PostgreSQLの高可用性と性能向上を実現！
Pgpool-IIの全貌と最新動向

2025/04/24

株式会社SRA OSS
彭 博 (ペン ボ)

自己紹介

ペン ボ

- 名前：彭 博 (Bo Peng)
  pengbo@sraoss.co.jp
- 所属：株式会社SRA OSS
  技術部　基盤技術グループ
- 職務：
  - OSS技術サポート、ミドルウェア構築、コンサルティング
  - PostgreSQLクラスタ管理ツールPgpool-II開発者

本セッションの流れ

- PostgreSQLクラスタに高可用性と負荷分散が求められる理由
- Pgpool-IIの概要および機能紹介
  - 自動フェイルオーバ、負荷分散、コネクションプーリングの仕組み
- Pgpool-II 4.6新機能紹介

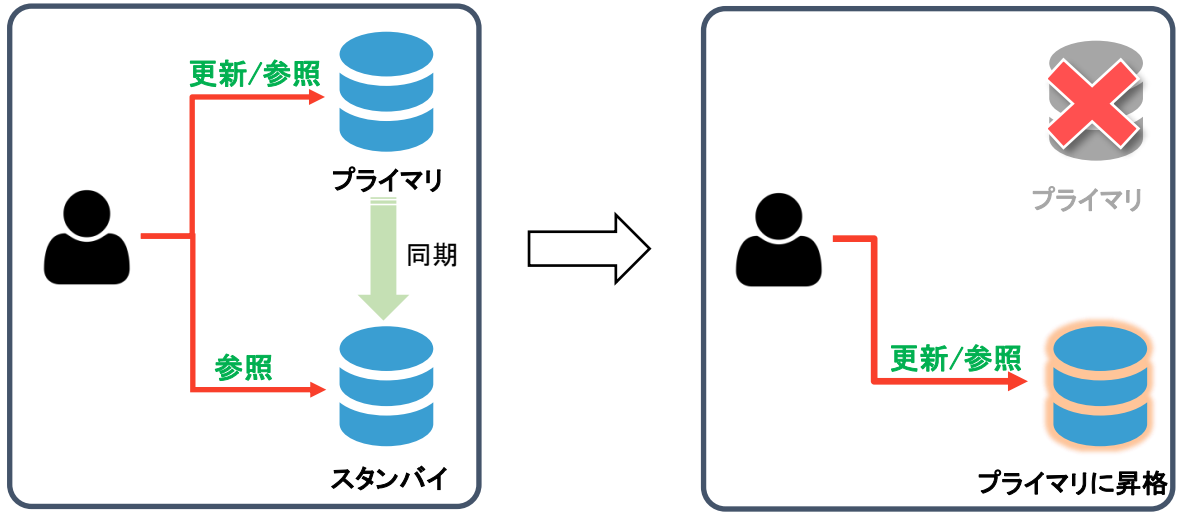
なぜクラスタ構成は必要なの？

高可用性

- データの冗長化
- 稼働系に障害が発生した際に、待機系に切り替えてサービスを継続させる
- ダウンタイムを最小限に抑える

参照負荷分散

- レプリケーションされている複数台のPostgreSQLで、参照クエリをいずれかのPostgreSQLに振り分ける
- プライマリの負荷を下げる
- 全体性能向上



PostgreSQL本体にはHAや負荷分散の機能は搭載されていない

自動フェイルオーバ/フェイルバックの仕組みがない

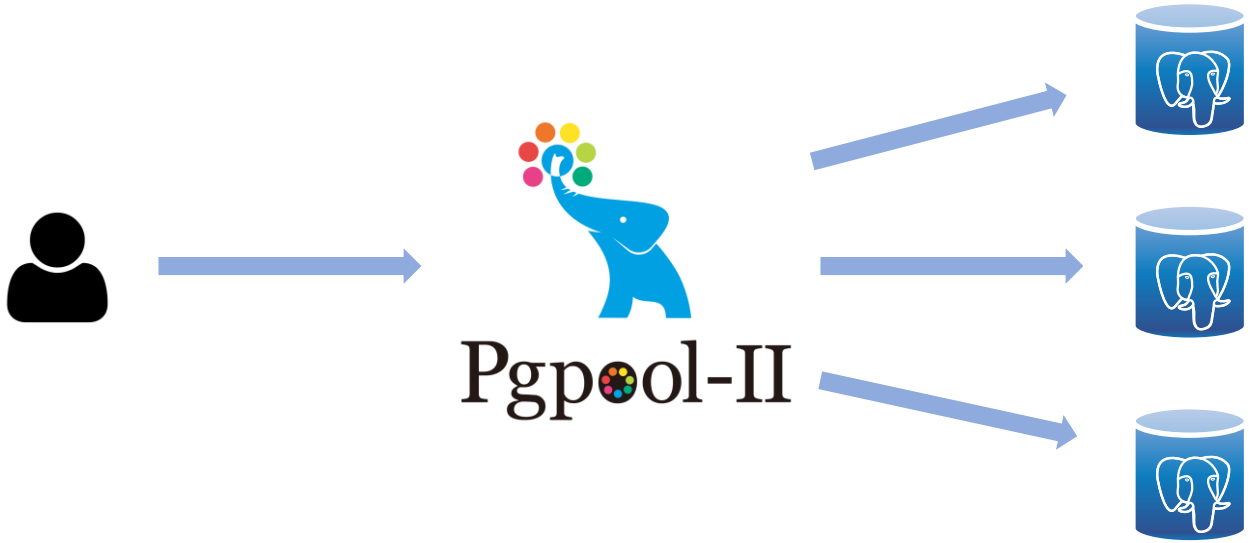
- 障害が発生した際に手動での切り替えが必要
  - 障害の検知
  - 待機系の昇格
  - 残りの待機系の同期元の変更
  - アプリケーション側でデータベース接続情報の変更
- 障害ノードの復旧
  - 障害が発生したノードの復旧とクラスタへの再組み込み

更新クエリと参照クエリの振り分け機能は提供されていない

- プライマリは更新/参照クエリの両方を処理可能
- スタンバイは参照クエリのみ処理可能
- アプリケーション側でクエリを振り分ける処理を実装する必要がある

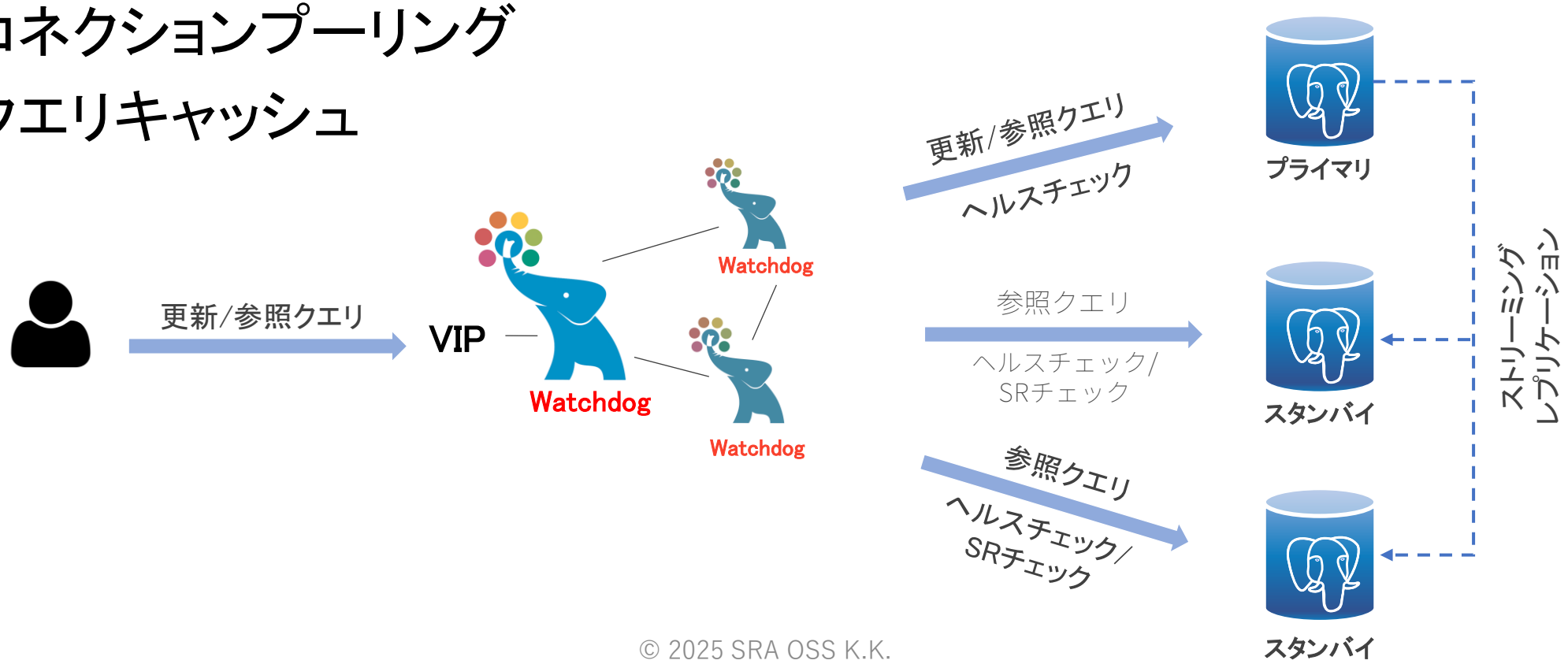
Pgpool-IIを利用することでこれらの課題を解決できる

- Pgpool-IIはクライアントとPostgreSQLの間で動作するミドルウェア
- Pgpool Global Development Groupによって開発・メンテナンスされているOSS
- PostgreSQL単体では実現できない自動フェイルオーバ、負荷分散、コネクションプーリングなどの機能を提供
- ユーザは複数PostgreSQLサーバを意識せず、1台のように見える



Pgpool-IIの主な機能

- 参照クエリの負荷分散
- 自動フェイルオーバー
- Watchdog
- コネクションプーリング
- クエリキャッシュ



自動フェイルオーバ

- Pgpool-IIは定期的に各PostgreSQLの状態を監視する
- PostgreSQLの障害を検知すると、フェイルオーバを実行する

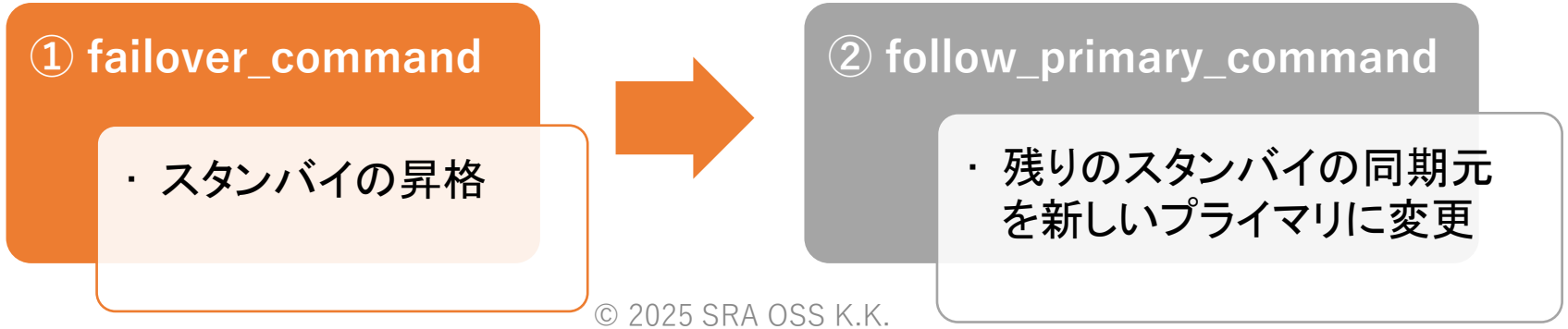
自動フェイルオーバ

フェイルオーバの契機

- 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
- PostgreSQLへの接続時、および接続後にネットワーク通信エラーが発生した場合
  - failover_on_backend_error = onの場合

フェイルオーバ処理

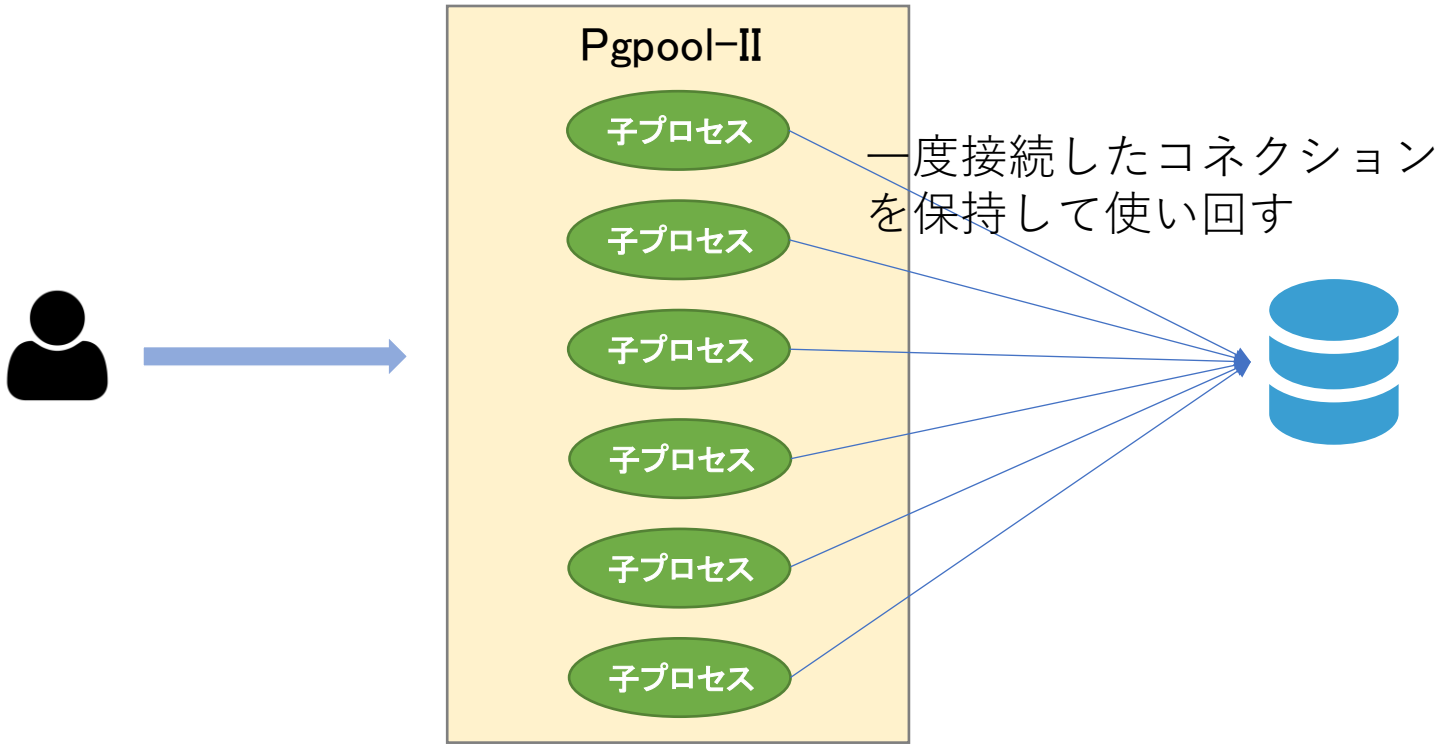
- スタンバイがダウンした場合
  - ダウンしたノードの状態を変更（up->down）
- プライマリがダウンした場合
  - ダウンしたノードの状態を変更し（up->down）、以下のパラメータに設定されているスクリプトを実行



コネクションプーリングによるDB接続高速化

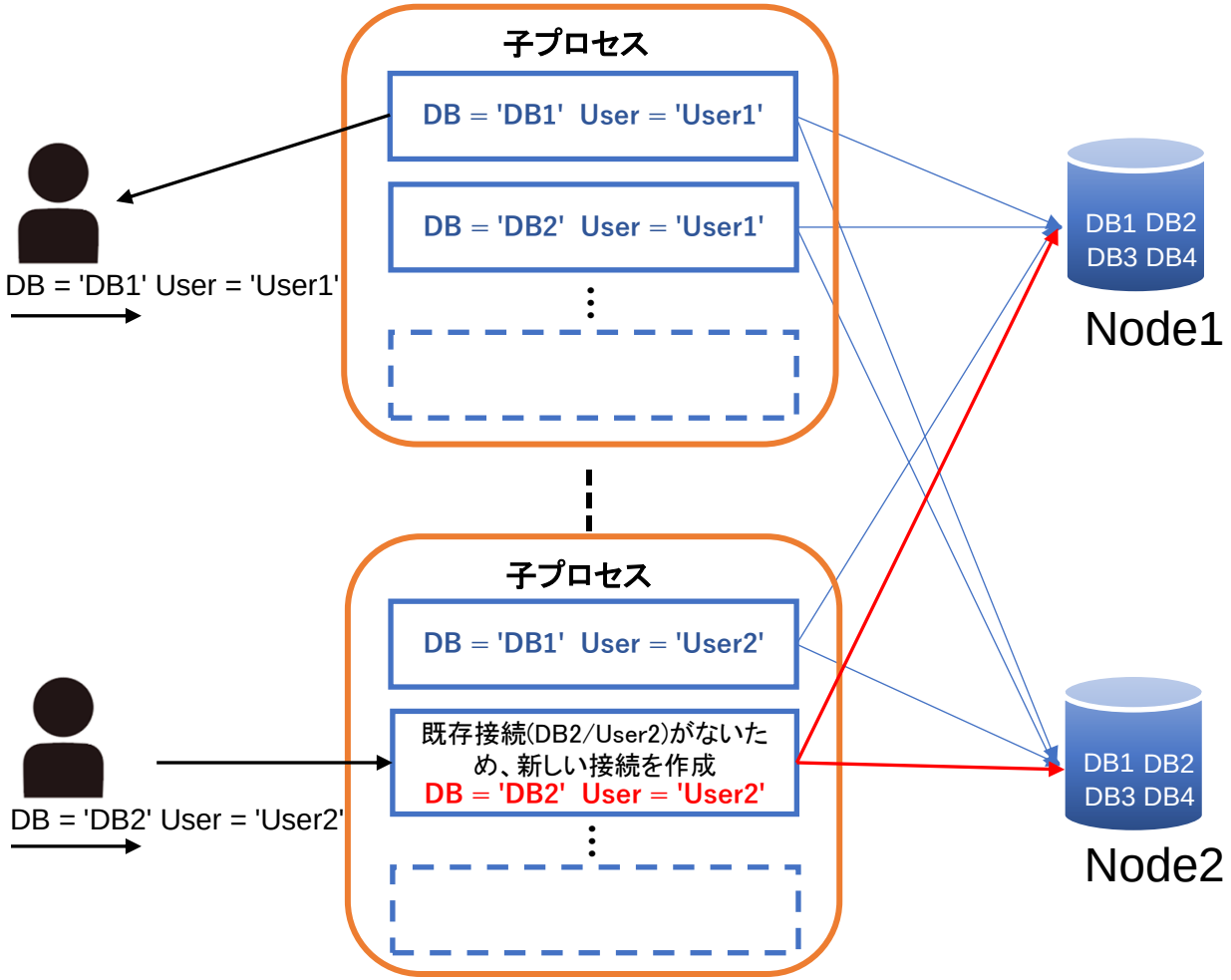
なぜコネクションプーリング？

- データベースにアクセスするたびに、接続処理にオーバーヘッドが発生する
- この処理を毎回繰り返すと、レスポンスが遅くなる
- 接続を保持し、使い回すことで、接続確立のオーバーヘッドを削減できる



Pgpool-IIコネクションプーリングの仕組み

1. 起動時に、Pgpool-IIはnum_init_children個の子プロセスをプリフォークし、各子プロセスはmax_poolの設定値まで接続をキャッシュする
2. Pgpool-IIはクライアントからの接続要求を待ち受ける
3. クライアントからの接続要求が来ると、1つの子プロセスが接続を受け付ける
4. この子プロセスは、保持している接続内に、接続情報（データベース/ユーザ）の一致するものがあるかどうかを探す。見つかったら再利用する。
5. 見つからなかった場合、新しい接続を作成し、それをプールに登録する。プールに空きスロットがない場合、最も古い接続をクローズし、新しい接続を作成する。
6. クライアントが接続を終了する。Pgpool-IIはクライアントへの接続をクローズするが、PostgreSQLへの接続を保持する



負荷分散(参照クエリ振り分け)

負荷分散

- SQLパーサを搭載しており、クエリを解析できる
  - それでも足りない情報は、Pgpool-IIが自動的にPostgreSQLに問い合わせる (例えば、関数の場合)
- 更新クエリはプライマリに送信、参照クエリは複数のPostgreSQLノード間で分散

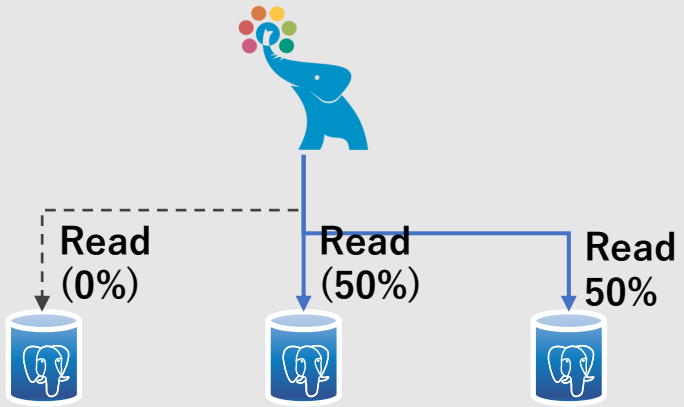
負荷分散モード

- セッションレベル (デフォルト)
- ステートメントレベル

負荷分散の比率設定

例えば、プライマリを更新処理専用にし、すべての参照クエリをスタンバイに振り分けたい場合

```
backend_weight0 = 0
backend_weight1 = 1
backend_weight2 = 1
```



より柔軟な負荷分散設定①

特定のクエリを負荷分散させたくない

① 文字列で指定

- primary_routing_query_pattern_listに指定されたSQLパターンを含むクエリは負荷分散されない

```
primary_routing_query_pattern_list = '.*my_table*.'
```

my_tableという文字列を含むクエリがプライマリノードに送信される

② コメントを付与

- 先頭にコメントが付与されたクエリは負荷分散されない

```
/*NO LOAD BALANCE*/SELECT * FROM t1
```

より柔軟な負荷分散設定②

database_redirect_preference_list

- データベース名でクエリの振り分け先を決定

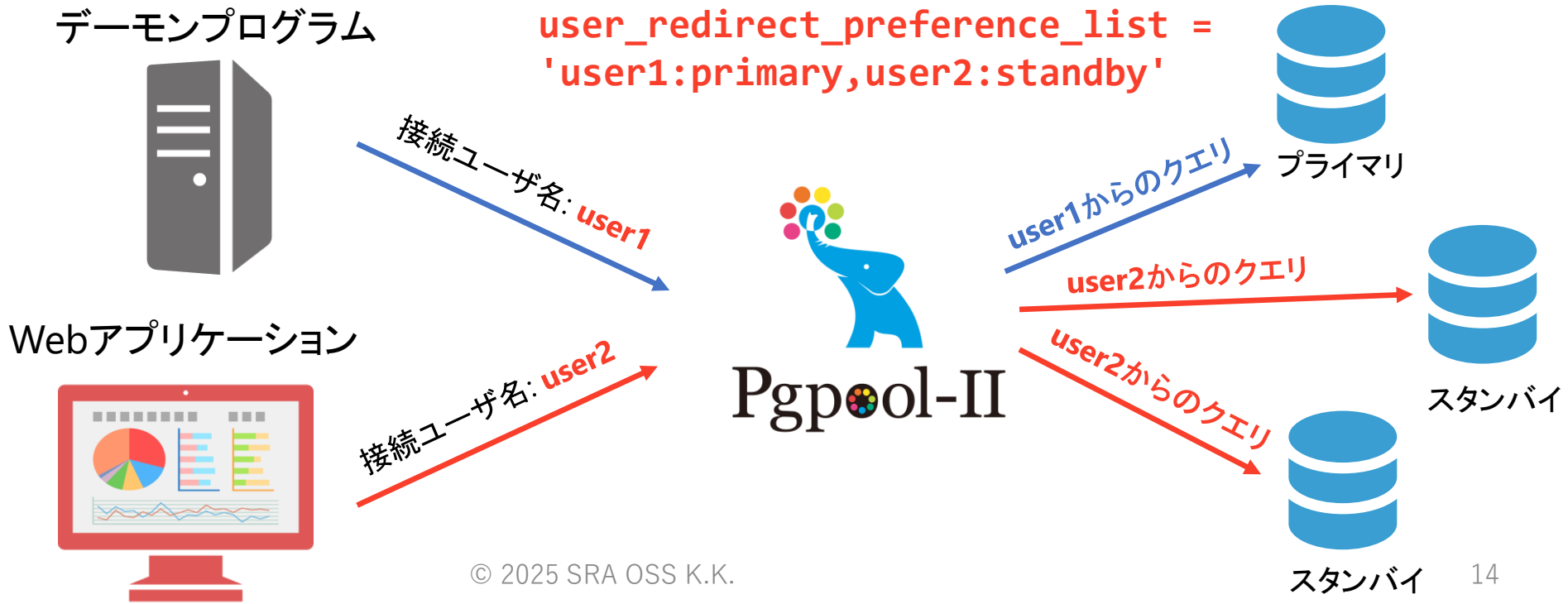
app_name_redirect_preference_list

- アプリケーション名でクエリの振り分け先を決定

user_redirect_preference_list

- ユーザ名でクエリの振り分け先を決定

ユースケース



Pgpool-II 4.6新機能紹介

Pgpool-II 4.6の新機能ハイライト

- クエリキャッシュの機能強化
- ロギングコレクターの機能強化
- バックエンドからのメッセージをログ可能に
- WatchdogのパラメータのIPv6対応

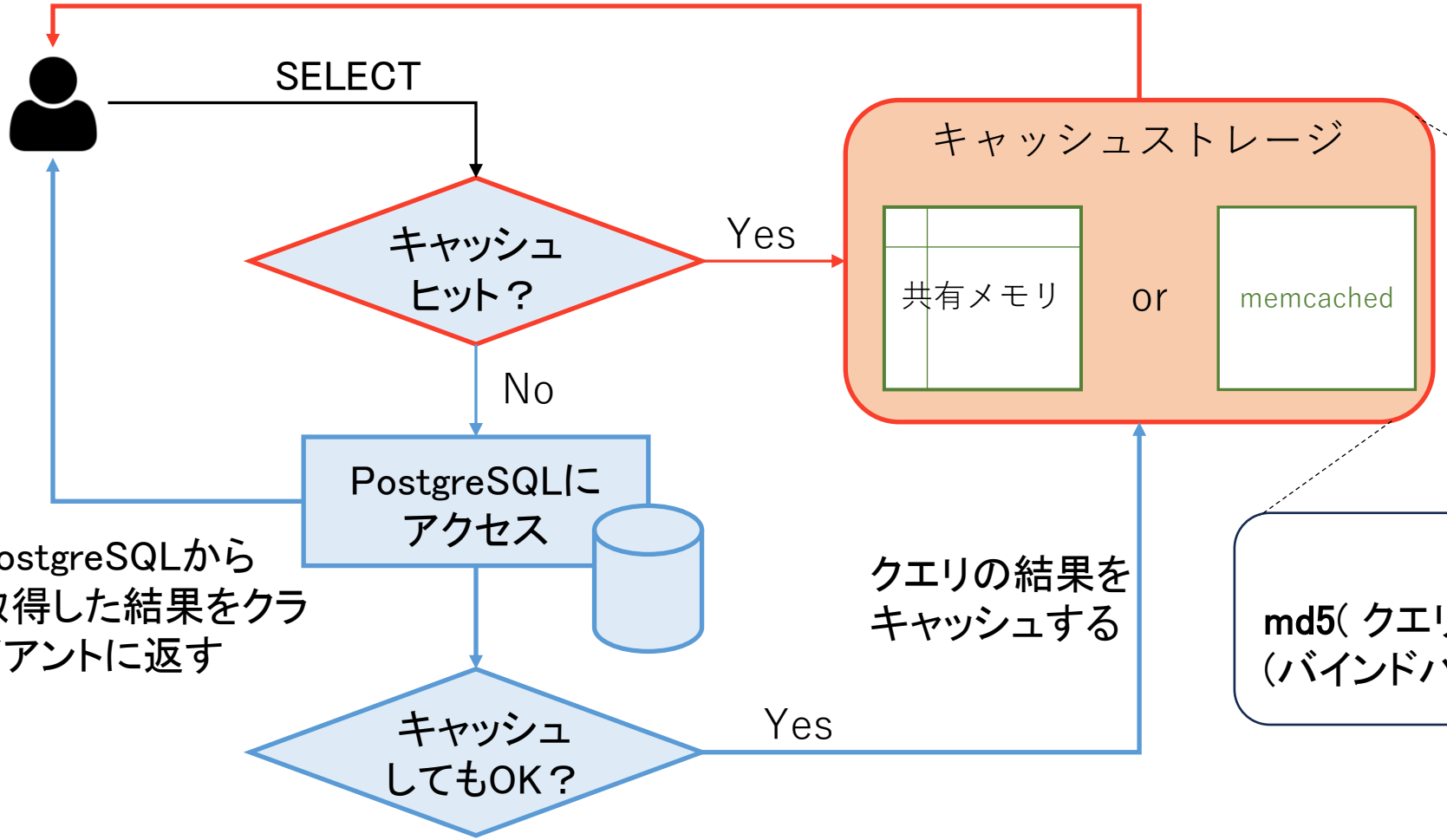
- PostgreSQL 17のraw parserの移植

クエリキャッシュとは

- SELECT文とその検索結果をキャッシュに保存→**SELECT性能向上！**
  - 次回以降、同じクエリが来たらPostgreSQLにアクセスせずに、キャッシュされている結果を直接返す
- キャッシュストレージタイプ
  - 共有メモリ
    - アクセスが高速だが、サイズに制限あり
  - memcached
    - 容量の設定が比較的自由だが、ネットワーク通信のオーバーヘッドあり

クエリキャッシュの仕組み

キャッシュから取得した結果をクライアントに返す ⟶ **速い！**



クエリをそのままキャッシュするのではなく、md5ハッシュを生成して、ハッシュ値をキャッシュする

クエリハッシュ

md5(クエリ文字列+データベース名+ユーザ名+(バインドパラメータ))

クエリキャッシュを強制的に作成する機能を追加

- クエリキャッシュは通常、自動的に作成される
- Pgpool-IIは安全性の観点から、immutableでない関数を含むクエリはキャッシュしない
  - たとえばtimestamp with timezoneを結果や引数で使う関数

```
SELECT now();
```

<= 通常はキャッシュしない

- それでも、内容を理解した上でキャッシュしたい場合もある
- 4.6からは、SQLコメント **/*FORCE QUERY CACHE*/** を付けることで、キャッシュを強制的に作成できるようになった

```
/*FORCE QUERY CACHE*/SELECT now();
```

<= 強制的にキャッシュする

クエリキャッシュが削除される契機

- キャッシュ対象のテーブルが更新されたとき
- キャッシュの有効期限が切れたとき (memqcache_expire)
- **PGPOOL SET CACHE DELETE** コマンドで対象クエリを削除したとき 4.6

```
PGPOOL SET CACHE DELETE 'QUERY';
```

- **pcp_invalidate_query_cache** コマンドを実行したとき 4.6
  - すべてのクエリキャッシュを削除
  - 今までは、クエリキャッシュの削除はPgpool-IIの再起動が必要だった

```
pcp_invalidate_query_cache -h localhost -p 9898 -U pgpool -W
```

ロギングコレクターの機能強化

- Pgpool-IIのロギングコレクターは、PostgreSQLから移植したもの
- PostgreSQLのロギングコレクターとほぼ同じ機能を持つ

```
logging_collector = on
```

関連パラメータ

パラメータ名	説明
log_directory	ログファイルの出力先ディレクトリ
log_filename	ログファイル名の形式
log_file_mode	ログファイルのパーミッション
log_rotation_age	ログファイルのローテーション間隔
log_rotation_size	ログファイルの最大サイズ
log_truncate_on_rotation	ログローテーション時に、既存の同名ファイルがある場合に上書きするかどうか

ロギングコレクターの機能強化

- これまでは、外部のログローテーションツールによってログファイルがローテートされた後、Pgpool-IIが新しいファイルに書き込むように切り替えるため、ロギングコレクターにシグナルを送る必要があった

```
kill -SIGUSR1 <pid of PgpoolLogger>
```

- 4.6からは、新しいPCPコマンドpcp_log_rotateでローテーションが可能に

```
pcp_log_rotate -h localhost -p 9898 -U pgpool -W
```

ロギングコレクターの機能強化

pcp_log_rotate のユースケース

```
/var/log/pgpool_log/pgpool.log {
    weekly
    rotate 53
    compress
    delaycompress
    missingok
    notifempty
    create 0600 postgres postgres
    postrotate
        /usr/bin/pcp_log_rotate -h localhost -p 9898 -U pgpool >/dev/null 2>&1 || true
    endscript
}
```

logrotate によってログファイルがローテートされた後、
Pgpool-II に新しいログファイルへの切り替えを指示する

ロギングコレクターの機能強化

- これまでは、ロギングコレクター関連の設定パラメータを変更する場合、Pgpool-II の再起動が必要だった

```
log_directory
log_filename
log_file_mode
log_rotation_age
log_rotation_size
log_truncate_on_rotation
```

- 4.6からは、**pgpool reload** だけで設定変更が反映されるようになった

```
pgpool reload
```

ログ設定関連パラメータ①

ログ設定関連パラメータ

- log_statement

実行したクエリが出力される

```
LOG:  statement: select  * from t1;
LOG:  statement: select  * from t2;
```

- log_per_node_statement

```
LOG:  DB node id: 0 backend pid: 14020 statement: select * from t1
LOG:  DB node id: 0 backend pid: 14020 statement: DISCARD ALL
LOG:  DB node id: 1 backend pid: 14031 statement: select * from t1
LOG:  DB node id: 0 backend pid: 14030 statement: DISCARD ALL
LOG:  DB node id: 1 backend pid: 14031 statement: DISCARD ALL
```

クエリを実行したバックエンドノードIDが出力される。
負荷分散の動作確認時に便利

ログ設定関連パラメータ②

ログ設定関連パラメータ

- notice_per_node_statement

  4.5以降

- クエリを実行したバックエンドノードIDとクエリがプロンプトに表示されるため、ログを確認する必要がない
- 従来のようにlog_per_node_statementを有効にしてログをgrepする必要がなくなり、テストケースの追加も容易になる

```
$ psql -h 127.0.0.1 -p 11000 test -c "select  * from t1;"
NOTICE:  DB node id: 0 statement: select  * from t1;
 id |             ts
----+----------------------------
  1 | 2025-04-04 15:00:04.736673
(1 行)

$ psql -h 127.0.0.1 -p 11000 test -c "select  * from t1;"
NOTICE:  DB node id: 1 statement: select  * from t1;
 id |             ts
----+----------------------------
  1 | 2025-04-04 15:00:04.736673
(1 行)
```

ログ設定関連パラメータ③

ログ設定関連パラメータ

- log_client_messages　4.0以降

デバッグメッセージなしで、クライアントメッセージのみをログ出力可能

```
LOG:  Parse message from frontend.
DETAIL:  statement: "S1", query: "select * from t1"
LOG:  Bind message from frontend.
DETAIL:  portal: "", statement: "S1"
LOG:  Execute message from frontend.
DETAIL:  portal: ""
LOG:  Close message from frontend.
DETAIL:  statement: "S1"
LOG:  Sync message from frontend.
LOG:  Terminate message from frontend.
```

新しい設定パラメータ log_backend_messages

- これまではバックエンドからのメッセージ種別を特定する仕組みがない。それを知るには log_min_messages を使ってデバッグログを出力する必要があった
- ただし、デバッグを有効にするとログ行数が非常に多くなる
- 4.6では、新しいパラメータ **log_backend_messages** を追加し、どのバックエンドからどの種別のメッセージが返されたかを記録できるようになった

```
LOG:  AuthenticationRequest message from backend 0
LOG:  AuthenticationRequest message from backend 1
LOG:  ParameterStatus message from backend 0
LOG:  ParameterStatus message from backend 1
LOG:  last ParameterStatus message from backend 0 repeated 13 times
LOG:  BackendKeyData message from backend 0
LOG:  last ParameterStatus message from backend 1 repeated 13 times
LOG:  BackendKeyData message from backend 1
LOG:  ReadyForQuery message from backend 0
LOG:  ReadyForQuery message from backend 1
LOG:  ParseComplete message from backend 1
LOG:  BindComplete message from backend 1
LOG:  DataRow message from backend 1
LOG:  CommandComplete message from backend 1
LOG:  CloseComplete message from backend 0
LOG:  CloseComplete message from backend 1
LOG:  ReadyForQuery message from backend 0
LOG:  ReadyForQuery message from backend 1
LOG:  CommandComplete message from backend 0
LOG:  CommandComplete message from backend 1
LOG:  ReadyForQuery message from backend 0
LOG:  ReadyForQuery message from backend 1
```

Watchdogのパラメータで IPv6が利用可能に

- これまでは、以下のコンポーネントでIPv6アドレスの使用が可能
  - PostgreSQLバックエンドノードのアドレス
  - Pgpool-II本体のlistenアドレス
  - PCPプロセスのlistenアドレス
- 一方、watchdogプロセスはIPv4またはUNIXドメインソケットのみ対応（IPv6非対応）
- 4.6からは、すべてのネットワーク関係のパラメータでIPv6が使えるようになった

PostgreSQL 17パーサの移植

- Pgpool-IIは読み取りクエリを振り分けするために、PostgreSQLのSQLパーサを移植している
- Pgpool-II 4.6では**PostgreSQL 17**のパーサを取り込んでいる

PostgreSQL 17パーサのおもな変更点

- MERGEでNOT MATCHED BY SOURCEとRETURNING句が使えるようになった
- 新しいCOPYオプションのON_ERROR ignoreとLOG_VERBOSITYが追加された
- すべての列に対して'*'を使ってCOPY FROMオプションのFORCE_NOT_NULLとFORCE_NULLが指定できるようになった

など

4.6へバージョンアップの注意点①

- 以下のパラメータのデフォルト値が'nobody'から''(空文字)に変更された
  - health_check_user
  - sr_check_user
  - recovery_user
  - wd_lifecheck_user

4.6ではそれらのパラメータに適切な値をセットする必要がある

4.6へバージョンアップの注意点②

- 文字列型の設定パラメータの先頭/末尾の空白を無視するようになった
  - たとえばunix_socket_directoriesやpcp_socket_dir

4.6へバージョンアップの注意点③

- これまでは、child_life_timeやchild_max_connectionsの設定によって以下のようなメッセージがLOGレベルで出力されていた
- 4.6では、これらのメッセージのログレベルがLOGからDEBUG1にダウングレードされた

```
LOG: reaper handler
LOG: reaper handler: exiting normally
```

4.6へバージョンアップの注意点④

- サンプルスクリプトの変更
  - ① follow_primary.sh.sample
    - pg_basebackupの処理を削除した (4.2ブランチまでバックポート)
    - pg_rewindが失敗した場合、状態を確認したうえで手動で復旧するのが最も安全な方法

```
# First try pg_rewind. If pg_rewind failed, run pg_basebackup.
    pg_rewind ...

# If pg_rewind failed, run pg_basebackup
    pg_basebackup ...
```

削除しました

  - ② アーカイブモードの無効化
    - 古いアーカイブを削除する処理が含まれていないので、場合によっては、ディスクフルになる可能性があった
    - アーカイブモードを有効化したい場合は、PostgreSQLの設定やスクリプトを手動で変更する必要がある

参考情報

- Pgpool-II 4.6 リリースノート
  - https://www.pgpool.net/docs/latest/ja/html/release-4-6-0.html (英語)
  - https://www.pgpool.net/docs/latest/en/html/release-4-6-0.html (日本語)
- Pgpool-II wiki
  - https://pgpool.net/mediawiki/index.php/Main_Page
- Pgpool-II ドキュメント
  - https://www.pgpool.net/docs/latest/en/html/ (英語)
  - https://www.pgpool.net/docs/latest/ja/html/ (日本語)
- バグ報告
  - https://github.com/pgpool/pgpool2/issues
- ML
  - https://pgpool.net/mediawiki/index.php/Mailing_lists

SRA OSS の Pgpool-II 技術サポート

▼ サポート内容

インストール、設定などの導入時の支援から、使い方や設計に関する質問、運用開始後のチューニングや障害対応まで幅広く対応します。

▼ 対象メニュー

「 PostgreSQL サポート＆保守サービス（ ゴールド or プラチナ ）」

→ PostgreSQL はもちろん、**Pgpool-II** を含むクラスタソフトウェア（Pacemaker/Corosync、DRBD）もサポートします。

▼ ポイント

PostgreSQL コミッター ＆ コントリビューター、Pgpool-II開発者が在籍しているため、ソースコードレベルで調査可能＝高品質なサービス提供をお約束いたします。

サービス詳細　https://www.sraoss.co.jp/prod_serv/support/pgsql-mainte/
お問合せ　https://www.sraoss.co.jp/contact.php

ご清聴ありがとうございました。