信頼性を向上させ、PostgreSQL 10に対応したPgpool-II 3.7のご紹介

PGConf.ASIA 2017

SRA OSS, Inc. 日本支社
取締役支社長
石井　達夫

SRA OSS,INC.

自己紹介

- OSSの開発とビジネスに携わっています
  - OSS活動
    - PostgreSQLのコミッタ
    - PostgreSQL用のクラスタソフト Pgpool-II の開発
  - ビジネス
    - PostgreSQLなどのOSSのサポート
    - PowerGres製品の製造、販売
    - PostgreSQLトレーニング

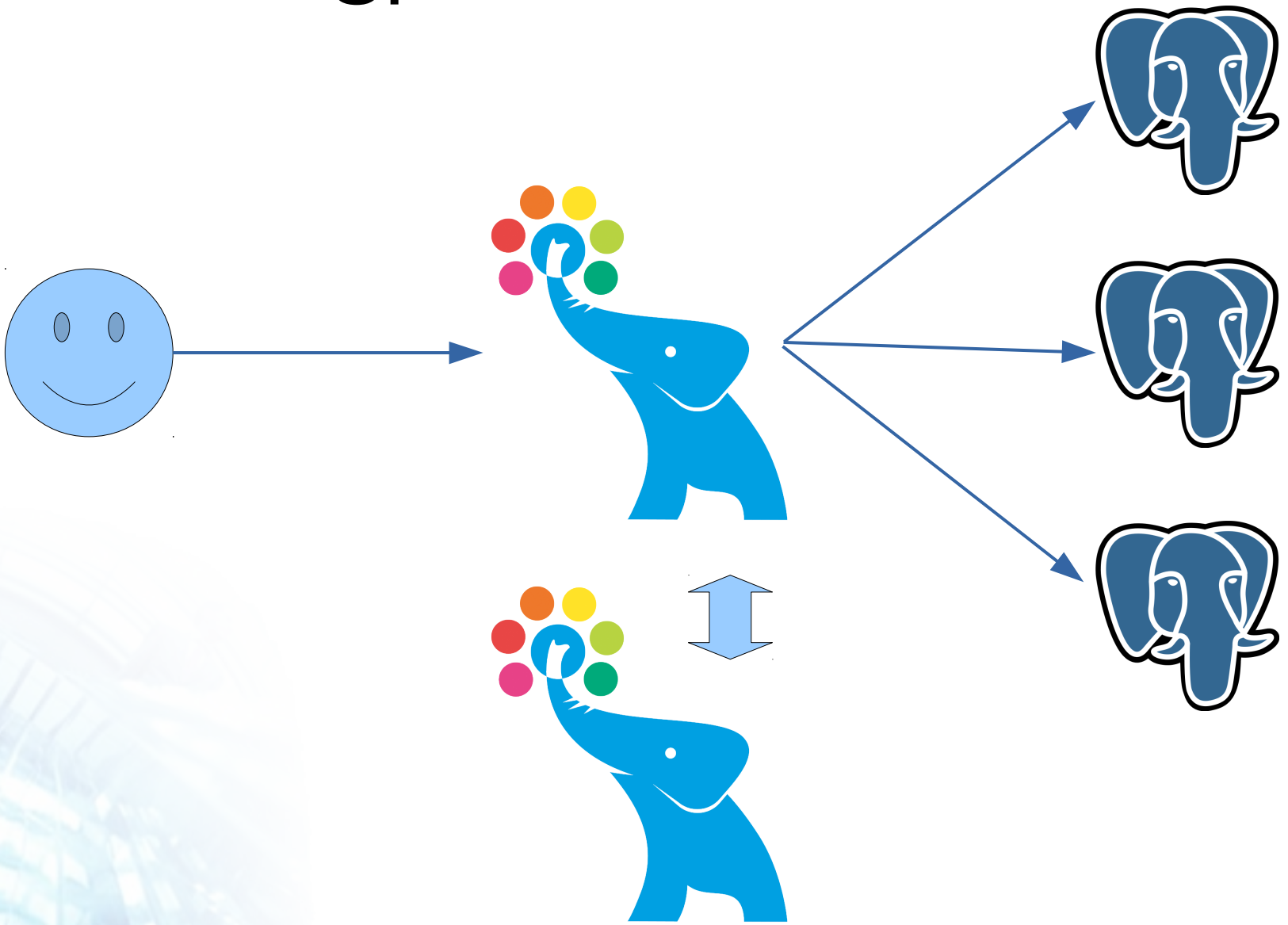
本日のアジェンダ

- Pgpool-IIとは
- Pgpool-II 3.7について
- 今後の予定

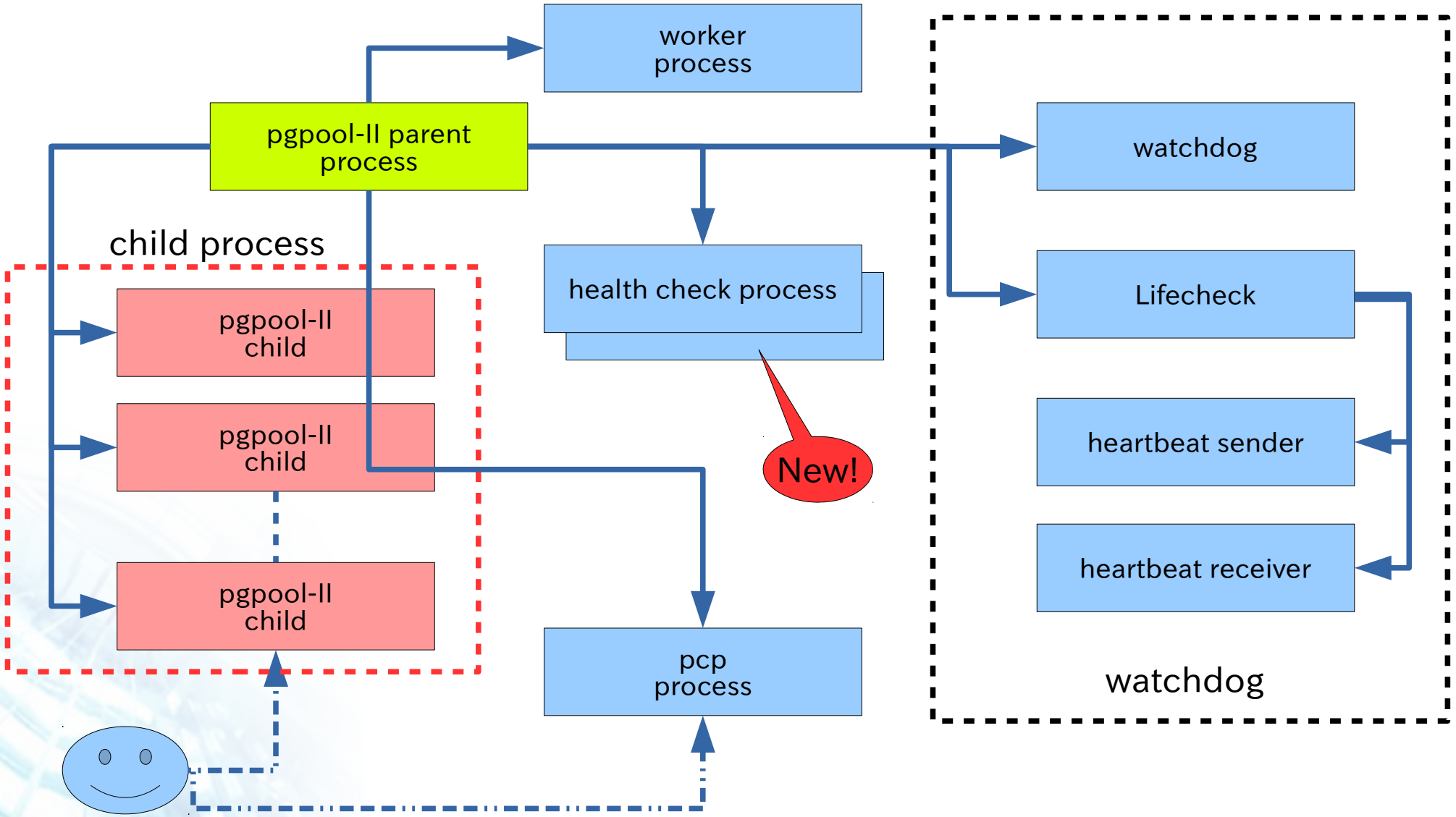
Pgpool-IIとは

- PostgreSQL専用のクラスタ管理ミドルウェア
  - PostgreSQL 7.4から10まで幅広いバージョンに対応
  - Amazon AuroraやRedShiftなど、PostgreSQLから派生した製品にも対応
- PostgreSQLと同じOSSライセンスで配付
- 複数のPostgreSQLを「クラスタ」として管理
  - 故障したサーバを自動的に検知して排除(自動フェイルオーバ)
  - 新しいサーバ、修理が完了したサーバを再同期してクラスタに復帰（オンラインリカバリ、フェイルバック）
  - 書き込みクエリと読み込みクエリを自動判断して振り分け
  - 読み込みクエリを複数のサーバに振り分け（負荷分散）
- PostgreSQLにない機能の追加
  - コネクションプーリング
  - クエリキャッシュ

Pgpool-IIの利用形態

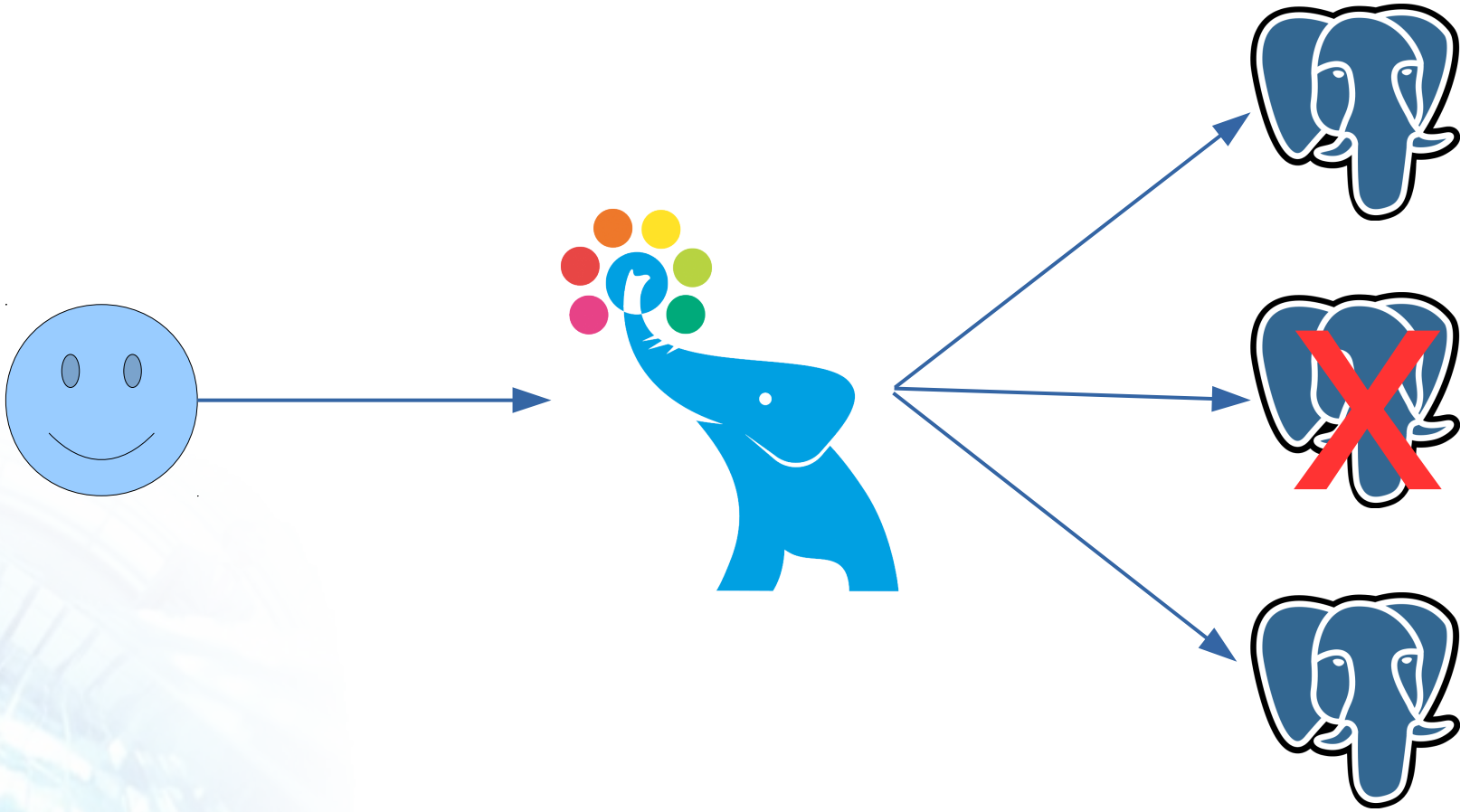


Pgpool-IIのプロセス構造



Pgpool-II 3.7の改良：ヘルスチェック

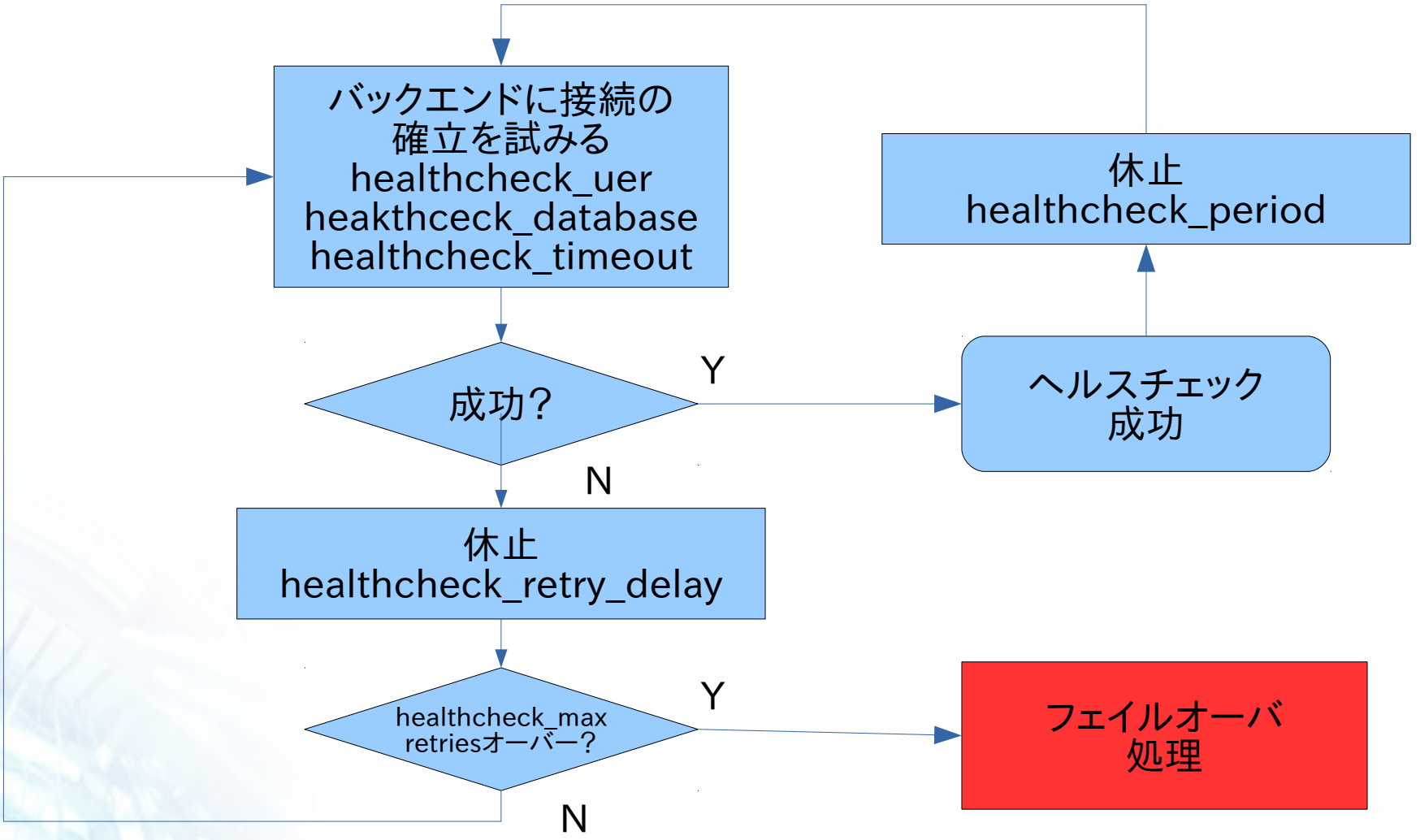
自動フェイルオーバ: PostgreSQL障害への対応



フェイルオーバ処理

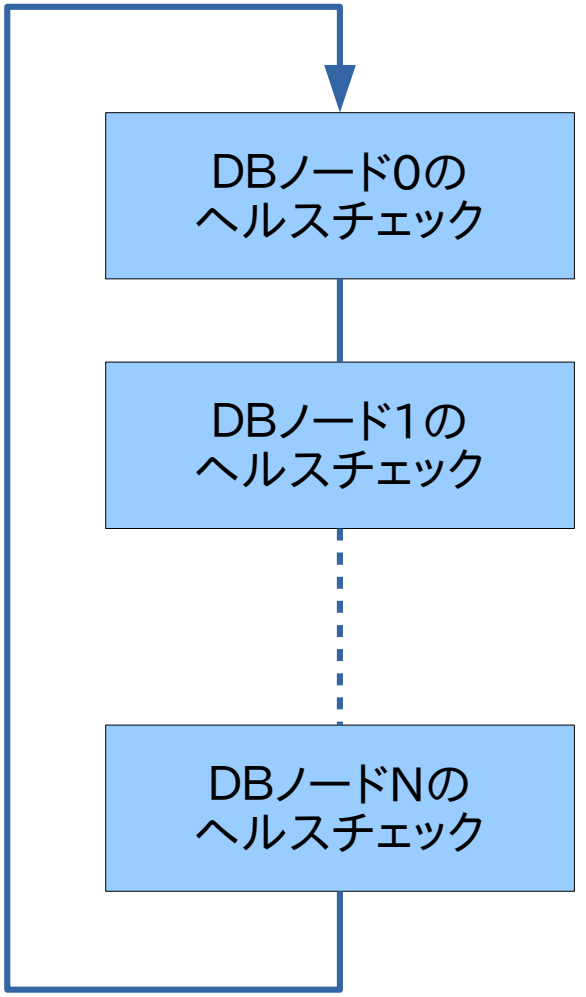
- フェイルオーバの契機
  - ヘルスチェック
    - バックエンドに接続を試み、成功したらOKとする
    - 各ノード毎にヘルスチェックを担当するプロセスが存在（Pgpool-II 3.7の新機能）
  - バックエンドからシャットダウン通知
    - シャットダウンのエラコードを検知
  - バックエンドとの通信でエラー
    - ソケット通信のエラーを検知
    - 通信経路のエラーなのか、バックエンドがダウンしているのかの区別はしない
  - バックエンドとの接続確立時にエラー
    - ソケット通信のエラーを検知
    - 通信経路のエラーなのか、バックエンドがダウンしているのかの区別はしない
  - PCPコマンドによる強制フェイルオーバ

ヘルスチェック処理とパラメータ



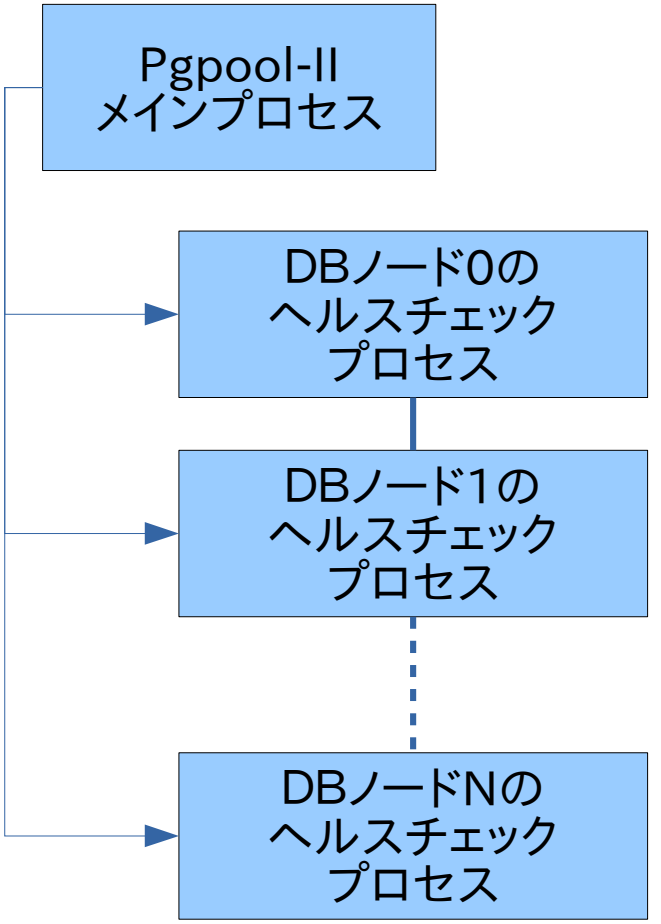
ヘルスチェックの問題点

- DBノードを順番にヘルスチェックするため、後のノードになればなるほど異常検出が遅れる可能性
- 1サイクルのヘルスチェック処理に時間がかかる可能性
- ヘルスチェックがPgpool-IIのメインプロセスの中で実行されるため、ヘルスチェック処理がメインプロセスに影響を与える可能性



Pgpool-II 3.7での ヘルスチェックの改良

- Pgpool-IIメインプロセスから各DBノード専用のヘルスチェックプロセスを起動
- Pgpool-IIメインプロセスがヘルスチェックプロセスの影響を受けず、安定性が向上
- 異常検知の遅延がなくなった
- 各DBノードごとに別々のヘルスチェックパラメータを設定可能



DBノードごとのヘルチェック パラメータの設定方法

- 従来のヘルスチェックパラメータに加え、たとえば“healthcheck_period0”のように、パラメータ名の最後にノード番号を追加すると、そのノード専用のパラメータになる
- 従来のノード番号なしのパラメータも記述可能。ノードごとのパラメータがない場合は、ノード番号なしのパラメータが採用される
- ノードごとのヘルスチェックパラメータの表示は“pgpool show health_check”で可能

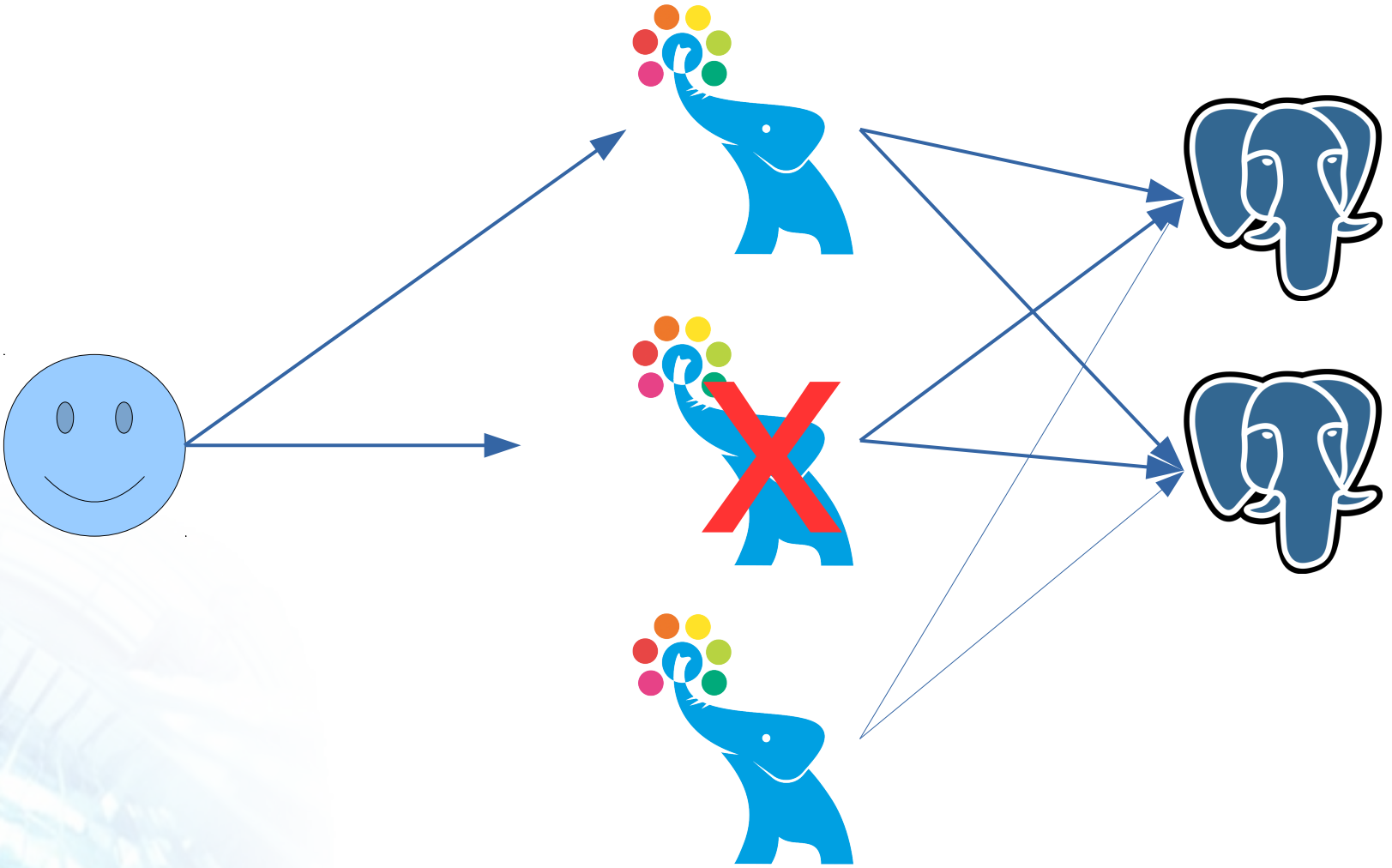
```
test=# pgpool show health_check;
           item             |  value  |                                                                                                  description
----------------------------+---------+-------------------------------------------------------------------------------------------------------------------------------------------
 health_check_period        | 10      | Time interval in seconds between the health checks.
 health_check_timeout       | 20      | Backend node health check timeout value in seconds.
 health_check_user          | t-ishii | User name for PostgreSQL backend health check.
 health_check_password      | *****   | Password for PostgreSQL backend health check database user.
 health_check_database      |         | The database name to be used to perform PostgreSQL backend health check.
 health_check_max_retries   | 5       | The maximum number of times to retry a failed health check before giving up and initiating failover.
 health_check_retry_delay   | 1       | The amount of time in seconds to wait between failed health check retries.
 connect_timeout            | 1000    | Timeout in milliseconds before giving up connecting to backend.
 health_check_period0       | 0       | Time interval in seconds between the health checks.
 health_check_timeout0      | 20      | Backend node health check timeout value in seconds.
 health_check_user0         | nobody  | User name for PostgreSQL backend health check.
 health_check_password0     | *****   | Password for PostgreSQL backend health check database user.
 health_check_database0     |         | The database name to be used to perform PostgreSQL backend health check.
 health_check_max_retries0  | 0       | The maximum number of times to retry a failed health check before giving up and initiating failover.
 health_check_retry_delay0  | 1       | The amount of time in seconds to wait between failed health check retries.
 connect_timeout0           | 10000   | Timeout in milliseconds before giving up connecting to backend.
 health_check_period1       | 10      | Time interval in seconds between the health checks.
 health_check_timeout1      | 20      | Backend node health check timeout value in seconds.
 health_check_user1         | t-ishii | User name for PostgreSQL backend health check.
 health_check_password1     | *****   | Password for PostgreSQL backend health check database user.
 health_check_database1     |         | The database name to be used to perform PostgreSQL backend health check.
 health_check_max_retries1  | 5       | The maximum number of times to retry a failed health check before giving up and initiating failover.
 health_check_retry_delay1  | 1       | The amount of time in seconds to wait between failed health check retries.
 connect_timeout1           | 1000    | Timeout in milliseconds before giving up connecting to backend.
(24 rows)
```

Pgpool-II 3.7の改良: Watchdog

Watchdog

- Pgpool-II自身の管理プロセス
- Pgpool-II自身が単一障害点にならないように、バックアップのPgpool-IIを配置可能にする
- Pgpool-IIに障害発生時には、別のPgpool-IIがVIPを引き継ぐ
- 奇数個(ただし3以上)のPgpool-IIが配置されている場合は、ネットワーク分断時にスプリットブレインが発生しないようにすることが可能

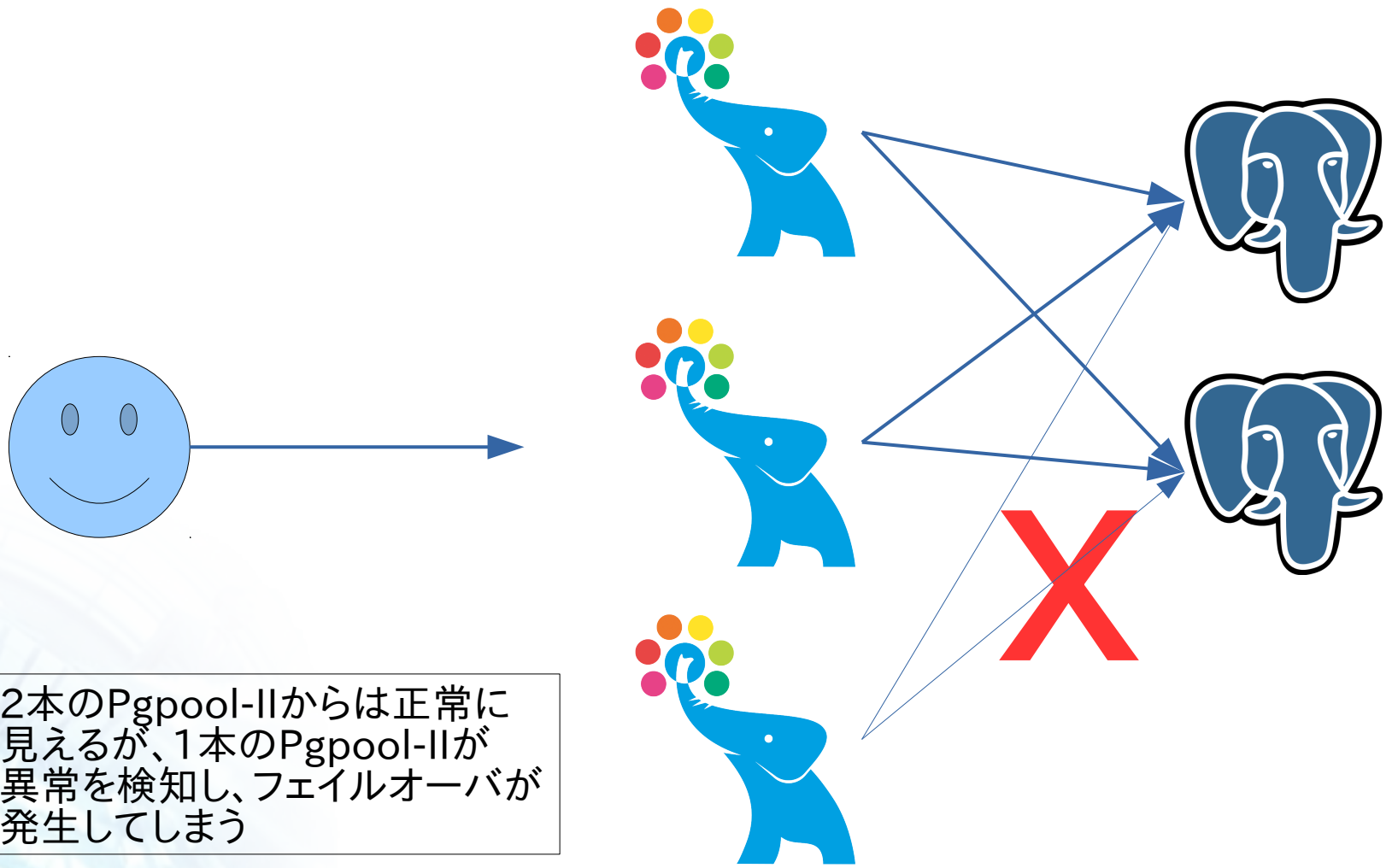
WatchdogにおけるVIPの切換



バックエンド障害検知の問題点

- バックエンド障害の検知は、Pgpool-IIから見て、バックエンドに接続が可能かどうかで判断しているので、本当にバックエンドが故障しているか、ネットワークが故障しているのかは区別できない
- Pgpool-IIが1本しかないときには問題にならない
  - 原因がなんであれ、PostgreSQLが使えないことには変わりない
- 複数のPgpool-IIが配置されているときに、特定のネットワークが故障しただけで、フェイルオーバが引き起こされてしまう

PostgreSQL障害誤検知問題



2本のPgpool-IIからは正常に
見えるが、1本のPgpool-IIが
異常を検知し、フェイルオーバが
発生してしまう

クォーラム合意方式のフェイルオーバ

- クォーラムが存在する場合に、各Watchdogは、バックエンドの障害を検知した際に「投票」を行う。
- マスターWatchdogが票を集計し、過半数票が取れた時のみフェイルオーバを実施。これにより、ネットワーク故障によるPostgreSQL障害の誤検知を防ぐことができる
- 否決されたwatchdogは、そのバックエンドを“quarantine”(隔離)状態とし、ローカルにそのバックエンドを切り離す
  - ただし、プライマリが隔離状態になっても、スタンバイの昇格操作は行われないので要注意
  - pcp_attach_nodeを隔離状態のバックエンドに実施しても無視される
- pcp_detatch_nodeによる手動フェイルオーバは、この設定を無視して強制的に行われる

「クォーラムが存在する」とは？

- 生きているwatchdogノードの数(Pgpool-IIの数)を全部足しても多数派を形成できない場合、「クォーラムが存在しない」と言う
  - watchdogノード数=5で、生きているノードが2以下の場合
  - watchdogノード数=3で、生きているノードが1以下の場合
- watchdogノード数が偶数の場合は、生きているノード数が1以上ならクォーラムが存在する、と決める

watchdogに追加されたパラメータ(1)

- failover_when_quorum_exists
  - クォーラムが存在すればフェイルオーバを実施
  - クォーラムが存在しない場合は、そのバックエンドを“quarantine”(隔離)状態とし、ローカルにそのバックエンドを切り離す
    - ただし、プライマリが隔離状態になっても、failover_commandは実行されず、スタンバイの昇格操作は行われないので要注意
    - pcp_detach_nodeを使って強制的にフェイルオーバさせることは可能
    - pcp_attach_nodeを隔離状態のバックエンドに実施しても無視される
      - ローカルに「ダウン」状態として扱う
  - デフォルトON。OFFならば、従来と同じ動作(クォーラムが存在しなくてもフェイルオーバを実施）

watchdogに追加されたパラメータ(2)

- failover_require_consensus
  - failover_when_quorum_existsがONかつクォーラムが存在する時のみ有効
  - 過半数のwatchdogノードが合意した時にのみフェイルオーバを実施する
  - デフォルトON

watchdogに追加されたパラメータ(3)

- allow_multiple_failover_requests_from_node
  - ONのとき、一つのwatchdogノードが複数のフェイルオーバリクエストを送信できる、すなわち複数票が投票できる
    - たとえばヘルスチェックでエラーが検出されても有効票に達しない場合、同じ障害が続いていれば、次のヘルスチェックで再度エラーが検出され、都合2票が投票されることになる
  - 複数投票することにより、フェイルオーバを引き起こせる
    - 他のwatchdogが検出しない恒久的な障害が発生した時にフェイルオーバを引き起こしたい時に使用
  - failover_when_quorum_existsとfailover_require_consensusがONの時のみ有効
  - デフォルトOFF

Amazon Auroaへの対応

- Amazon Auroraとは
  - 正式名称: PostgreSQL Compatibility for Amazon Aurora
  - 高いWrite性能（RDSの2-3倍)、リードレプリカ、自動フェイルオーバ
- 検索処理の負荷分散、クエリキャッシュ、コネクションプーリングなど、自動フィエルオーバ以外の機能をサポート
- Auroraでは常にWriter(マスターノード)が決まっているため、新しく “ALWAYS_MASTER”フラグを追加
- 詳細な設定方法がドキュメントに書かれています

論理レプリケーションへの対応

- 論理レプリケーションとは
  - PostgreSQL 10で追加された新機能
  - 論理的なトランザクションログを「パブリッシャー」から「サブスクライバー」に送信することによって、特定のテーブルだけをレプリケーション可能
- Pgpool-IIの論理レプリケーションモードの動作
  - すべてのパブリッシャーは「マスターノード」に存在すると想定
  - すべてのサブスクライバーは「スレーブノード」に存在すると仮定
  - write queryはマスターノードに
  - read queryはマスターノードとスレーブノードの間で負荷分散
  - 負荷分散されるテーブルがレプリケーションされるようにするのはユーザの責任
  - pgpool_setupも対応(-m l)

PostgreSQL 10パーサの取り込み

- PostgreSQL 10で追加された、“CREATE SUBSCRIPTION”などの新しい構文を正しく解釈できるようになった

pool_hba.confの改良

- IPアドレスだけでなくホスト名が使えるようになった
- ロード処理が改善され、パフォーマンスが向上

その他の改良・変更

- pcp_node_infoに “role”カラムを追加
  - primary, standbyなどの区分を出力
- master_slave_sub_modeのデフォルトが、“slony” から “stream” に変更
- pgpool_setupがSlonyモードに対応
- “pgproto”を使用する新しいテストスィートを追加
  - 拡張問い合わせのテストが主目的

今後の予定

- 次のバージョン“3.8”以降でやりたいこと（私見も入っています）
  - プライマリノードの検出方法の改良
    - 現在は、最初にプライマリを見つけたら、それ以外はすべてスタンバイと見なす
      - しかし、これでは単独のバックエンドや、他のプライマリに追従しているスタンバイを見逃してしまう
      - 検査を厳密化（たとえば、スタンバイの接続先のプライマリをチェックする）
  - 負荷分散処理の改良
    - より細かい単位での指定
  - プラグイン機能の実装？
- 是非ご意見、リクエストをお寄せください

参考URL

- PostgreSQL
  - http://www.postgresql.org
- Pgpool-II
  - http://pgpool.net
- Pgproto
  - https://github.com/tatsuo-ishii/pgproto

Thank you!

SRA OSS, INC.