Pgpool-II内部情報セミナー

第1回

SRA OSS, Inc. 日本支社

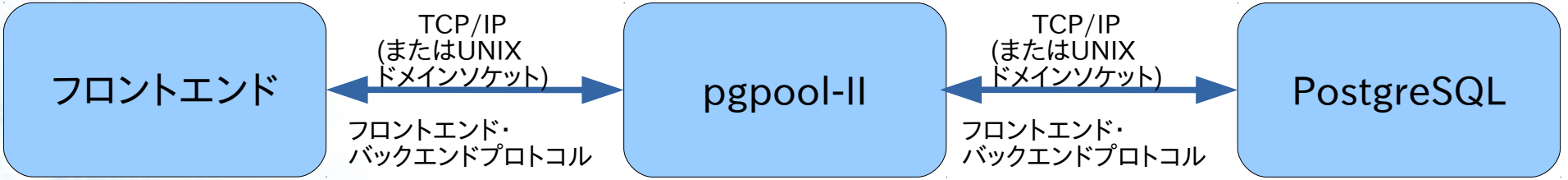
- 本講義は、Pgpool-II 3.5を前提にお話します

目次

- Pgpool-IIの利用形態
- Pgpool-IIの主な機能
- PostgreSQLの通信プロトコル
- Pgpool-IIのプロセス構造
- Pgpool-IIのソースツリー
- Pgpool-IIの処理概要
- コネクションプーリング
- フェイルオーバ
- ヘルスチェック
- 負荷分散
- オンラインリカバリ

Pgpool-IIの利用形態

- pgpool-IIはproxy型のシステム。フロントエンドとPostgreSQLの間に挟まる形で利用される
- フロントエンドからは普通のPostgreSQLに、PostgreSQLからは単なるフロントエンドに見える
- 通信プロトコルは、PostgreSQLで使用されている「フロントエンド・バックエンドプロトコル」をそのまま使用
  - クライアントを「フロントエンド」(Frontend)、サーバを「バックエンド」(Backend)と呼ぶ
  - フロントエンドはTCP/IP(またはUNIXドメインソケット)でバックエンドに接続し、SQLなどの要求を投げて結果を受け取る



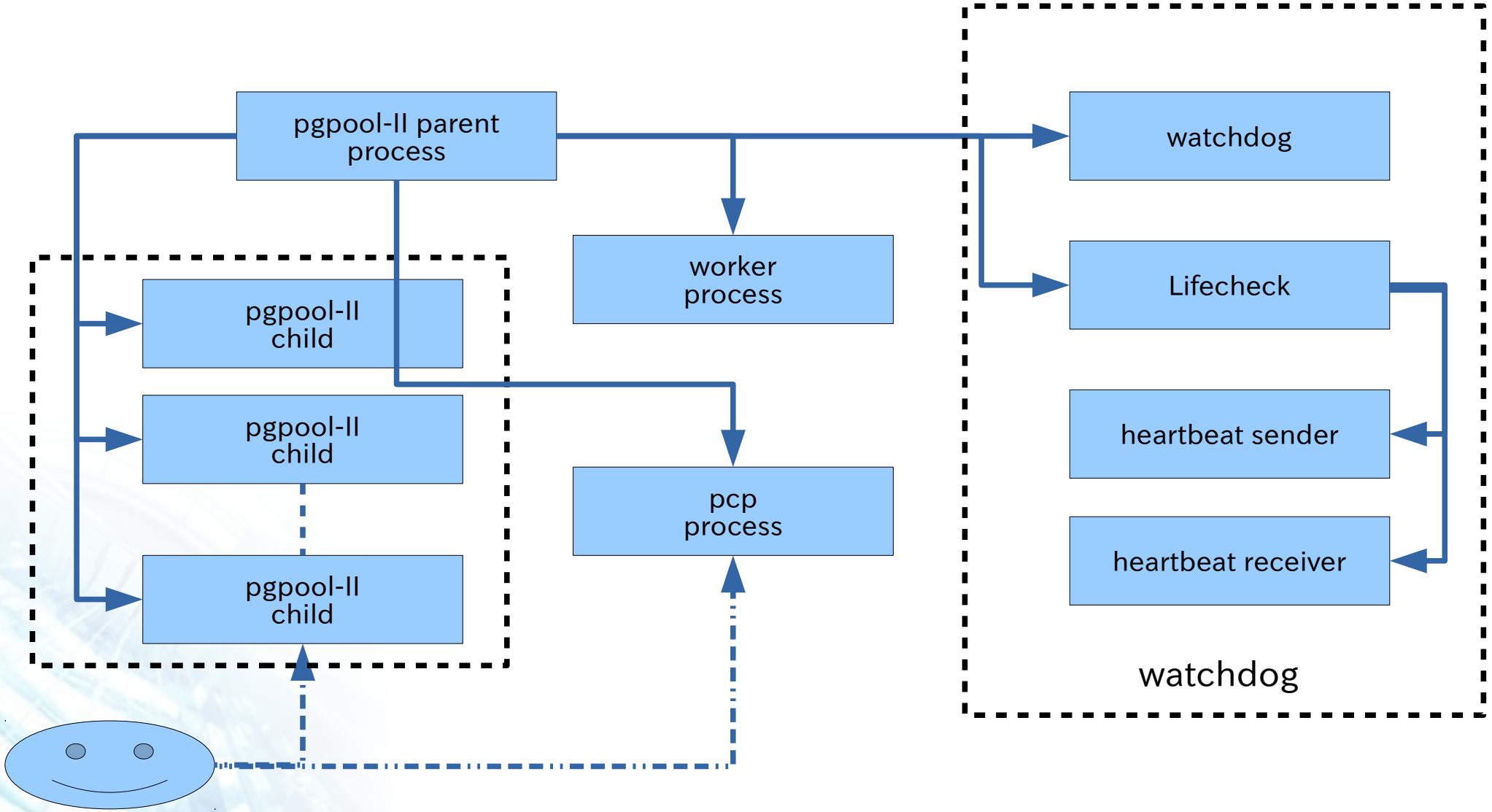
Pgpool-IIの主な機能

性能向上	コネクションプーリング 検索負荷分散 クエリキャッシュ
高可用性	自動フェイルオーバ フェイルオーバスクリプト フォローマスタスクリプト watchdog
クラスタ管理	オンラインリカバリ
クラスタとアプリケーションの親和性	クエリの自動振り分け

PostgreSQLの通信プロトコル

- PostgreSQLの通信プロトコルは過去1回大きく変更されており、PostgreSQL 7.3までのプロトコルを「バージョン2(Version 2:V2)」、PostgreSQL 7.4以降を「バージョン3(Version 3:V3)」と呼ぶ
- 本講義では、V3のみを扱う
  - V2を使うのかV3を使うのかはフロントエンドが決める
  - V2は今でも古いODBCドライバなどで使われている
  - PostgreSQLのバックエンドはV2を処理できるようになっている
  - V3プロコルトは更に「単純問い合わせ」(simple query protocol)と「拡張問い合わせ」(extended query protocol)の2系統に別れる。拡張問い合わせはJavaなどで使われる。

Pgpool-IIのプロセス構造



Pgpool-IIのソースツリー

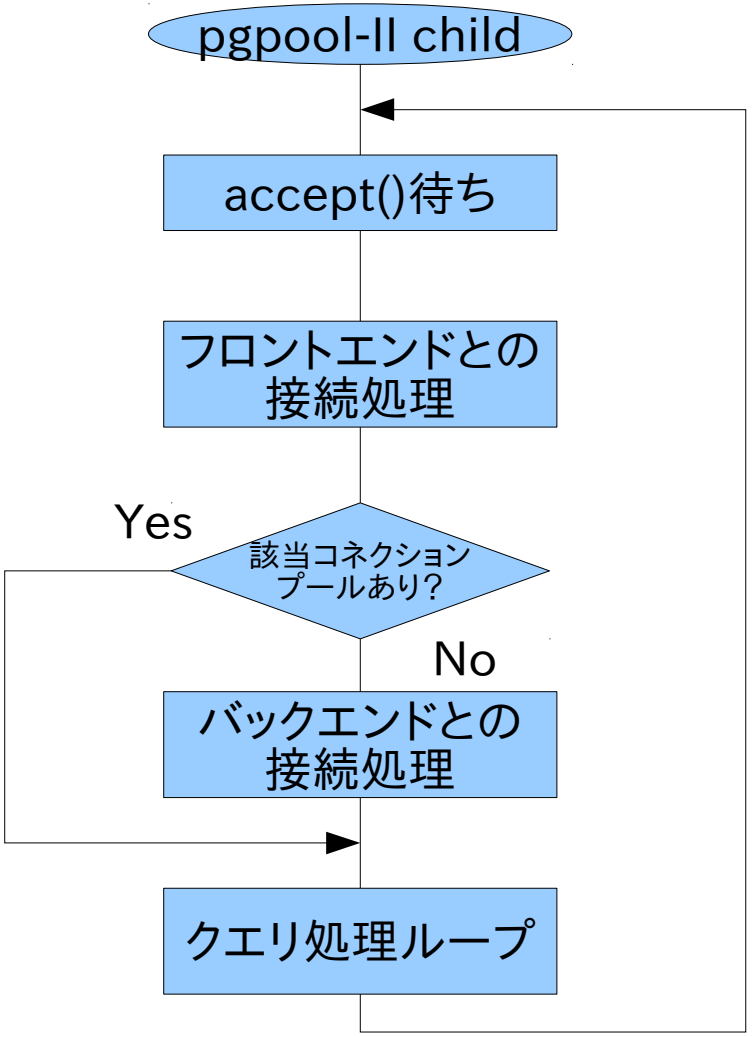
- トップレベルにはconfigureなどがあり、autoconfでビルドシステムを作っているので、以下のファイルを編集すること
  - configure.ac (version番号がこの中で指定されている)
  - Makefile.am
- ソースコードはsrc以下にある

src以下のソースツリー

ディレクトリ名	説明
auth	認証
config	設定ファイル
context	コンテキスト
include	includeファイル
libs	pcpライブラリ
main	メインプロセス
parser	SQLパーサ
pcp_con	pcpプロセス
protocol	プロトコル処理
query_cache	クエリキャッシュ
redhat	RPM関係
rewrite	SQL書き換え処理
sample	設定ファイルサンプルなど
sql	ユーザ定義関数
streaming_replication	ストリーミングレプリケーション用workerプロセス
test	テストスィート
tools	pcpコマンドツール
utils	各種ユーティリティルーチン
utils/mmgr	メモリーマネージャ
utils/error	エラー処理
watchdog	watchdog

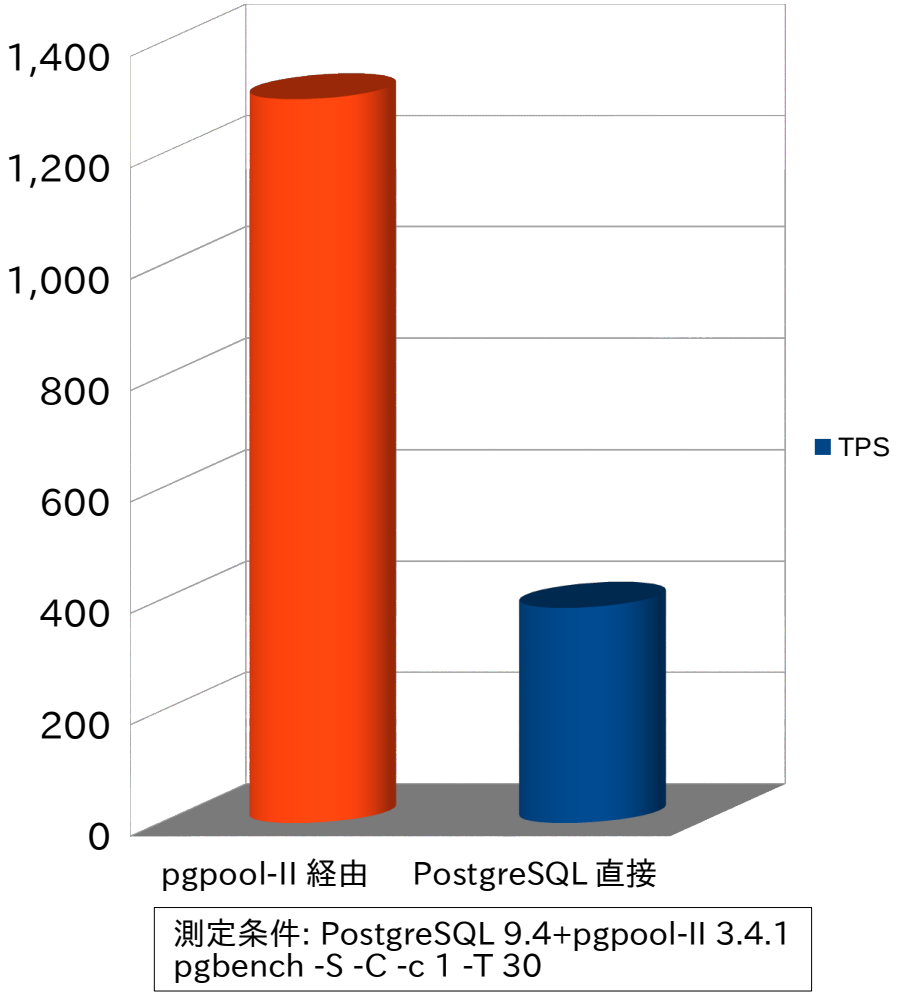
Pgpool-IIの処理概要

- pgpool-IIでは、親プロセスはlisten()を発行した状態で子プロセスをfork()する
- 子プロセスは、それぞれ個別にaccept()を発行し、フロントエンドからの接続要求があると、OSがどれか一つのpgpool-II子プロセスを選択して処理を渡す
- その子プロセスは、フロントエンドとの接続処理を認証を含めて行い、成功したらすべてのバックエンドと接続を行った後に、フロントエンドからのクエリを受つけてバックエンドに渡す無限ループに入る
- フロントエンドとの接続が切れるか、終了メッセージを受け取ると無限ループを抜けて再びaccept()待ちに入る

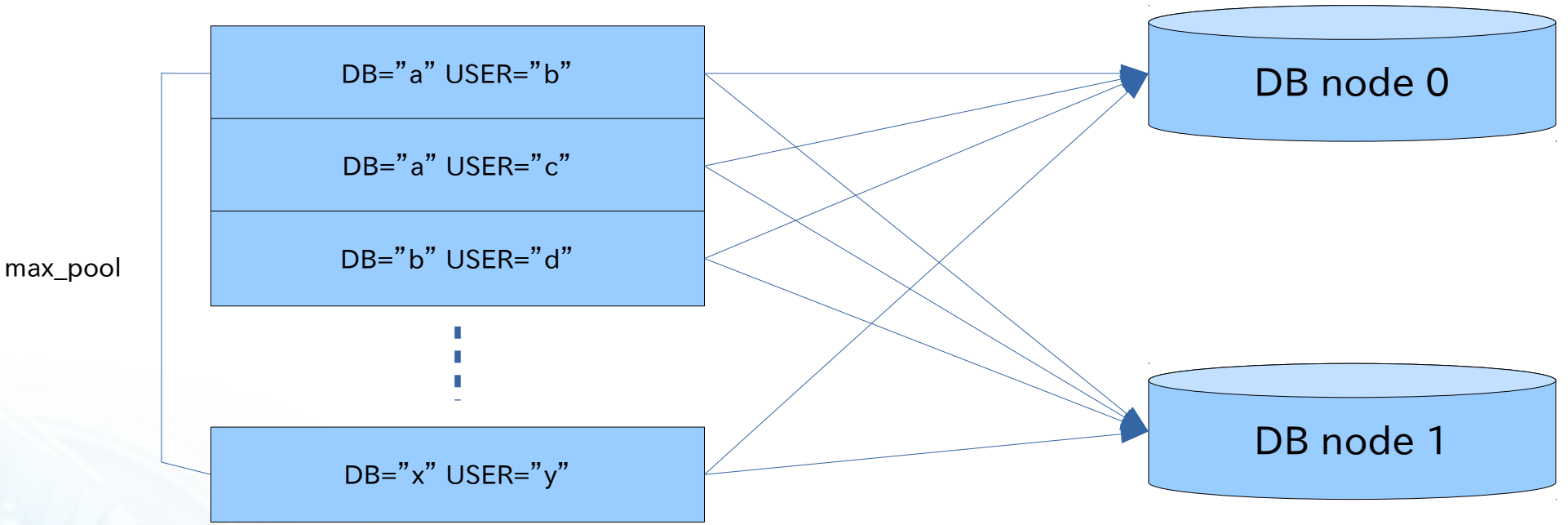


コネクションプーリングの意義

- PostgreSQLは接続に時間がかかる
- DBに接続しっぱなしにして接続時間を節約
- コネクションプーリングの有効時間を設定することも可能
- Javaなどの環境では自前のコネクションプーリングを持っていることがあり、その場合は効果はない



コネクションプーリングの仕組み



max_poolを使い切ると一番古いコネクションが
開放され、そのスロットが再利用される
(LRU管理)

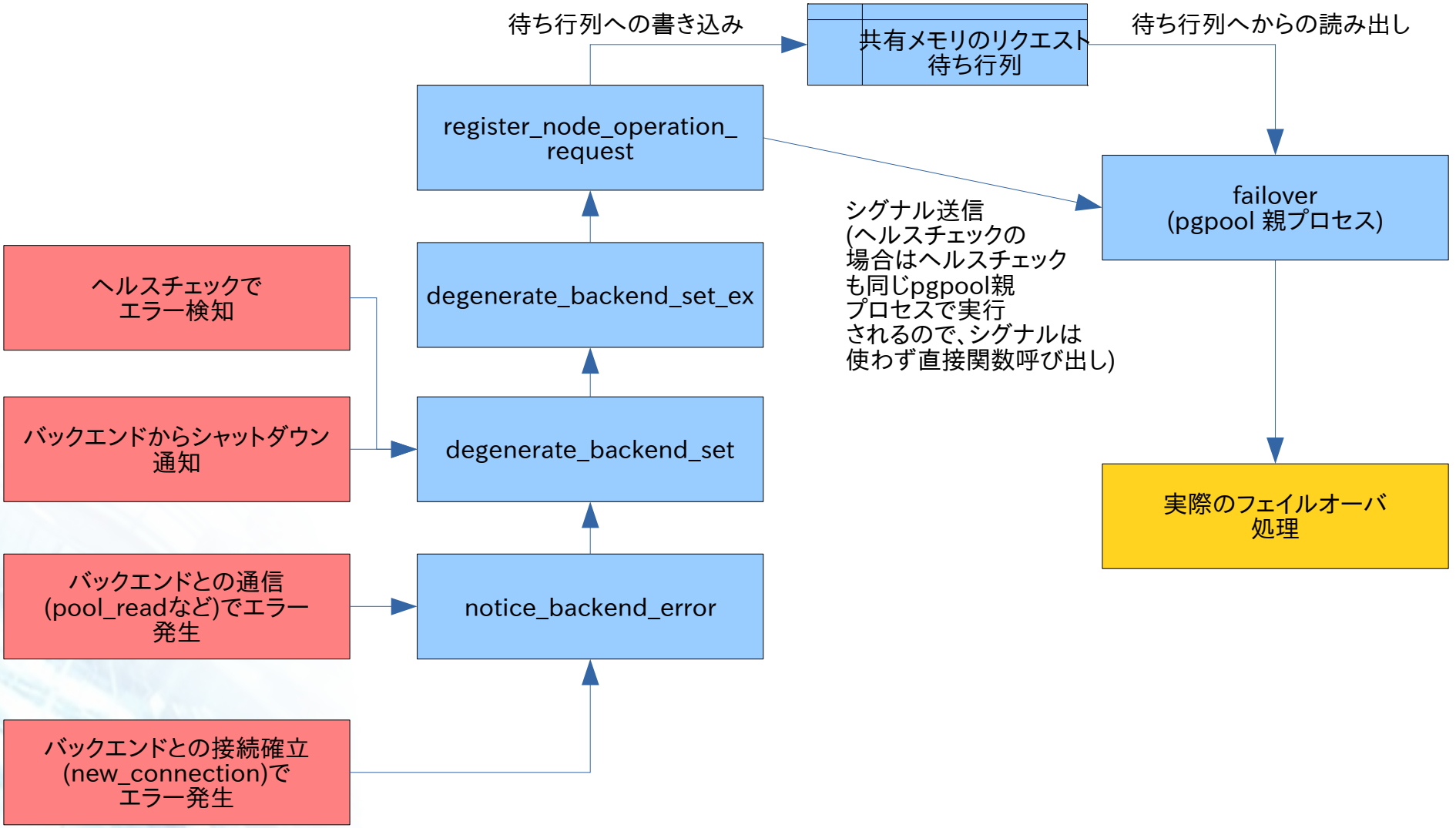
フェイルオーバ処理

- ヘルスチェック、あるいはバックエンドとの接続や処理中にエラーが発生すると、バックエンドは切り離され、残ったバックエンドで処理を継続する。これを「フェイルオーバ」(fail over)という
  - 「縮退」(degeneration)という言葉を使う場合もあるが、意味は同じ
- 手動でフェイルオーバを引き起こすことも可能で、これを「スイッチオーバ」(switch over)という
  - pcp_detach_nodeを利用
  - この場合、バックエンドが実際に停止しているかどうかにはpgpool-IIは関知しない

フェイルオーバ処理の概要

- フェイルオーバの引き金はいくつかある
  - ヘルスチェック
  - バックエンドからシャットダウン通知
  - バックエンドとの通信でエラー
  - バックエンドとの接続確立時にエラー
- いずれの場合も、同じルートでファイルオーバ処理が動く（次頁スライド参照）
- フェイルオーバ処理は主にpgpoolメインで行われる
  - pgpool子プロセスの再起動
  - 共有メモリ上のステータスの変更
  - フェイルオーバスクリプトやフォローマスターコマンドの起動

フェイルオーバ発生の契機と通知の流れ



フェイルオーバ処理の詳細

- src/main/pgpool_main.c:failover()に処理が記述されている
- 処理の流れ
  - すでにフェイルオーバ処理中なら何もせずに終了
  - 共有メモリのリクエストキューを読み出す
  - すでにダウンステータスのノードでなければ、ダウンステータスをセット
  - 新しいマスタノードを選定
    - マスタノードとは、ダウンしていないノードのうち、物理的にノード番号が一番小さなもの
  - ノードダウンの状況によって処理を切り分ける
    - ストリーミングレプリケーションモードの場合
      - pgpool-II 3.6以降では、ノードダウンリクエストの詳細が“switch over”であり、ダウンしたノードがスタンバイノードの場合は、ロードバランスノードとして、ダウンしたスタンバイノードを使っているpgpool子プロセスのみ再起動する（実際には、ヘルスチェックでタイムアウトしたとき以外は、すべて“swicth over”扱いとなる）。再起動されない子プロセスは、現在のセッションが終了すると、自分で再起動する
    - それ以外の場合は、全pgpool子プロセスを再起動する
  - （あれば）failover_commandを起動する
    - 引数は、ダウンしたノードID、failover_command、旧マスターノード、新マスターノード、旧プライマリノード
    - ダウンしたノードが複数あれば、failover_commandは各ノードに対して起動される
  - プライマリノードがダウンした場合は、（あれば）follow_master_commandを起動する
    - follow_master_commandは、ダウンしていないすべてのスタンバイノードに対して起動される
  - 必要ならばPCPプロセスを再起動する

フェイルオーバコマンドの実行

- フェイルオーバが発生した場合、pgpool.confの“failover_command”が設定されていれば、そのコマンドが実行される
  - 主な役割は、プライマリノードが落ちた時に新しいプライマリを決めてスタンバイからの昇格処理を行うこと
  - その他、フェイルオーバの発生をメールで知らせるなども可能
- 起動は、pgpool-IIの親プロセスが行う
- よって、フェイルオーバコマンドの実行権限はpgpool-IIプロセスと同じ
- 「新しいプライマリノード番号」の情報は渡ってこないので注意!
  - 新しいプライマリノードを選ぶのは、フェイルオーバコマンドの役割
  - 例:新しいマスタノードを新しいプライマリノードとする

- フェイルオーバコマンドに渡される引数
  - %p: ダウンしたノードのポート番号
  - %D: ダウンしたノードのデータベースクラスタへのパス
  - %d: ダウンしたノードのノード番号
  - %h: ダウンしたノードのホスト名
  - %H: 新しいマスタノードホスト名
  - %m: 新しいマスタノード番号
  - %r: 新しいマスタノードのポート番号
  - %R: 新しいマスタノードのデータベースクラスタへのパス
  - %M: 古いマスタノード番号
  - %P: 古いプライマリノード番号

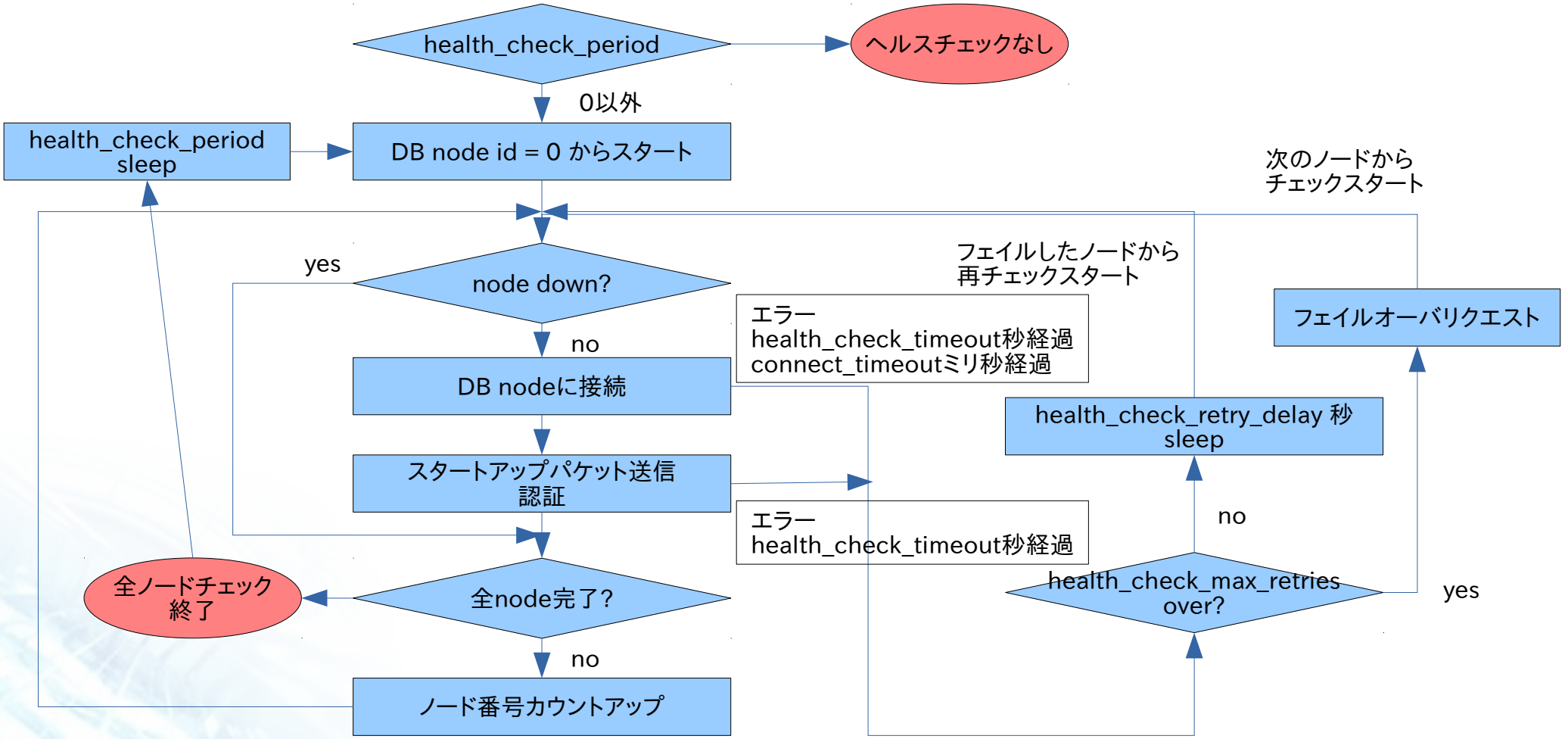
フェイルオーバコマンドの例

```
#!/bin/sh
# Execute command by fail over.
# special values:  %d = node id
#                  %h = host name
#                  %p = port number
#                  %D = database cluster path
#                  %m = new master node id
#                  %M = old master node id
#                  %H = new master node host name
#                  %P = old primary node id
#                  %% = '%' character
failed_node_id=$1
failed_host_name=$2
failed_port=$3
failed_db_cluster=$4
new_master_id=$5
old_master_id=$6
new_master_host_name=$7
old_primary_node_id=$8
trigger=/var/log/pgpool/trigger/trigger_file1

if [ $failed_node_id = $old_primary_node_id ];then# primay failed
    ssh -T postgres@$new_master_host_name touch $trigger    # let standby take over
fi
```

ヘルスチェック

- pgpool-II親プロセスの中で実施されるDBの監視処理
- すべてのバックエンドに対して、スタートアップパケットを送信し、応答が返ってきたことをもって正常と判断する
- それ以外は異常と見なす
- したがって、バックエンドがダウンした時だけでなく、ネットワークに異常がある場合にも異常と見なされる
- ネットワークに一時的なエラーが発生する可能性がある場合は、ヘルスチェックのリトライを設定するのが効果的

ヘルスチェック処理と関係パラメータ



検索負荷分散

- 検索処理(read query)を複数のバックエンドに分散することにより、システム全体として性能向上を図ることができる。これを「検索負荷分散」あるいは単に「負荷分散」と呼ぶ(load balance, load balancing)
- ストリーミングレプリケーションモードでは、この他プライマリサーバにread queryを投げずに更新処理に専念させることで、性能向上を狙う戦略もある
- フロントエンドがpgpool-IIに接続した際に、負荷分散用のDBノード（ロードバランスノード）が選択され、そのセッションが終了するまで変化しない
- ストリーミングレプリケーションモードでは、必ずプライマリサーバにクエリを接続し、その他にロードバランスノードに検索クエリを送信する。プライマリサーバがロードバランスノードに選択されることもあり、その場合はプライマリサーバのみにクエリが送信されることになる

負荷分散に関係する設定項目

- load_balance_mode
  - offの場合は負荷分散は無効
    - ストリーミングレプリケーションモードの場合は、プライマリノードにのみクエリを送る
    - それ以外の場合は、マスターノードに送る
- backend_weightN (Nは0,1,2...)
  - 各DBノードの負荷分散の重み
  - 0にすると、ロードバランスノードにそのノードが選ばれない
  - 0以外の場合は、他のbackend_weightとの相対比較になる
- database_redirect_preference_list
  - ストリーミングレプリケーションモードでのみ有効
  - 「データベース名:DBノード番号」で、そのデータベースに接続したら、指定DBノード番号に検索クエリを投げる。カンマ区切りで複数項目指定可能。データベース名には正規表現を利用できる。
  - DBノード番号として、“primary”を指定すると、プライマリノードにクエリを投げる
  - DBノード番号として、“standby”を指定すると、スタンバイノードの中から、与えられたbackend_weightに基づいて負荷分散ノードを選ぶ
    - スタンバイノードがひとつもない場合は、プライマリノードを選択する
- app_name_redirect_preference_list
  - データベース名の代わりにアプリケーション名を使う以外は、database_redirect_preference_listと同じ
  - database_redirect_preference_listよりもこちらが優先

負荷分散に関係する設定項目(続き)

- white_function_listとblack_function_list
  - white_function_list
    - DBへの書き込みを行わない関数名を列挙する。この関数を使っているSELECTは負荷分散の対象となる
  - black_function_list
    - DBへの書き込みを行なう関数名を列挙する。この関数を使っているSELECTは負荷分散の対象とならない
  - white_function_listとblack_function_listは、どちらか一方のみ指定できる
  - white_function_listもblack_function_listもどちらも指定されていない場合は、関数を使っているSELECTは負荷分散の対象となる
- ignore_leading_white_space
  - 負荷分散の対象となるかどうかはクエリが “SELECT”で始まっているかどうかで判断する(実際にはWITHなども検索クエリと見なされる。詳細はソースコードを参照)。よって、行頭にスペースがあると、SELECTと見なされない。この設定を有効にすると、行頭のスペースが無視されるので、行頭にスペースを含むSELECTも負荷分散の対象となる
- allow_sql_comments
  - 同様の理由で、行頭にSQLコメントを含むSELECTを負荷分散の対象としたい場合は、有効にする
- sr_check_period, delay_threshold
  - sr_check_periodが0より大きく、ストリーミングレプリケーションの遅延がこの設定以上になると、そのDBノードには検索クエリが投げられなくなる。代わりに、プライマリノードにクエリが投げられる

SQL構文解析による負荷分散の判定
(ストリーミングレプリケーションモードのみ)

- ソースは、send_to_where (context/pool_query_context.c)
- パースツリーのノードタイプに対応したenumデータの配列を持っている(SelectStmtなど)。それをバイナリサーチして対応するノードを見つける
- 見つからない場合は一律プライマリノードに送る。SQLパーサが知らないSQL文を入力された場合に該当する(可能性としては、上記データを更新していない場合もある)
- SELECTの場合
  - SELECT INTO, SELECT FOR UPDATE, SELECT FOR SHAREはプライマリノード
  - WITH句を使っていて、WITH句にSELECT以外が含まれている場合はプライマリノード
  - それ以外はどこにでも送れる(負荷分散可能)
- COPY FROMは負荷分散可能
- LOCK文
  - ロックの強さがRowExclusive以上の場合はプライマリ、それ以外はすべてのノード(プライマリと負荷分散ノード)
- トランザクション文(BEGINなど)
  - 2層コミットはプライマリ、それ以外はすべてのノード
  - ただし、BEGIN READ WRITEは、ただのBEGINにしてスタンバイに送る
- トランザクション文(ENDなど)
  - すべてのノードに送る

SQL構文解析による負荷分散の判定
(ストリーミングレプリケーションモードのみ)続き

- SET transaction_read_only
  - TO OFFならプライマリ、それ以外はすべてのノード
- 以下のSET文はすべてプライマリ
  - SET TRANSACTION ISOLATION LEVEL SERIALIZABLE or SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL SERIALIZABLE
  - SET transaction_isolation TO 'serializable'
  - SET default_transaction_isolation TO 'serializable'
- SET TRANSACTION
  - “READ WRITE”はプライマリ、それ以外はすべてのノード
- SET SESSION CHARACTERISTIC
  - “READ WRITE”はプライマリ、それ以外はすべてのノード
- DISCARD
  - すべてのノード
- PREPARE
  - 再帰的に目的のSQLを解析
- EXECUTE, DEALLOCATE
  - ここではとりあえずプライマリとするが、PREPAREと同じ送信先をあとで選択

ロードバランスノードの決定

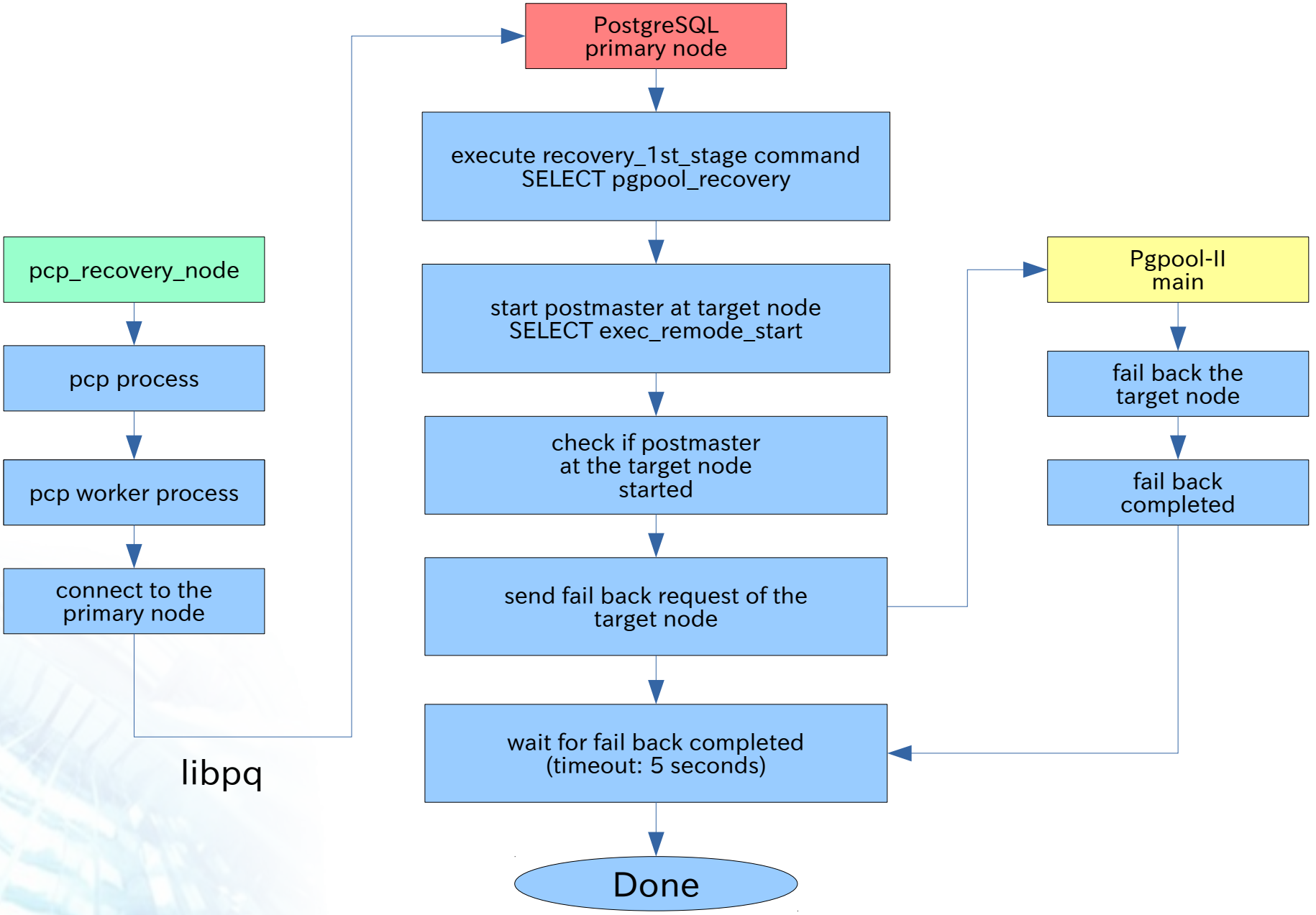
- ロードバランスノードとは？
  - フロントエンドがpgpool-IIに接続した時に割り当てられるDBノード番号で、負荷分散できるクエリが投げられる先
  - セッションが終了するまでは、そのロードバランスノードが維持される
- ロードバランスノードの決定方法
  - db_redirect_preference_list によりロードバランスノードを仮決めする
  - app_redirect_preference_list によりロードバランスノードを仮決めする
  - 「スタンバイノードのうちのどれでも良い」場合以外はここまでで決まったノードをロードバランスノードとする。
  - 乱数を生成してどのノードをロードバランスノードにするか決める。backend_weightが均等でない場合は、その重み付けが反映される
  - Streaming replication modeでは、primaryノードがロードバランスノードを兼用する場合もある。特に standby が1台しかない構成では、1/2の確率でそうした状況になる (backend_weightが等しく、かつ db_redirect_preference や app_redirect_preference が設定されていない場合)

オンラインリカバリ

- オンラインリカバリとは、Pgpool-IIを停止することなくDBノードを他のノードと同期させ、運用に供する操作のこと
- ここでは、ストリーミングレプリケーションモードを前提に説明する
- recovery_1st_stage_command
  - primaryからターゲットノードにbase backupするユーザ定義のスクリプト

recovery_1st_stage_commandの例

```
#/bin/sh
#
master_node_host_name=`hostname`
master_db_cluster=$1
recovery_node_host_name=$2
recovery_db_cluster=$3
PORT=$4
tmp=/tmp/mytemp$$
trap "rm -f $tmp" 0 1 2 3 15

psql -p $PORT -c "SELECT pg_start_backup('Streaming Replication', true)" postgres

rsync -C -a -c --delete --exclude postgresql.conf --exclude postmaster.pid ¥
--exclude postmaster.opts --exclude pg_log ¥
--exclude recovery.conf --exclude recovery.done ¥
--exclude pg_xlog ¥
$master_db_cluster/ $recovery_node_host_name:$recovery_db_cluster

ssh -T $recovery_node_host_name mkdir $recovery_db_cluster/pg_xlog
ssh -T $recovery_node_host_name chmod 700 $recovery_db_cluster/pg_xlog
ssh -T $recovery_node_host_name rm -f $recovery_db_cluster/recovery.done

cat > $tmp <<EOF
standby_mode          = 'on'
primary_conninfo      = 'host=$master_node_host_name port=$PORT user=postgres'
trigger_file = '/var/log/pgpool/trigger/trigger_file1'
EOF

scp $tmp $recovery_node_host_name:$recovery_db_cluster/recovery.conf

psql -p $PORT -c "SELECT pg_stop_backup()" postgres
```



次回の話題

- 認証
- クエリキャッシュ
- watchdog

主なURL

- Pgpool-IIオフィシャルサイト
  - 日米サイトあり
  - http://www.pgpool.net
- GitHubミラー
  - https://github.com/pgpool/pgpool2

Pgpool-II