Pgpool-IIではじめる PostgreSQLのクラスタ運用

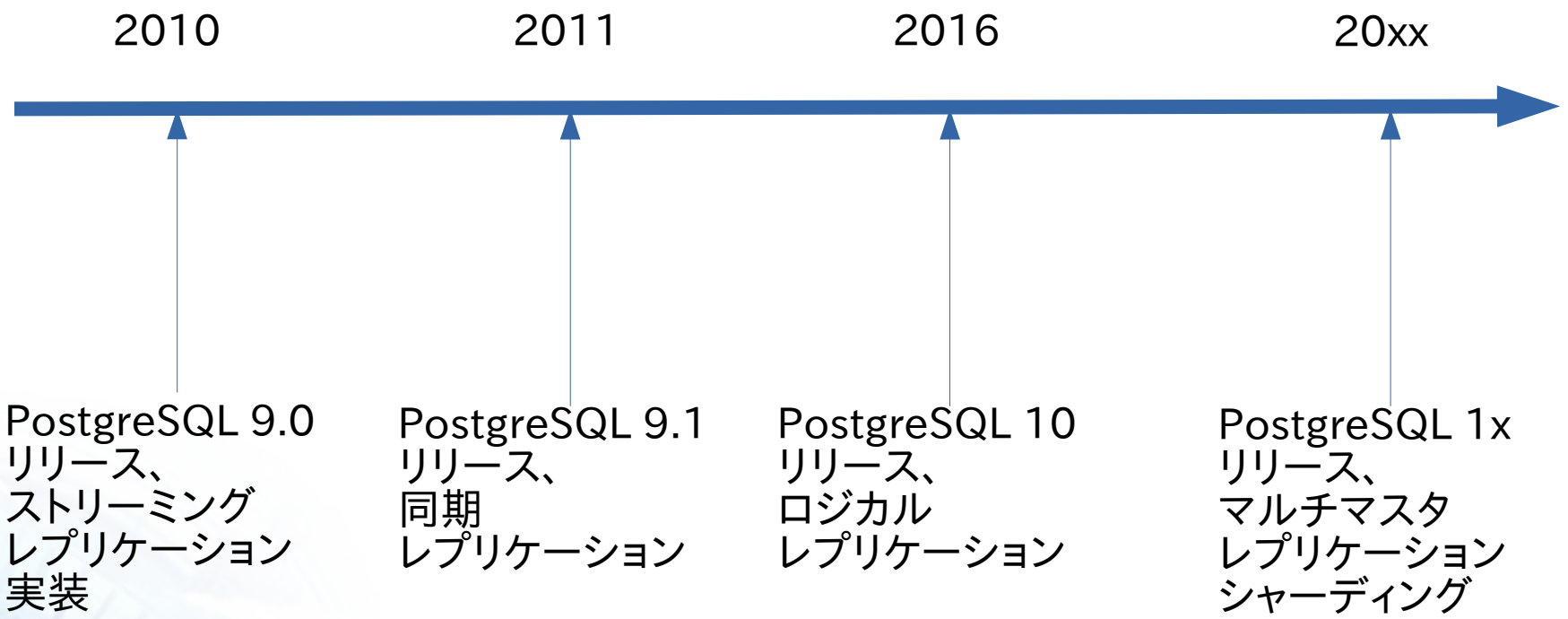
SRA OSS, Inc. 日本支社
取締役支社長
石井　達夫

SRA OSS, INC.

自己紹介

- SRA OSS, Inc. 日本支社という会社で、OSSの開発とビジネスに携わっています
  - PostgreSQLのリリース以来開発に関与し、現在はコミッタの一人
  - PostgreSQL用のクラスタソフト Pgpool-II の最初のバージョンを開発、今も開発リーダを務める

PostgreSQLのクラスタ機能のロードマップ



クラスタ構成によるメリット

- 可用性の向上
  - サーバダウン時の復旧時間の短縮
  - できるだけ直近のデータまで復旧したい
- 同時アクセス時の検索性能の向上
  - 複数の「レプリカ」で検索処理を分担して性能向上
  - 更新処理は性能向上できない
    - 別の技術が必要(PostgreSQLではまだ正式サポートされていない)

クラスタ構成による耐障害性の向上

バックアップ
取得

障害
発生!

復元

- 復元中は運用停止
- 復元に要する時間はデータ量に比例する(ときには数日以上かかることも)
- バックアップ時点までしか戻れない

プライマリ
サーバ

リアルタイム
コピー取得

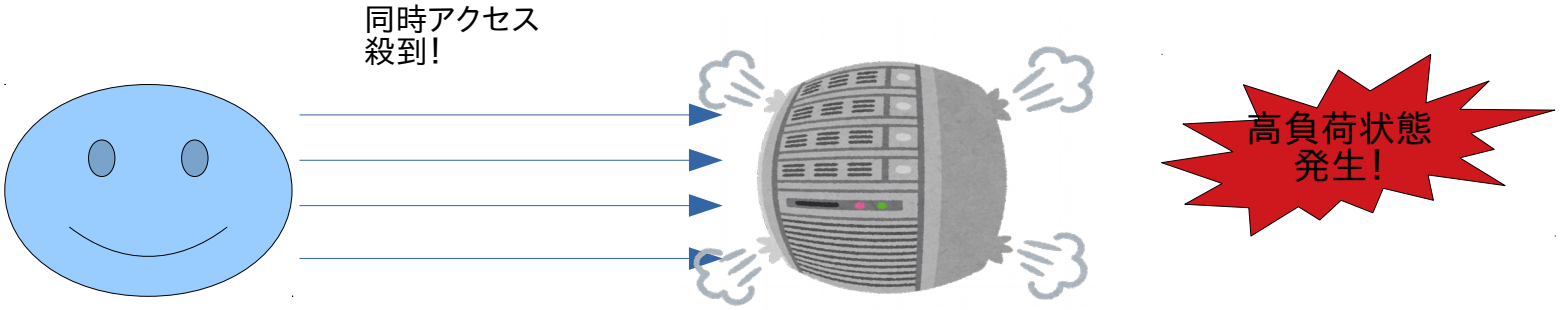
新プライマリ
サーバ

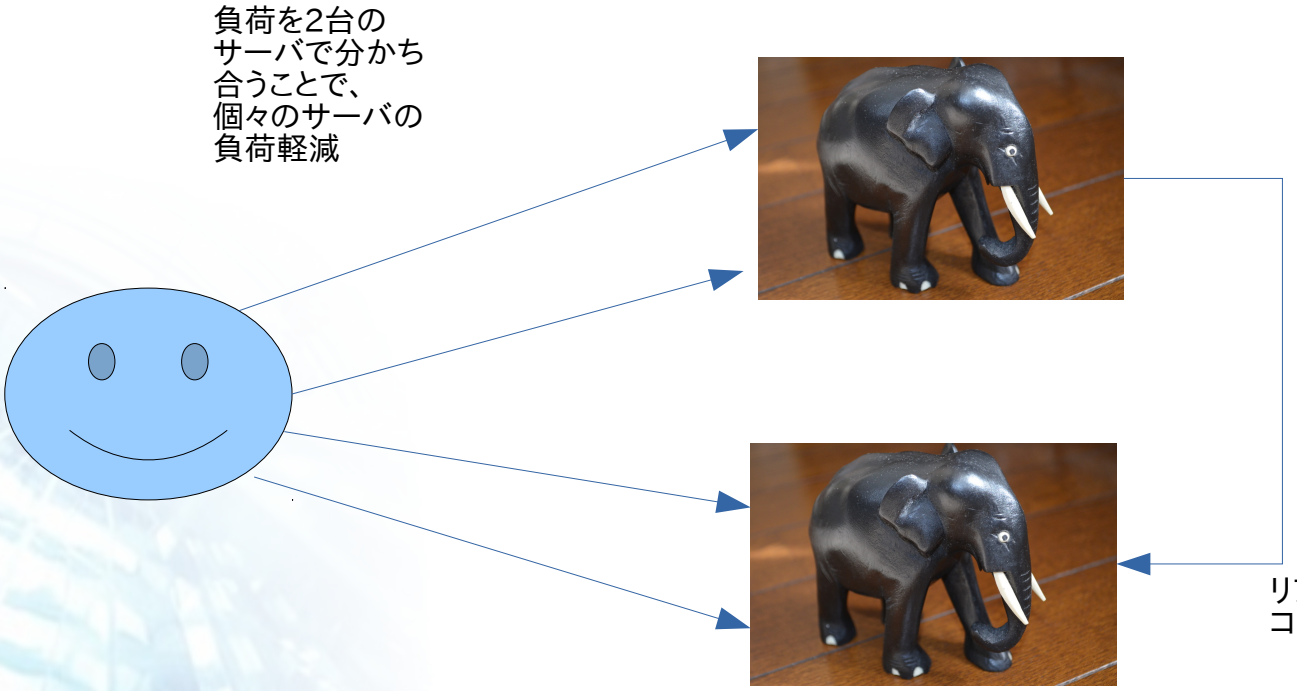
障害
発生!

リアルタイム
コピー取得

- 障害発生時にプライマリサーバを切り替え(フェイルオーバ)
- データ量が大きくてもサーバ切り替え時間は変わらない(数分以内)
- ほぼ直前のコミットまで復元可能

クラスタ構成による検索性能向上





- 検索負荷を2台のサーバで分かち合うことで、個々のサーバの負荷軽減
- 更新性能は向上しない

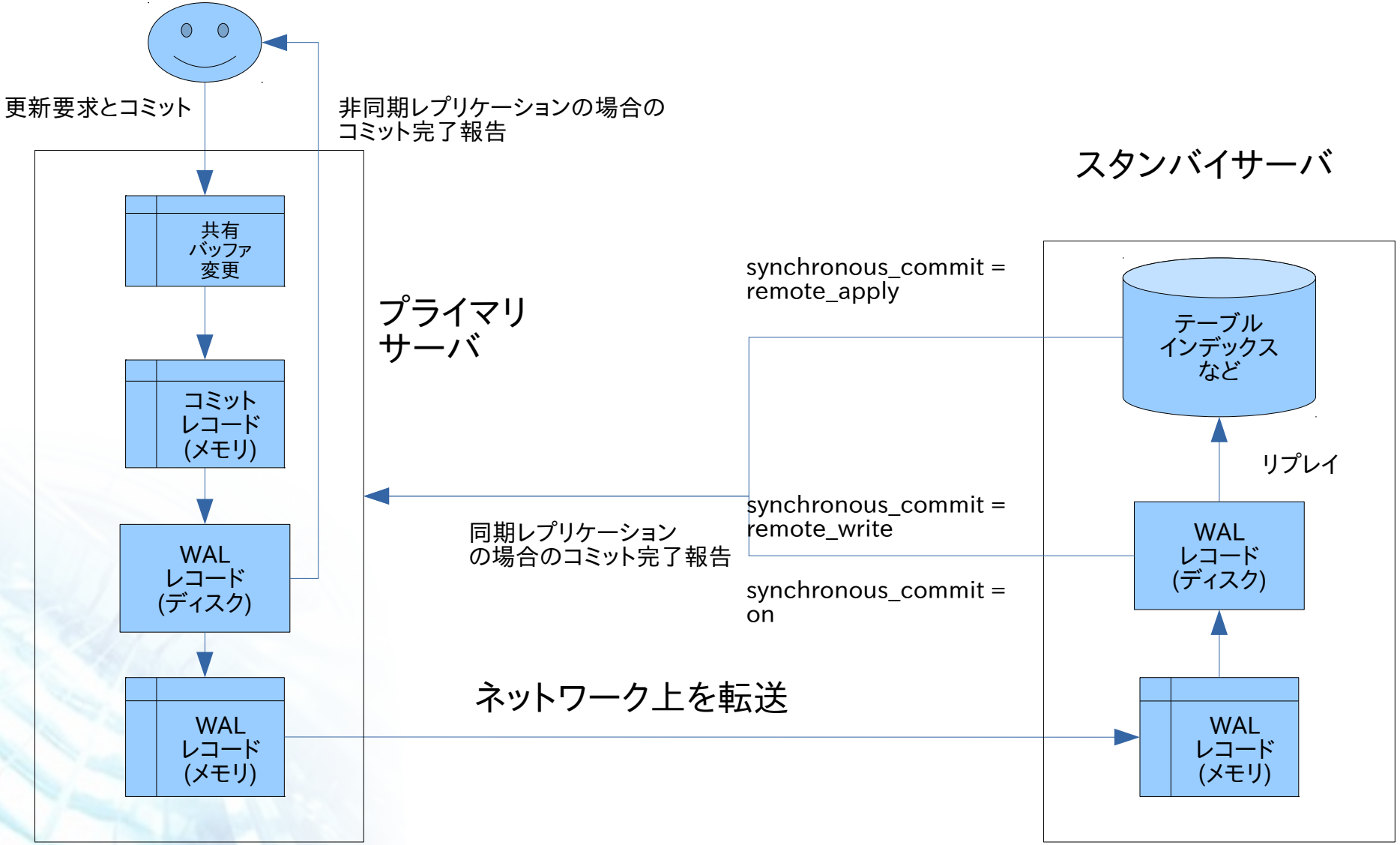
PostgreSQLにおける レプリケーション技術(1)

- ストリーミングレプリケーション
  - 広く使われ、実績もある(PostgreSQL 9.0以降で利用可能)
  - 「物理」レプリケーションの一種
  - トランザクションログレコードをリアルタイムで一個の主サーバ（プライマリ）から複数の従サーバ（スタンバイ）に転送、スタンバイはログを再生（リプレイ）してデータをコピーする
  - データ更新はプライマリでのみ可能。スタンバイはリードオンリー
  - データベースクラスタのほぼすべてのデータをレプリケーションできるが、特定のデータベースやテーブルだけをコピー（あるいはコピーしない）ということはできない
  - スタンバイはプライマリに昇格できるので、これを利用してプライマリ故障に対応する
  - 異なるメジャーバージョンのPostgreSQL間ではレプリケーションできない
  - 本日はこちらを前提にお話します

PostgreSQLにおける レプリケーション技術(2)

- ロジカルレプリケーション(PostgreSQL 10以降)
  - 歴史が浅く、まだあまり実績はない
  - 「論理」レプリケーションの一種
  - DB操作の論理情報を含むトランザクションログレコードをリアルタイムで一個のパブリッシャから複数のサブスクライバに転送、サブスクライバはログを再生(リプレイ)してデータをコピーする
  - データ更新はパブリッシャでもサブスクライバでも可能。ただし、更新の衝突はユーザが自己責任で解消する
  - 指定したテーブルのみコピーできる
  - 一部レプリケーションできない操作がある(たとえば DDL)
  - 異なるメジャーバージョンのPostgreSQL間でもレプリケーションできる
    - 当然のことながら、PostgreSQL 10以降のメジャーバージョンに限る

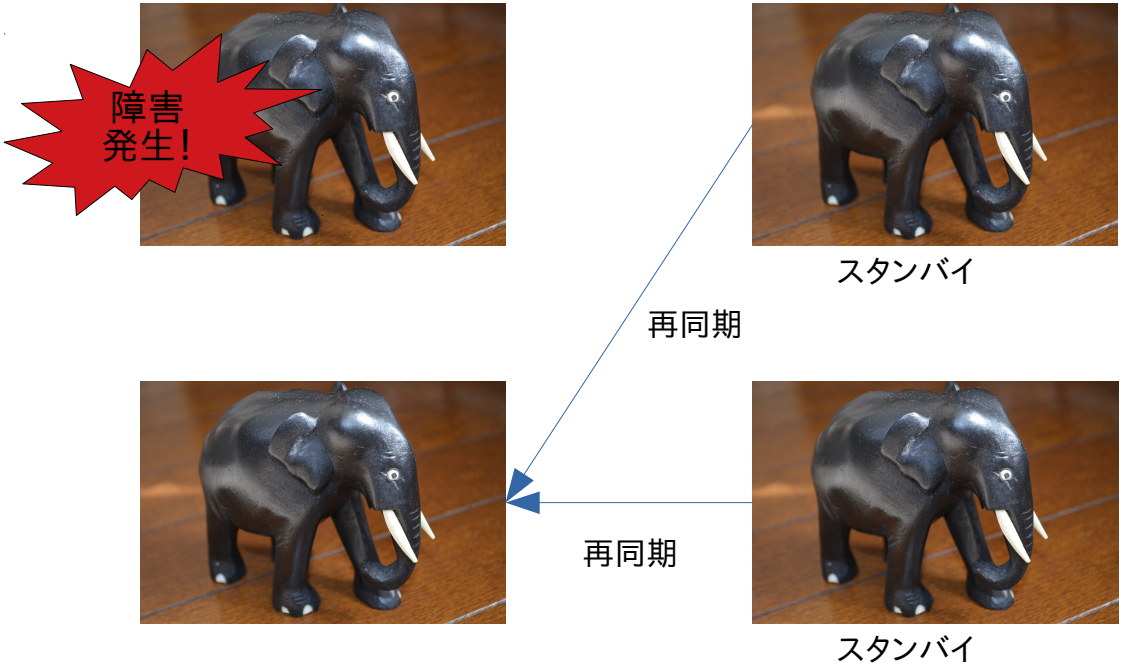
ストリーミングレプリケーションの仕組み



SRA OSS, INC.

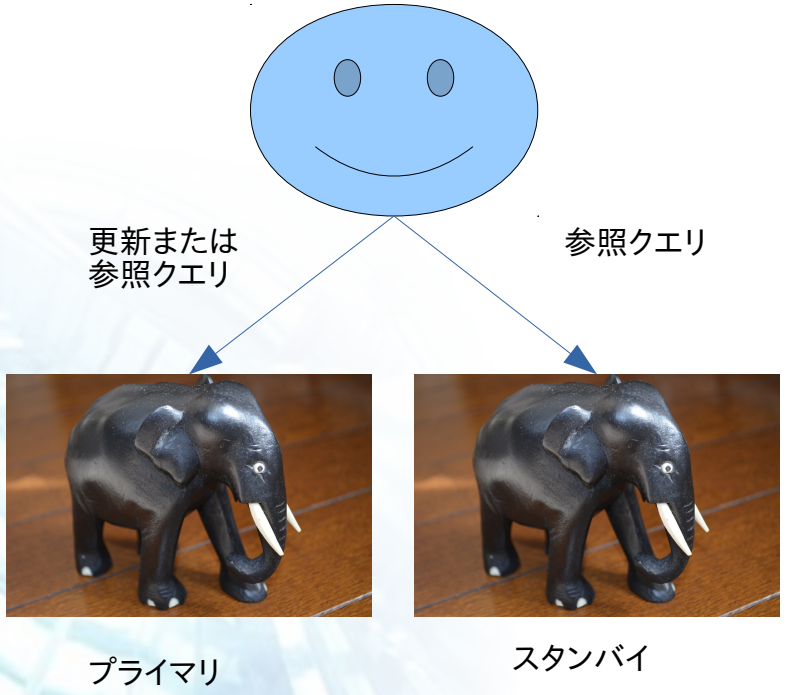
ストリーミングレプリケーション クラスタ管理の難しさ(1)

- 障害発生後のサーバ昇格は手動で実施しなければならない
- プライマリサーバが切り替わったら、スタンバイを新プライマリに手動で再同期する必要がある



ストリーミングレプリケーション クラスタ管理の難しさ(2)

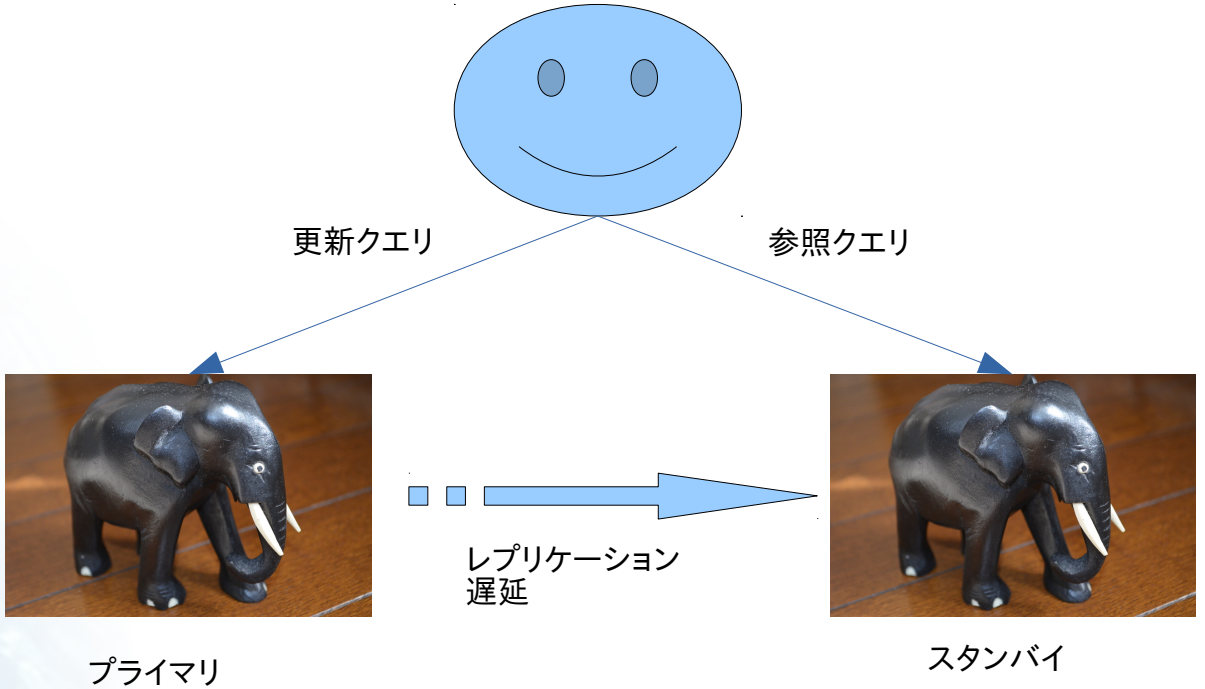
- スタンバイサーバには更新クエリを送ることができないので、アプリケーションの変更が必要かもしれない。その他にもスタンバイが受け付けないクエリがある
- クエリはすべてプライマリに投げる、という解決方法もあるが、検索負荷分散のメリットが失われる



- 一時テーブル
- 強いロック
- 2相コミット
- シーケンス操作
- LISTEN/NOTIFY
- 明示的な更新トランザクション宣言
  - SET TRANSACTION READ WRITEなど

ストリーミングレプリケーション クラスタ管理の難しさ(3)

- プライマリで発生した更新がスタンバイに届くまでに予想外の遅延(レプリケーション遅延)が発生することがある。たとえばマスターデータを更新したあと、直後に別セッションでスタンバイを参照すると、まだその更新を読み出せないことがある
- 原因は、ネットワーク帯域幅不足、スタンバイサーバのスペック不足など

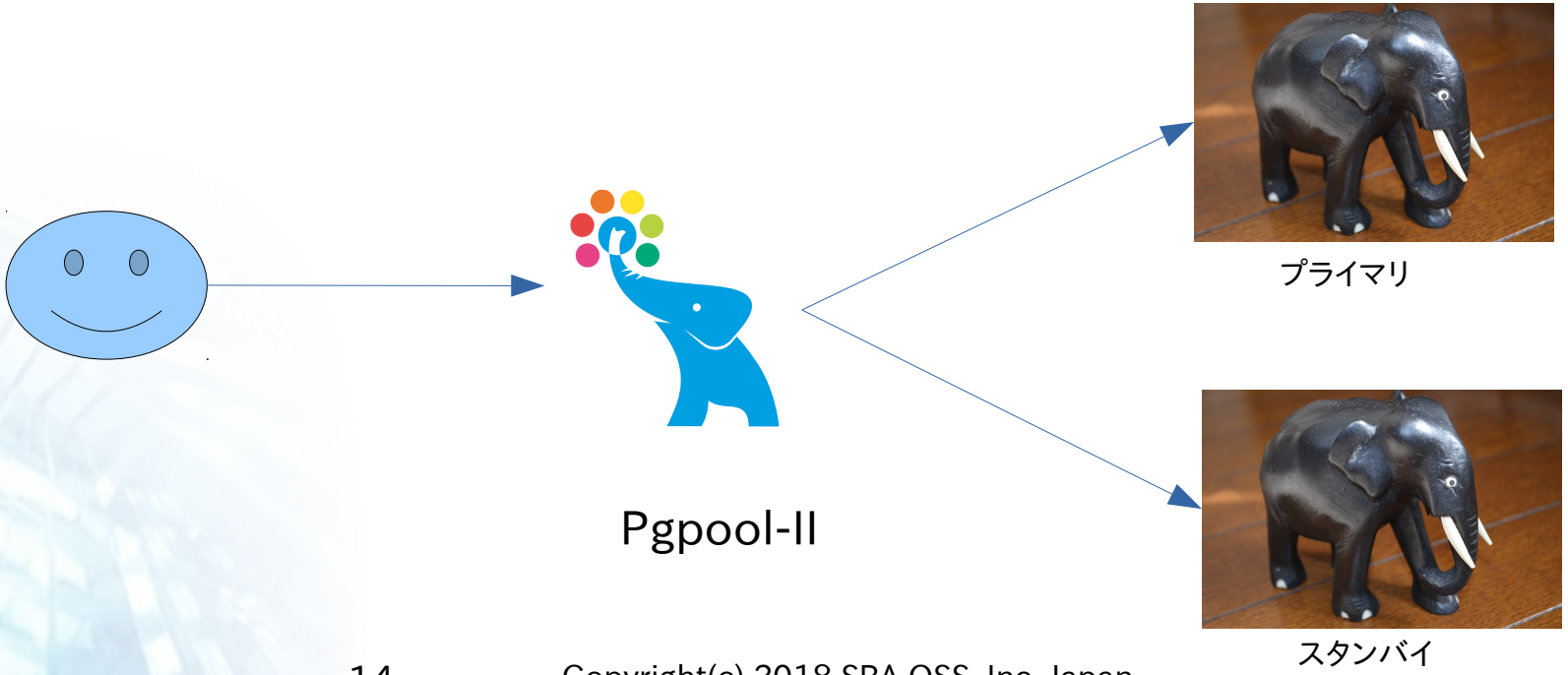


Pgpool-IIを使って問題を解決

- Pgpool-IIを使って解決ないし問題を軽減できるもの
  - スタンバイに投げても良いクエリは？
    - SQLパーサを内蔵し、クエリの種類を細かく識別してスタンバイに投げるクエリを自動決定
  - レプリケーション遅延への対応
    - レプリケーション遅延を随時モニター。遅延が大きい場合はクエリをスタンバイに投げない
  - 手動フェイルオーバ問題
    - バックエンドを監視して、ダウン時にフェイルオーバスクリプトを起動
  - 新プライマリへの同期追随問題
    - フェイルオーバ後にスタンバイを同期するスクリプトを定義できる

Pgpool-IIとは

- クライアントと(複数の)PostgreSQLの間に入るプロキシサーバ
- PostgreSQLをできるだけ透過的に単一サーバに見せる
- クラスタ管理の一部を自動化
- OSSとして配布
- PostgreSQLと同様、年に一度メジャーバージョンアップ
- 現在Pgpool-II 4.0を開発中(実は本日beta1をリリース予定)
- 詳細は https://www.pgpool/net をご覧ください。



スタンバイに送信しても良いクエリの判断

- 設定ファイルで各PostgreSQLに送る検索クエリの比率を設定できる
  - 通常は、各サーバ同一比率
    - backend_weight0 = 1.0
    - backend_weight1 = 1.0
    - Backend_weight2 = 1.0
- Pgpool-IIに接続したセッションで、上記比率に基づいて乱数で検索クエリを送信するPostgreSQLを決定(「ロードバランスノード」と呼ぶ)
- 以後、可能であればセッション終了まで検索クエリをロードバランスノードに投げる

スタンバイに送信しても良いクエリの判断

- SQLパーサで解析、判断
- パーサだけで判断できないものはシステムカタログに問い合わせ
  - 毎回問い合わせていると遅くなるので、キャッシュを内蔵
- 注意を要する問い合わせ
  - 一時テーブル
  - 強いロック
  - 2相コミット
  - シーケンス操作
  - LISTEN/NOTIFY
  - 明示的な更新トランザクション宣言
  - SET TRANSACTION READ WRITEなど

検索負荷分散のオプション調整パラメータ(1)

- black_function_list, white_function_list
  - 副作用のある関数(内部で更新を実施する関数)を呼び出すSELECTを登録できる
    - 例: nextval()
- database_redirect_preference_list
  - 特定のデータベースに対して、検索クエリの送信先を指定できる
    - postgres:primary(0.3),mydb[0-4]:standby(0.7)
    - ()の中に重みの比率を書けるのは、4.0の新機能
- app_redirect_preference_list
  - 特定のアプリケーション名に対して、同様のことが可能

検索負荷分散のオプション調整パラメータ(2)

- black_query_pattern_list
  - 常にプライマリに送信するクエリを登録できる(正規表現利用可能)
    - 特定のテーブルに対するSELECTは常にプライマリに送信したい場合などに利用
  - Pgpool-II 4.0の新機能
- allow_sql_comments
  - onの場合、SQLコメントをSQLの先頭に書くと検索クエリがプライマリに送られる
    - /*NO LOAD BALANCE*/SELECT 1;

検索負荷分散のオプション調整パラメータ(3)

- disable_load_balance_on_write
  - 更新クエリが表れたときの挙動を指定できる
  - そのセッション内で行われた更新を、続く検索クエリでできるだけ最新データを参照したい場合に利用
  - アプリケーションをできるだけ修正せずにクラスタ環境に移行したいときに利用
    - off
      - この機能を使わない
    - transaction
      - 明示的なトランザクション内で更新クエリが表れたら、トランザクション終了まで検索クエリをプライマリに送信
    - trans_transaction
      - “transaction”と同様だが、そのトランザクションが終了して、次のトランザクションが開始しても検索クエリをプライマリに送信し続ける
    - always
      - 更新クエリが表れたら、以後セッション終了まで検索クエリをプライマリに送信
  - offでは検索負荷分散が最大の効果を発揮するが、旧来アプリケーションとの互換性は最小、alwaysは検索負荷分散の効果が最小だが、旧来アプリケーションとの互換性は最大
  - Pgpool-II 4.0の新機能

レプリケーション遅延への対応

- sr_check_period
  - レプリケーション遅延を測定する時間間隔を秒単位で指定
- delay_threshold
  - レプリケーション遅延がここで指定したバイト数より大きければ、そのスタンバイをロードバランスノードに選ばない
- レプリケーション遅延の検知はリアルタイムではないことに注意!

手動フェイルオーバ問題の解決

- Pgpool-IIが定期的にPostgreSQLの状態をチェックし、問題があればユーザ定義のフェイルオーバスクリプトを起動することによって、自動フェイルオーバを実現できる

フェイルオーバスクリプトの例（pgpool_setupが生成したスクリプトの一部)

```
#! /bin/sh
# Execute command by failover.
# 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
#                  %R = new master database cluster path
#                  %r = new master port number
#                  %% = '%' character

[中略]

if [ a"$failed_node_id" = a"$old_primary_node_id" ];then        # master failed
!           new_primary_db_cluster=${mydir}/data"$new_master_id"
            echo $pg_ctl -D $new_primary_db_cluster promote >>$log    # let standby take over
            $pg_ctl -D $new_primary_db_cluster promote >>$log         # let standby take over
            sleep 2
fi
```

新プライマリへの同期追随問題

- フェイルオーバ完了後、新プライマリへの同期追随を行うスクリプトを“follow_master_command”でスクリプトを設定できる

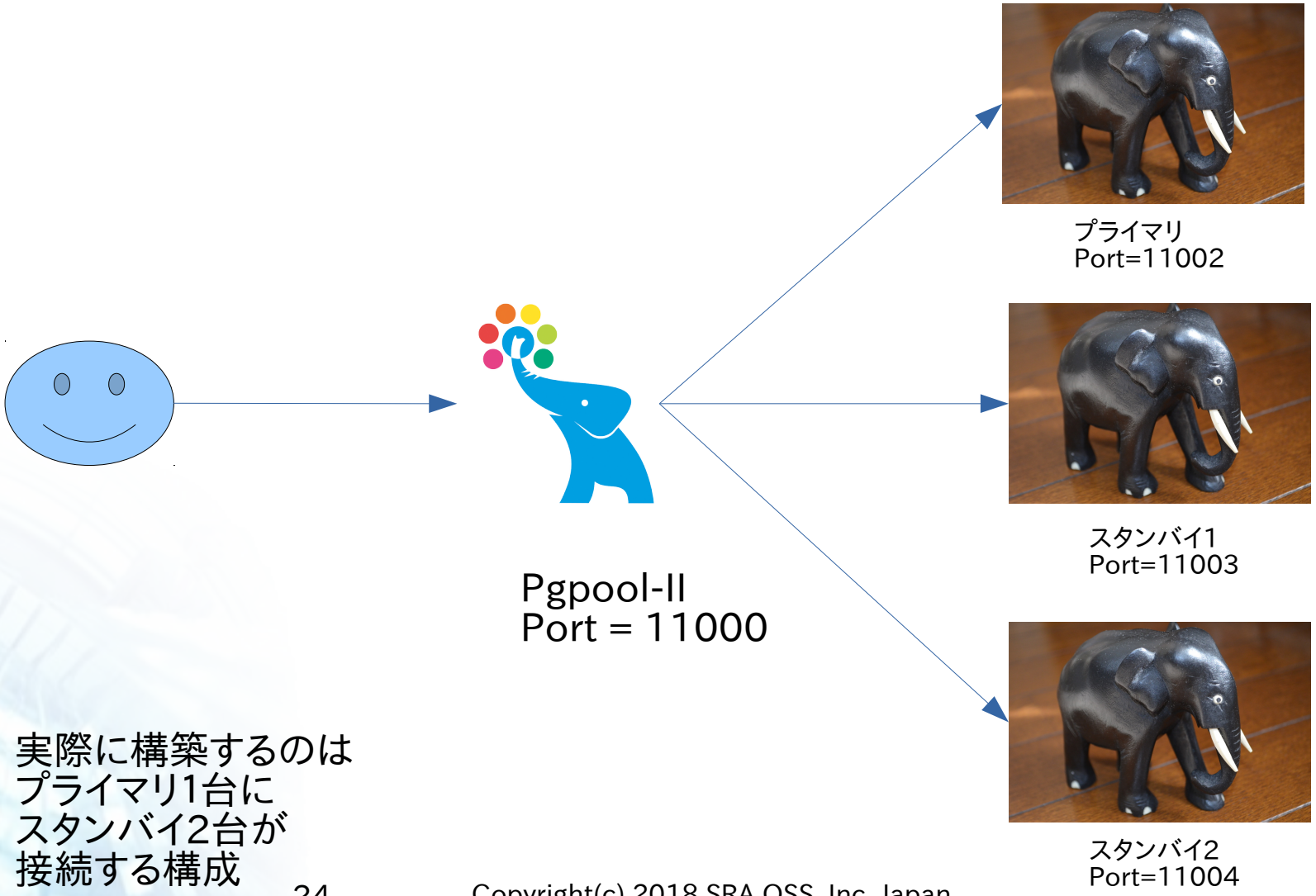
Follow_master_commandの例（pgpool_setupが生成したスクリプトの一部)

```
#! /bin/sh
# Execute command by failover.
# 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
#                  %R = new master database cluster path
#                  %r = new master port number
#                  %% = '%' character


[中略]

# Stop standby node if it's running
if $pg_ctl -D $failed_db_cluster status|grep "is running" >/dev/null 2>&1
then
        $pg_ctl -w -m f -D $failed_db_cluster stop >> $log 2>&1
        sleep 10
        # recovery the node
        pcp_recovery_node -w -h localhost -p $PCP_PORT -n $failed_node_id >> $log 2>&1
else
        echo "$failed_db_cluster is not running. skipping follow master command." >> $log
fi
```

デモタイム

- シナリオ
  - 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

Pgpool-II付属ツールを使って実験環境構築(1)



実際に構築するのは
プライマリ1台に
スタンバイ2台が
接続する構成

Pgpool-II付属ツールを使って実験環境構築(2)

```
$ pgpool_setup -n 3
PostgreSQL major version: 10
Starting set up in streaming replication mode
creating startall and shutdownall

[中略]


temporarily start data0 cluster to create extensions
temporarily start pgpool-II to create standby nodes
 node_id | hostname | port  | status | lb_weight |  role   | select_cnt | load_balance_node | replication_delay | last_status_change
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------+---------------------
 0       | /tmp     | 11002 | up     | 0.333333  | primary | 0          | true              | 0                 | 2018-09-18 11:18:08
 1       | /tmp     | 11003 | down   | 0.333333  | standby | 0          | false             | 0                 | 1970-01-01 09:00:00
 2       | /tmp     | 11004 | down   | 0.333333  | standby | 0          | false             | 0                 | 1970-01-01 09:00:00
(3 rows)

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 | last_status_change
---------+----------+-------+---------+-----------+---------+------------+-------------------+-------------------+---------------------
 0       | /tmp     | 11002 | up      | 0.333333  | primary | 0          | false             | 0                 | 2018-09-18 11:18:08
 1       | /tmp     | 11003 | up      | 0.333333  | standby | 0          | true              | 0                 | 2018-09-18 11:18:13
 2       | /tmp     | 11004 | waiting | 0.333333  | standby | 0          | false             | 0                 | 2018-09-18 11:18:22
(3 rows)

shutdown all

pgpool-II setting for streaming replication mode is done.
To start the whole system, use /home/t-ishii/testdb/startall.
To shutdown the whole system, use /home/t-ishii/testdb/shutdownall.
pcp command user name is "t-ishii", password is "t-ishii".
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

- PostgreSQLのストリーミングレプリケーションを使ったクラスタ構成の概要と問題点を説明しました
- そして、Pgpool-IIによる問題の解決方法の提案を行いました
- Pgpool-IIは非常に多機能なソフトで、今回ご説明できなかった機能が他にもたくさんあります
  - たとえば、Pgpool-IIを単一障害点にしないための「watchdog」、オンメモリクエリ結果キャッシュなど
- 更なる情報
  - Pgpool-IIのウェブサイトには、ほぼプロダクションで使えるレベルの構成例が載っています
  - SRA OSSでは、Pgpool-IIでクラスタを組むハンズオントレーニングも実施しています。ぜひご活用ください

宣伝

- 来週の火曜日(9/25)に、「Pgpool-II Day」を開催します
  - Pgpool-II生誕15周年を記念して
  - Pgpool-II開発者（海外も含む)による技術講演
  - 間もなくリリース予定のPgpool-II 4.0の説明
- SRA OSS, Inc.日本支社のページ →「イベント、セミナー一覧」→「Pgpool-II 4.0 Anniversary」
  - https://www.sraoss.co.jp/event_seminar/2018/0925.php
- たまにしかやらないイベントなので、ぜひご参加ください！

Thank you!