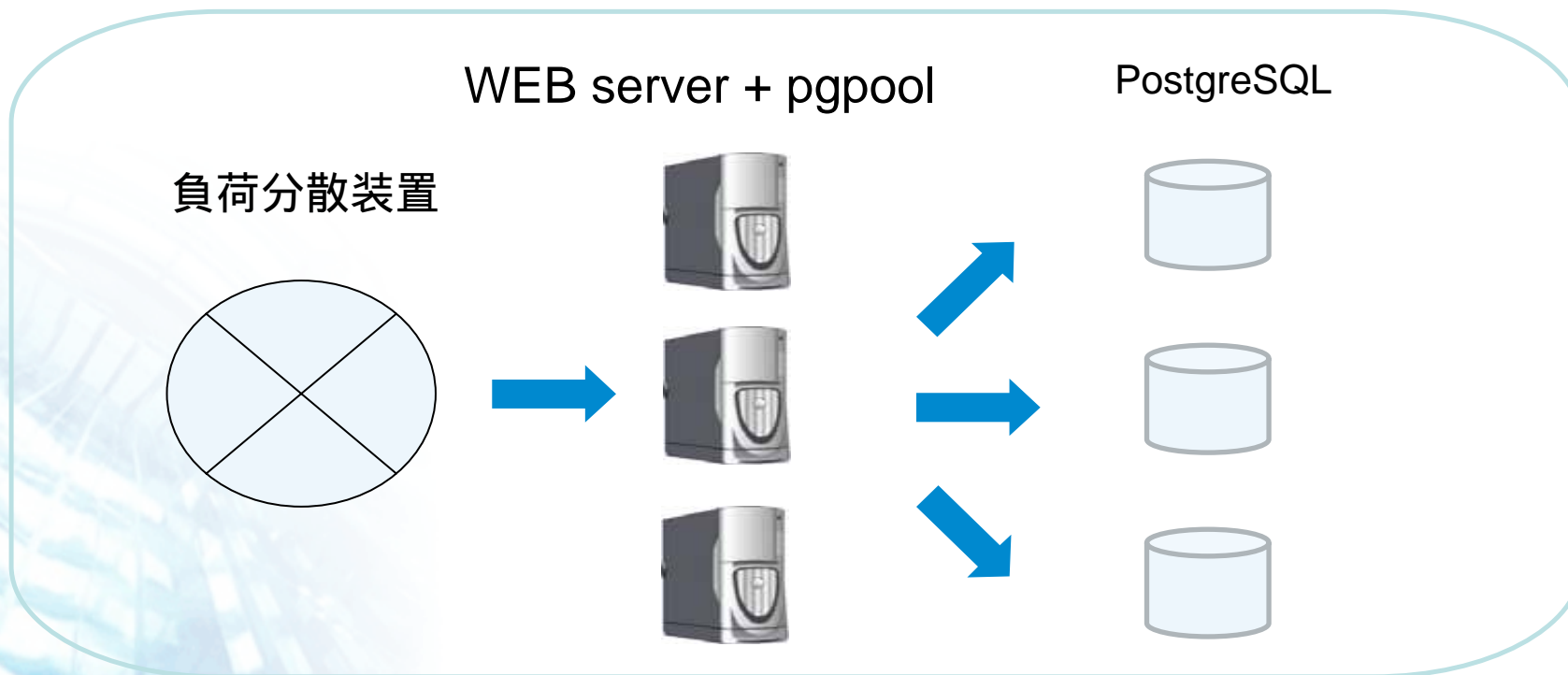
PostgreSQL 8.3.1
(5432 port)

PostgreSQL 8.2.5
(5434 port)

pg_standbyが
/tmp/pgsql-trigger を監視

今後の予定

- オンラインリカバリも含めpgpool-II複数台構成のサポート



ご清聴ありがとうございました

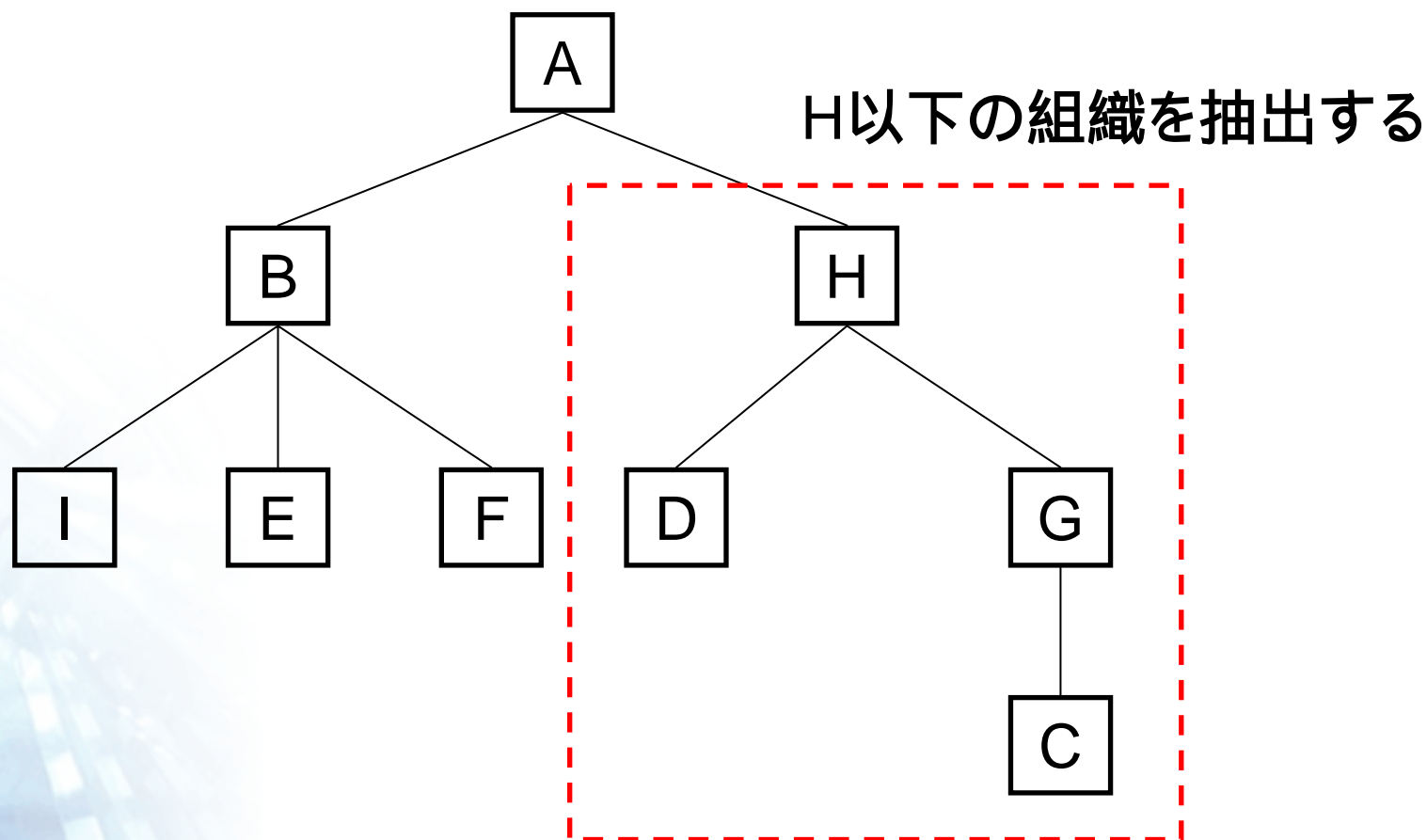
時間が余ったので別の話題
pgpoolとは関係ない
再帰問合せのデモ

SQLの弱点

- SQLは構造的なデータの扱いに弱い
 - テーブルは集合論をベースとしている
- 構造的なデータ
 - 木構造
 - 部分木を取り出す操作等
 - グラフ構造
 - 最短経路問題

再帰問合せをSQL99でサポートし、
シンプルに構造的なデータを扱うことが可能

再帰問合せ例：組織階層を表現



テーブル構造

ノードID	親ノードID	名前
-------	--------	----

department テーブル

```
CREATE TABLE department (  
  id INT PRIMARY KEY,  
  parent_id INT REFERENCES department,  
  name TEXT  
);
```

parent_id の外部キー制約で自己参照

データ

```
-- root node
INSERT INTO department VALUES (-1, NULL, 'ROOT');
INSERT INTO department VALUES (0, -1, 'A');
INSERT INTO department VALUES (1, 0, 'B');
INSERT INTO department VALUES (2, 0, 'H');
INSERT INTO department VALUES (3, 1, 'I');
INSERT INTO department VALUES (4, 1, 'E');
INSERT INTO department VALUES (5, 1, 'F');
INSERT INTO department VALUES (6, 2, 'D');
INSERT INTO department VALUES (7, 2, 'G');
INSERT INTO department VALUES (8, 7, 'C');
```

再帰問合せによる部分木の取得

- WITH RECURSIVE句
 - SQL:2008(draft)に準拠(7.13 <問合せ式>)

```
WITH RECURSIVE subdepartment(id, parent_id, name) AS
(
  -- 初期集合
  SELECT root.id, root.parent_id, root.name FROM department AS root
  WHERE root.name = 'H'
  UNION ALL
  -- 再帰集合
  SELECT child.id, child.parent_id, child.name FROM department AS child,
  subdepartment AS sd
  WHERE sd.id = child.parent_id;
)
SELECT * FROM subdepartment;
```

PostgreSQL 8.4に向けて

- **最初のバージョンのパッチを投稿**
 - いくつかコメントやバグレポートをもらい、第2版を作成中
 - 7月のCommitFestに間に合わせたい
- **テスト・コメント大歓迎**
- <http://www.sraoss.jp/~y-asaba/pgsql/recursive-query.html>

ご清聴ありがとうございました