PostgreSQLによる クラスタ運用および負荷分散術

SRA OSS, Inc. 日本支社
OSS事業本部
星合　拓馬

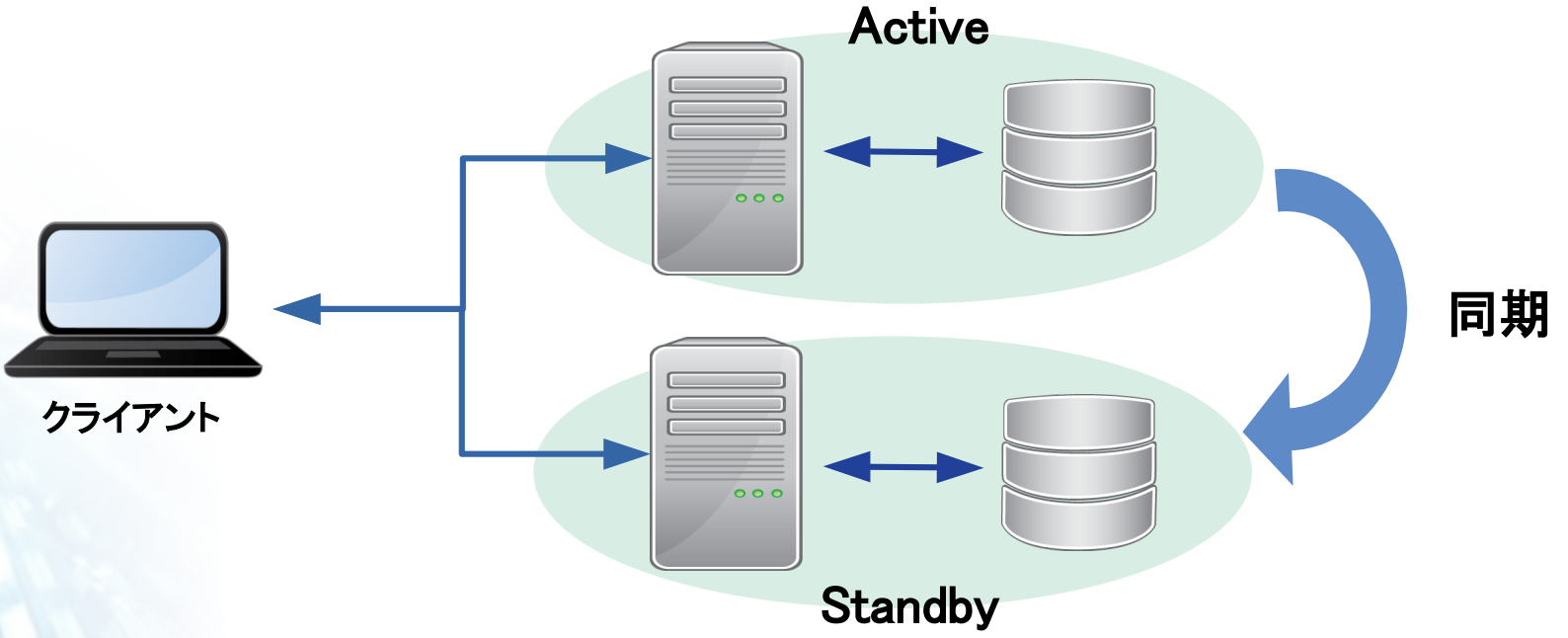
SRA OSS,INC.

自己紹介

- SRA OSS,Inc. 日本支社
  - PostgreSQLのサポート業務に従事
- OSSコミュニティ活動
  - PostgreSQLのIVM開発
  - Pgpool-IIのコミッター(2018.09～)

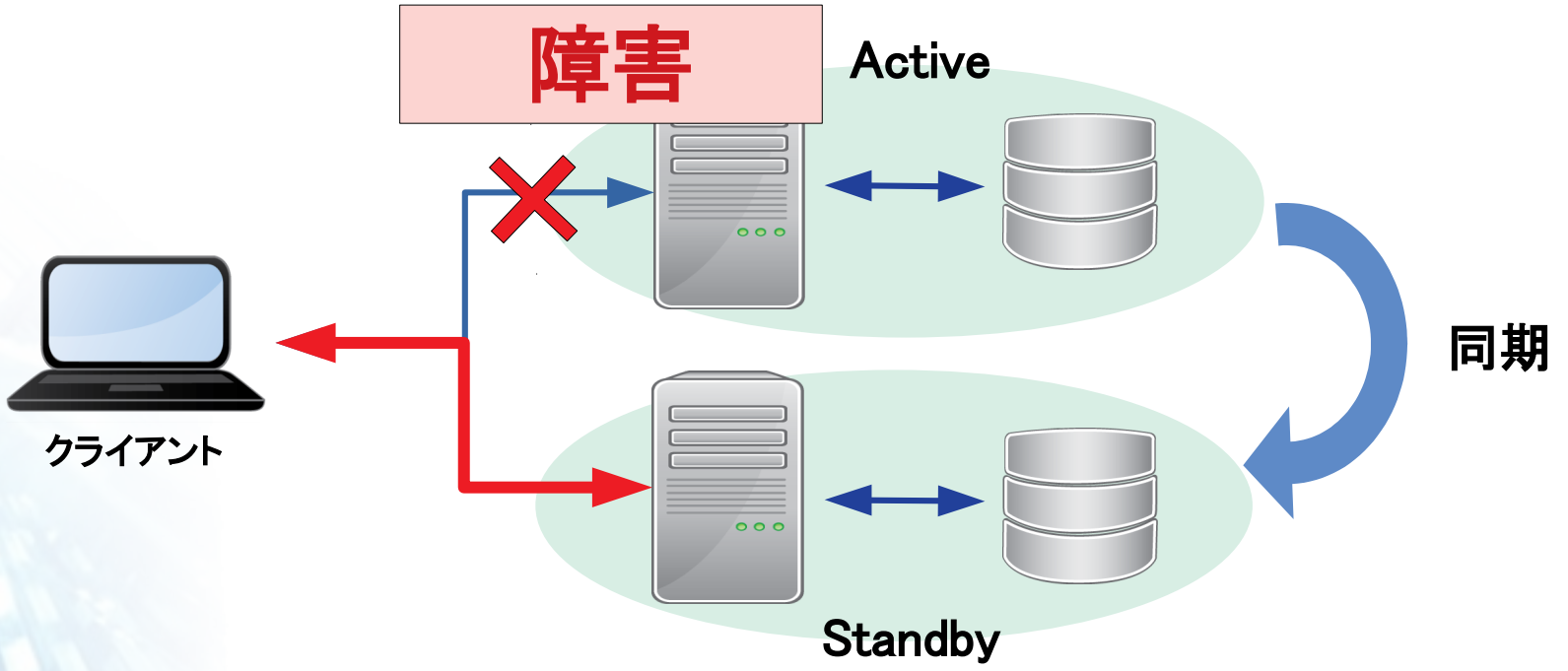
クラスタ構成

- 可用性の向上
  - 冗長化と障害時の切り替えによるダウンタイムの軽減
- 負荷分散
  - 複数サーバに処理を分散させることによる負荷の軽減



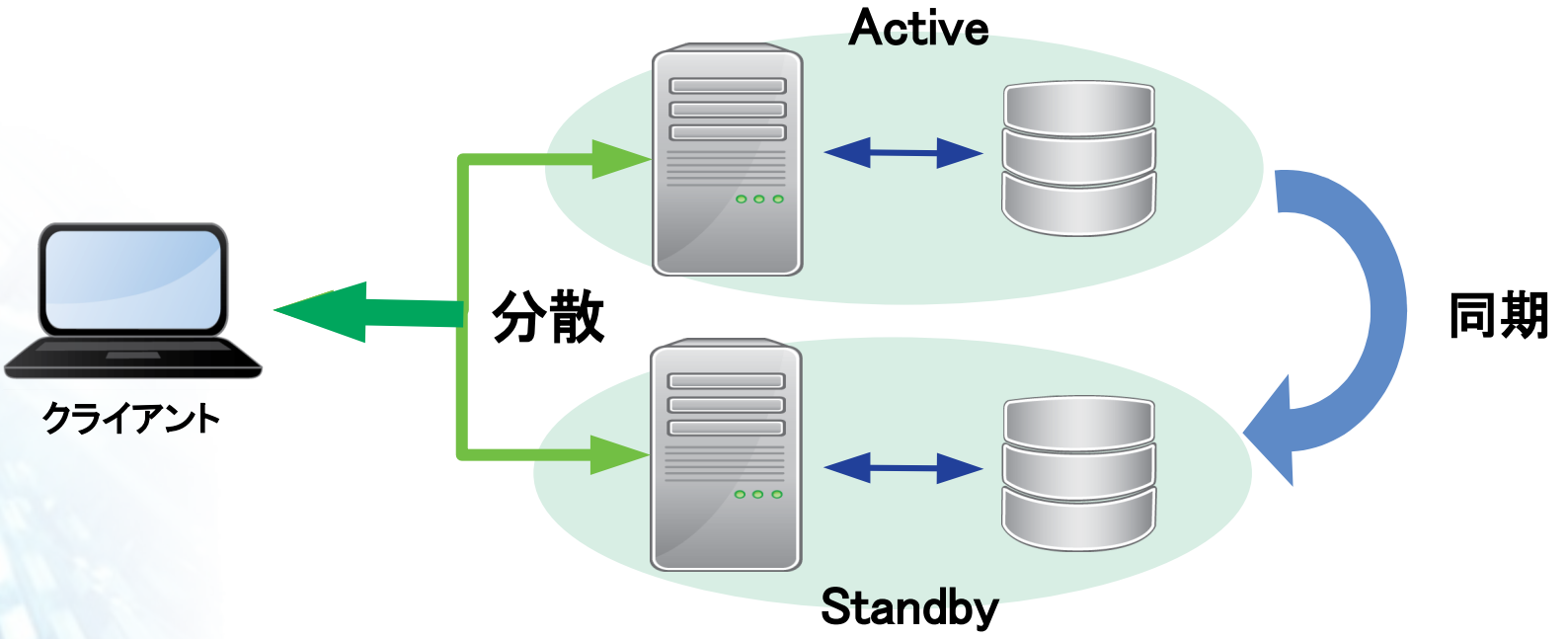
クラスタ構成

- 可用性の向上
  - 待機系(Standby)を用意することで、主系(Active)に異常が発生した際もサービスを継続可能
    - ダウンタイムの軽減



クラスタ構成

- 負荷の分散
  - 複数のサーバで処理を並列に実施(**スケールアウト**)
  - サーバ単位の負荷が軽減し高スループットを実現可能



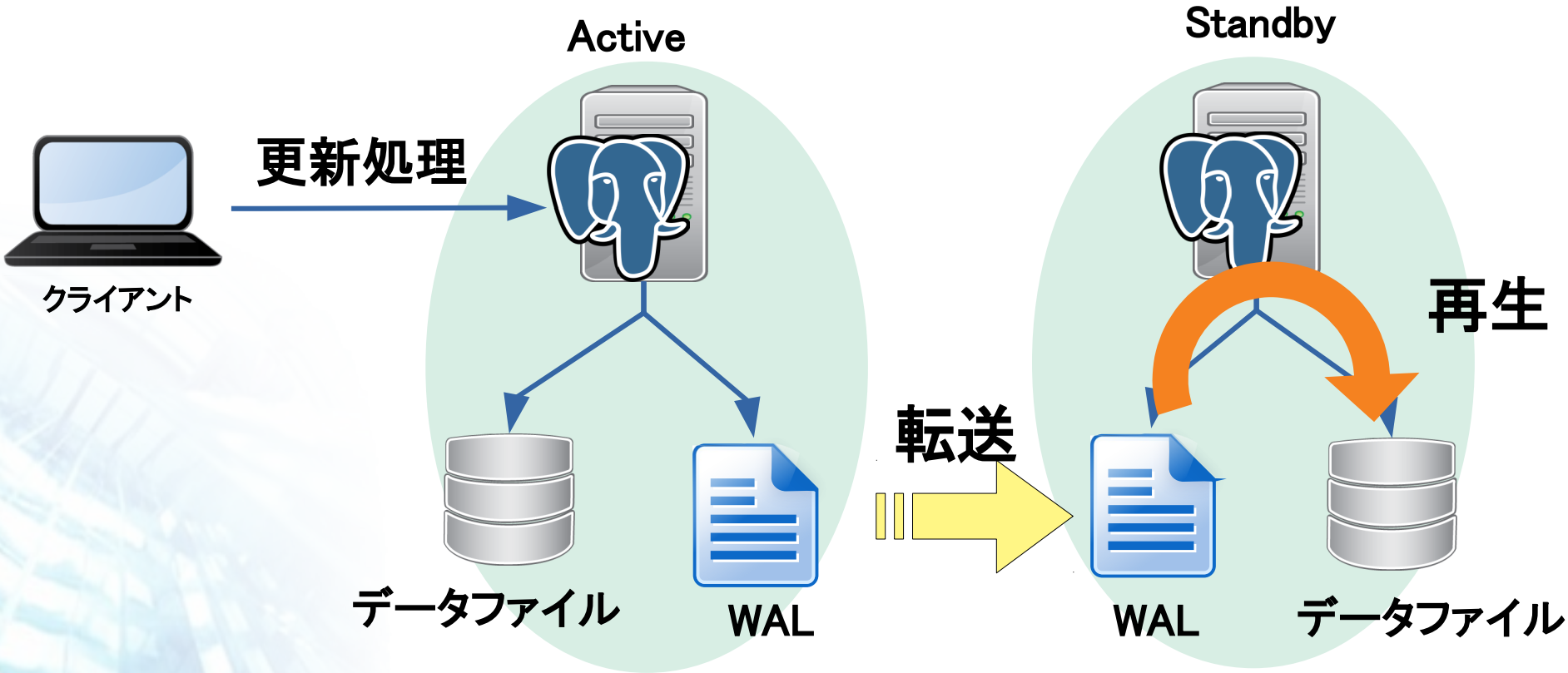
PostgreSQLのクラスタ機能

- レプリケーション
  - 別のデータベースへのデータ複製・同期
  - リアルタイムでの同期が可能

- 二つのレプリケーション機能を提供
  - ストリーミングレプリケーション(PostgreSQL 9.0～)
  - ロジカルレプリケーション(PostgreSQL 10～)

ストリーミングレプリケーション(SR)

- 主系のトランザクションログ(WAL)を待機系に転送
- 待機系でWALを再生することで同期

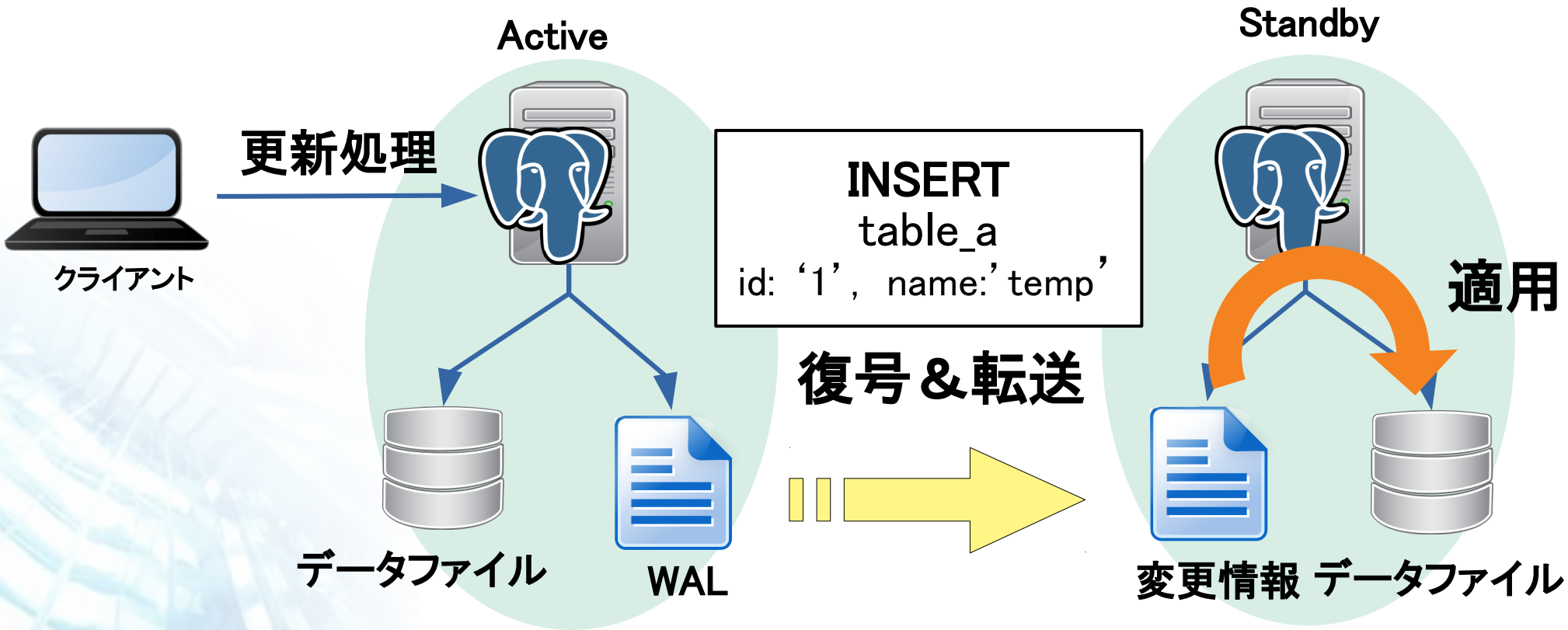


ストリーミングレプリケーション(SR)

- PostgreSQLの一般的なレプリケーション方法
- トランザクションログ(WAL)を待機系に転送
  - 物理的な変更内容で同期されるため、物理レベルで同じデータベースが作られる
  - 待機系は物理バックアップから作成する必要がある
  - PostgreSQLは同じメジャーバージョンである必要がある
- レプリケーションの対象は、すべてのデータベース
- 更新クエリは主系のみ、参照クエリは待機系も使用可能

ロジカルレプリケーション(LR)

- 主系の**論理的な変更情報**を待機系に転送
- 待機系で変更情報を適用することで同期



ロジカルレプリケーション

- PostgreSQL10で追加された新しいレプリケーション
- トランザクションログ(WAL)から論理的な変更情報をデコード(復号)し、待機系(subscriber)に転送
  - テーブル単位でレプリケーションの設定が可能
  - 異なるメジャーバージョン間でも利用可能
- 主系＆待機系どちらも参照、更新可能

クラスタ運用時の課題

- 問題が発生した際に手動での対応が必要
  - 故障時の待機系へのフェイルオーバ

    ①障害の検知

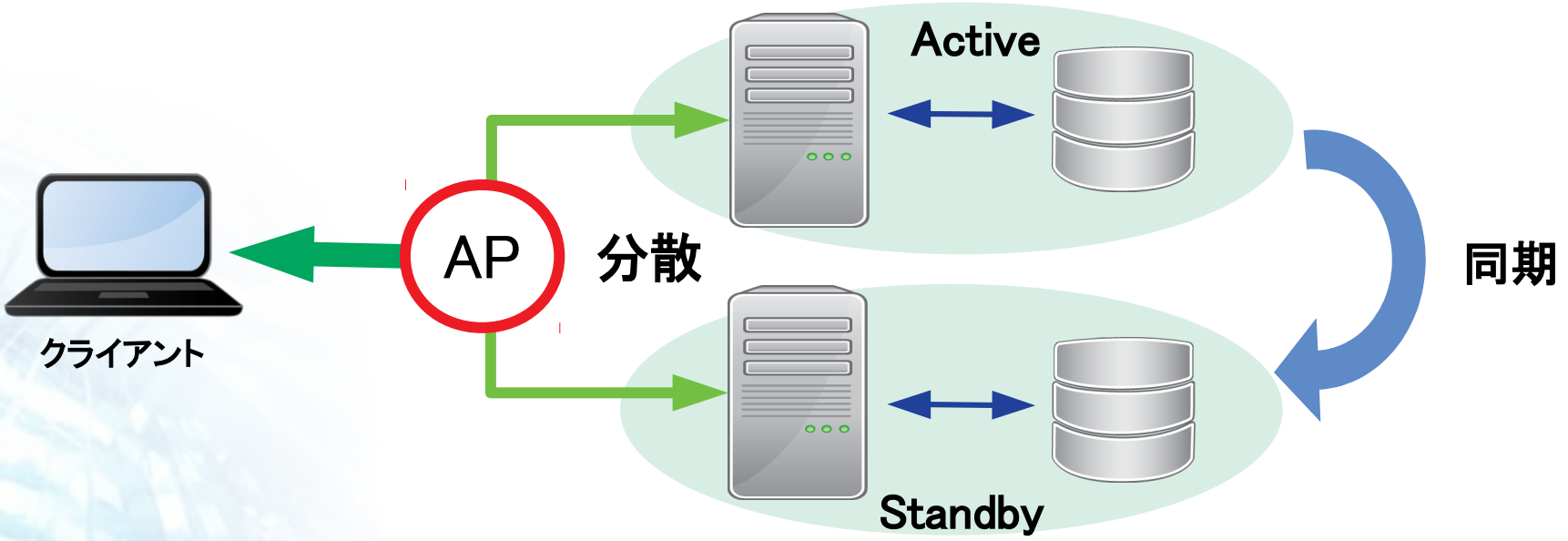
    ②待機系の昇格(promote)

    ③残りの待機系の同期先の変更

    ④クライアントの参照先の変更
  - 故障したノードの復旧
    - 障害が発生したノードを復旧し、クラスタに再度組み込む
    - オンラインバックアップからのリカバリを実施

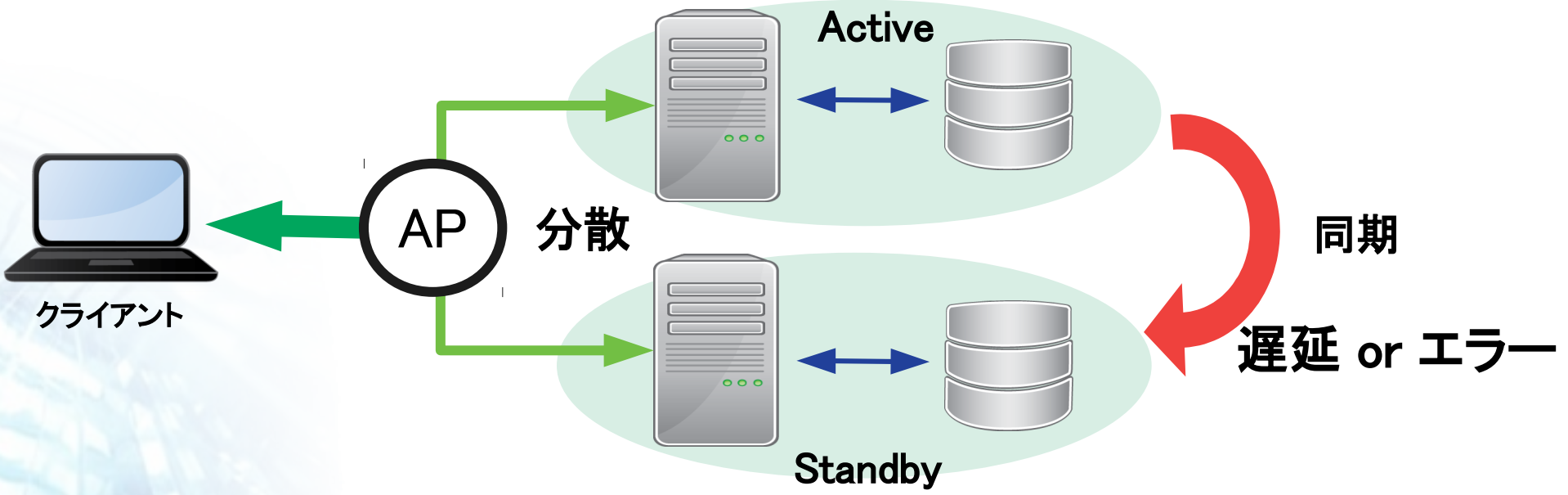
クラスタ運用時の課題

- 負荷分散の制御
  - 待機系で実行できるのは、参照クエリのみ
    - アプリケーション側で実行先を切り替える必要がある



クラスタ運用時の課題

- 負荷分散の制御
  - レプリケーションの遅延の考慮
    - 何らかの理由により、待機系への同期が遅れる場合がある
    - 主系で直前に変更されたデータが反映されず、待機系を参照した際に古い情報を返す可能性がある

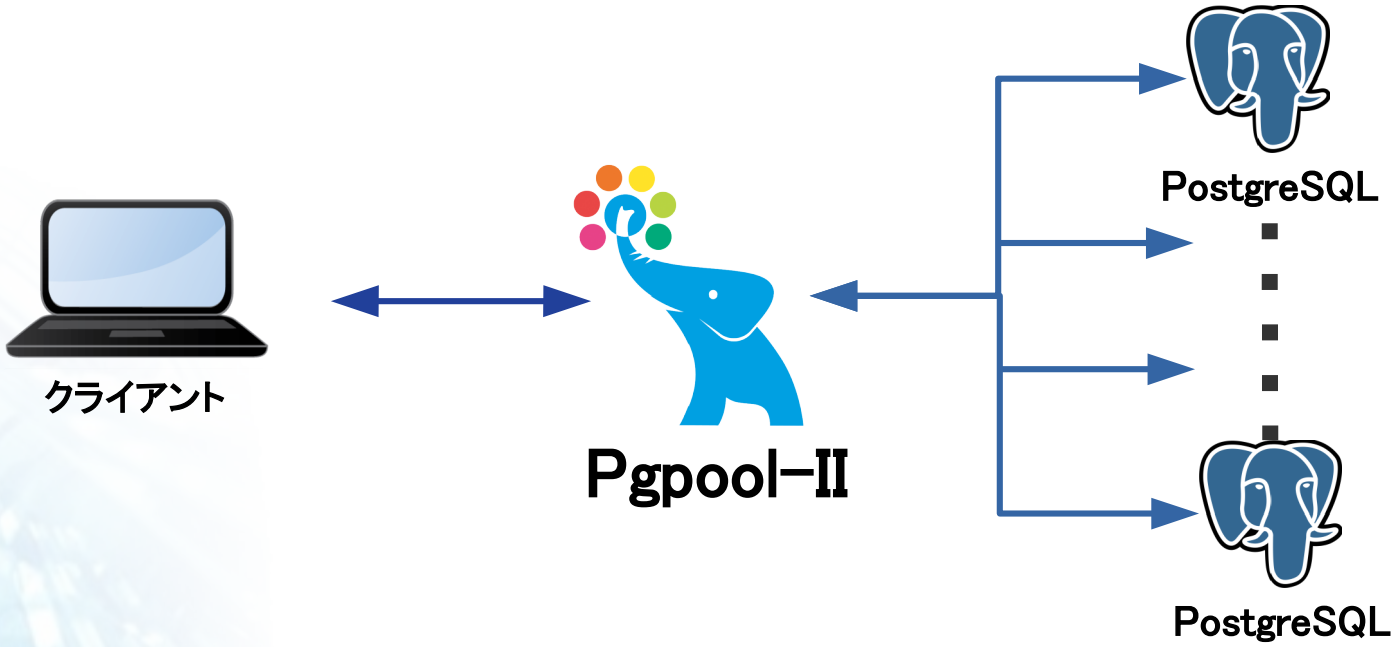


改善するためには

- Pgpool-II によりこれらの運用が容易になる
  - PostgreSQLの自動フェイルオーバ機能
    - 主系の故障を検知すると、待機系に自動的に切り替え
  - オンラインリカバリ機能
    - 故障したノードの復旧が可能
  - 負荷分散制御
    - PostgreSQLのプロトコル、クエリを理解
    - 柔軟な負荷分散が可能
  - レプリケーションの遅延監視
    - レプリケーションが遅延している場合、負荷分散を休止

Pgpool-II

- クライアント-PostgreSQL間で動作するミドルウェア
  - プロキシサーバのように振る舞い、クライアントからは1台のPostgreSQLのように見える



Pgpool-II

- 様々な機能を提供
  - コネクションプーリング
  - **自動フェイルオーバ**
  - **オンラインリカバリ**
  - **負荷分散**
  - インメモリクエリキャッシュ
  - watchdog(Pgpool-II自体のHA構成)

Pgpool-II 負荷分散

- 参照クエリをクラスタ内のPostgreSQLノードで分散
  - load_balance_mode = on で有効
  - セッションごとに負荷分散先のノードを決定
  - セッション内では、(基本的に)選択されたノードでクエリを実行
- 負荷分散の比率を設定可能
  - backend_weightN パラメータ(Nはノードの番号)
    (例)参照クエリを特定ノードに集中させる

    backend_weight0 = 0.5

    backend_weight1 = 1.0

    backend_weight2 = 1.0

Pgpool-II 負荷分散

- 負荷分散の詳細な条件を設定可能
  - 更新を含む/含まない関数を指定
    - black_function_list/white_function_list
    - 正規表現も利用可能
      例）
      black_function_list = ‘insert_func, delete_.*_func’
  - 負荷分散しないクエリを指定
    - black_query_pattern_list
    - 正規表現も利用可能

Pgpool-II 負荷分散

- データベース名による負荷分散の比率の指定
  - database_redirect_preference_list
- アプリケーション名による負荷分散の比率の指定
  - app_name_redirect_preference_list
- 更新クエリ後の負荷分散
  - disable_load_balance_on_write
    - off: 更新クエリ後も参照クエリは負荷分散
    - **transaction**: 明示的トランザクション内で更新クエリ後は負荷分散しない
    - trans_transaction: 更新クエリ後は、セッションが終了するまで明示的トランザクション内では負荷分散しない
    - always: 更新クエリ後は、セッションが終了するまで負荷分散しない

Pgpool-II 負荷分散

- クエリレベルの負荷分散機能(**4.1 新機能**)
  - statement_level_load_balance
  - セッション毎ではなく、クエリ毎に負荷分散ノードを選出
  - 1セッションで複数トランザクションが長時間実行されるようなアプリケーションで有効

Pgpool-II 負荷分散

- 負荷分散の制限
  - SELECTクエリである
  - 一時テーブル/unloggedテーブルを使用していない
  - システムカタログを使用していない
  - トランザクション内の場合、更新クエリが実行されていない
  - FOR UPDATE/FOR SHAREが指定されていない

  etc...

Pgpool-II 自動フェイルオーバ

- フェイルオーバの契機
  - ノードへの接続、および接続中にエラーが発生した場合
    - failover_on_backend_error = on の場合
  - ヘルスチェックでダウンと判定された場合
    - ヘルスチェック: 各PostgreSQLノードの状態を監視するプロセス
    - 実行間隔、最大リトライ回数などを設定可能
- failover_commandで実行する処理を設定
  - 実用レベルのサンプルはドキュメントで紹介
    - 主な内容は待機系のpromoteなど
  - 障害ノード/新しいマスターノードの情報は引数として取得可能

Pgpool-II 自動フェイルオーバ

- failover_commandのスクリプトのサンプル

```
#!/bin/bash
# This script is run by failover_command.

PGHOME=/usr/pgsql-11

## Test passwrodless SSH
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${NEW_MASTER_NODE_HOST} -i ~/.ssh/id_rsa_pgpool ls /tmp > /dev/null

if [ $? -ne 0 ]; then
    exit 1
fi

## If Standby node is down, skip failover.
if [ $FAILED_NODE_ID -ne $OLD_PRIMARY_NODE_ID ]; then
    ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@$OLD_PRIMARY_NODE_HOST -i ~/.ssh/id_rsa_pgpool "
        ${PGHOME}/bin/psql -p $OLD_PRIMARY_NODE_PORT -c ¥"SELECT pg_drop_replication_slot('${FAILED_NODE_HOST}')¥"
    "


    if [ $? -ne 0 ]; then
        exit 1
    fi

    exit 0
fi

## Promote Standby node.
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null ¥
    postgres@${NEW_MASTER_NODE_HOST} -i ~/.ssh/id_rsa_pgpool ${PGHOME}/bin/pg_ctl -D ${NEW_MASTER_NODE_PGDATA} -w promote

if [ $? -ne 0 ]; then
    exit 1
fi

exit 0
```

Pgpool-II オンラインリカバリ

- pcp_recovery_node コマンド

  例: pcp_recovery_node -h ホスト名 -p ポート番号 -n ノード番号

  - 停止ノードをオンラインリカバリし、待機系として組み込む
  - SR環境では、recovery_1st_stage_commandを設定する
    - pg_basebackupによる主系のオンラインバックアップの取得
    - 待機系の設定を追加
      - ～PostgreSQL11
        - recovery.conf の作成
      - PostgreSQL12～
        - postgresql.conf の変更
        - standby.signal の作成

Pgpool-II オンラインリカバリ

- 自動フェイルオーバ後のオンラインリカバリ
  - 故障した旧主系のノードをオンラインリカバリし、待機系として復帰させる
    →自動でHA構成を維持
  - follow_master_commandで実現可能
- follow_master_command
  - 主系がフェイルオーバした後に実行するコマンドを指定
  - pcp_recovery_nodeを呼び出すように設定しておくことで、フェイルオーバからオンラインリカバリを自動化可能

自動フェイルバック機能

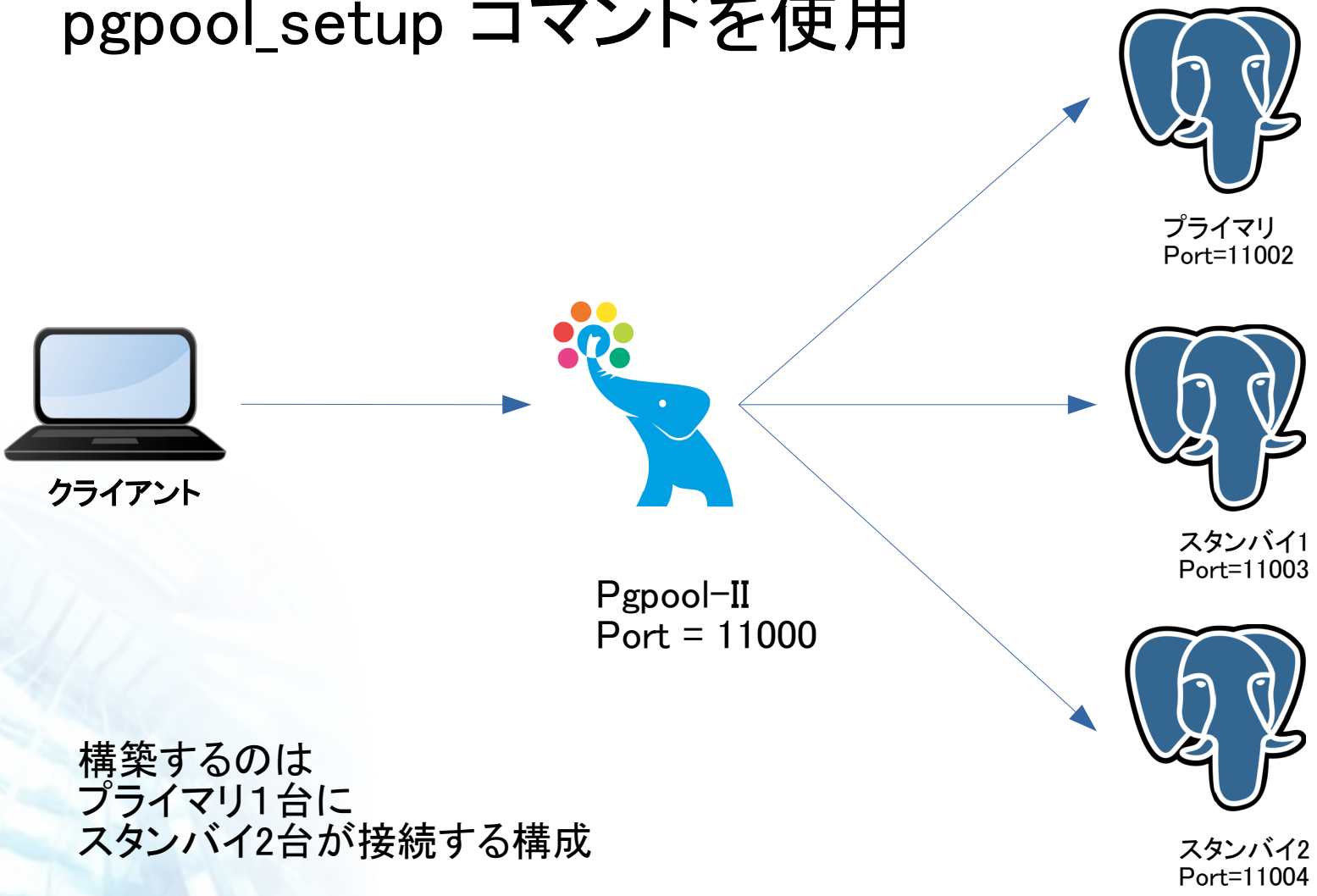
- ダウン判定となった待機系ノードが正常な状態に復帰した場合に自動で再組み込みできる(**4.1新機能**)
  - ネットワークやサーバの一時的な障害により待機系がダウンと判定された場合に有効
  - SR構成であれば、一時的に同期が遅れても自動で回復する
- auto_failback
  - 自動フェイルバックを有効化する
- auto_failback_interval
  - 自動フェイルバックの最小実行間隔を指定
  - ネットワークの不具合などで頻繁に自動フェイルオーバ/フェイルバックが繰り返されるのを防ぐことができます

デモタイム

- シナリオ(自動フェイルオーバ)
  - プライマリサーバをシャットダウン
  - Pgpool-IIがそれを検知し、スタンバイ1を昇格、新プライマリにする
  - 続いてフォローマスターコマンドが起動され、スタンバイ2が新プライマリ(旧スタンバイ1)に同期、追随する
- シナリオ2(自動フェイルバック)
  - その後、スタンバイ2をシャットダウン
  - Pgpool-IIがそれを検知し、スタンバイ2を切り離す
  - スタンバイ2を再起動する
  - Pgpool-IIがスタンバイ2が正常に起動していることを検知し、再組み込みする

Pgpool-II付属ツールを使って実験環境構築

pgpool_setup コマンドを使用



構築するのは
プライマリ1台に
スタンバイ2台が接続する構成

デモンストレーション

```
$ pgpool_setup -n 3
PostgreSQL major version: 115
Starting set up in streaming replication mode
creating startall and shutdownall

[中略]


recovery node 1...pcp_recovery_node -- Command Successful
done.
recovery node 2...pcp_recovery_node -- Command Successful
done.
creating follow master script
 node_id | hostname | port  | status | lb_weight |  role   | select_cnt | load_balance_node | replication_delay | replication_state |
replication_sync_state | last_status_change
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------+-------------------
+------------------------+---------------------
 0       | /tmp     | 11002 | up     | 0.333333  | primary | 0          | false             | 0                 |                   |
                  | 2019-09-12 20:05:40
 1       | /tmp     | 11003 | up     | 0.333333  | standby | 0          | false             | 0                 | streaming         |
async                     | 2019-09-12 20:05:46
 2       | /tmp     | 11004 | up     | 0.333333  | standby | 0          | true              | 0                 | streaming         |
async                     | 2019-09-12 20:05:51
(3 rows)

shutdown all

pgpool-II setting for streaming replication mode is done.
To start the whole system, use /work/test/demo/startall.
To shutdown the whole system, use /work/test/demo/shutdownall.
pcp command user name is "hoshiai", password is "hoshiai".
Each PostgreSQL, pgpool-II and pcp port is as follows:
#1 port is 11002
#2 port is 11003
#3 port is 11004
pgpool port is 11000
pcp port is 11001
The info above is in README.port.
```

まとめ

- クラスタ構成の用途
  - 高可用性と負荷分散
- PostgreSQLにおけるクラスタ機能
  - ストリーミングレプリケーション機能
  - 課題
    - ファイルオーバ等の手動作業
    - 負荷分散の制御が外部で必要
- Pgpool-IIによる課題の解決

Pgpool-II 参考情報

- ドキュメント

  http://www.pgpool.net/docs/latest/ja/html/
- Wiki

  https://www.pgpool.net/mediawiki/jp/index.php/メインページ
- メーリングリスト(ML)

  https://pgpool.net/mediawiki/jp/index.php/Mailing_lists
- バグ報告

  https://www.pgpool.net/mantisbt/
- ソースコード(gitリポジトリ)

  https://git.postgresql.org/gitweb/?p=pgpool2.git;a=summary

Thank you!