PGECons
PostgreSQL Enterprise Consortium

2022年度WG3活動成果報告
運用課題解決編
PgBouncer性能評価

PostgreSQL エンタープライズコンソーシアム
WG3 運用課題解決検証チーム

Contents

1. 本検証の目的
2. PgBouncerの仕様・使いどころ
3. PgBouncer性能検証
4. まとめ

責任範囲

- 本資料は、PGEConsが独自に検証した結果であり、結果はPGEConsの責任の元、公開しています。

1. 本検証の目的

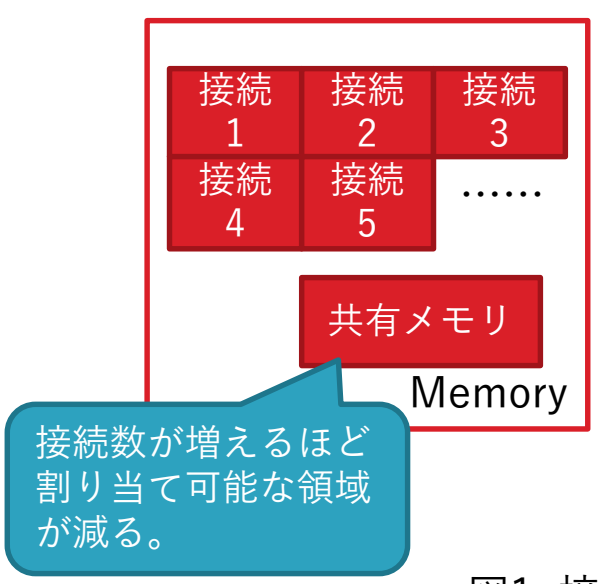
課題: 接続数増加により性能劣化を引き起こす

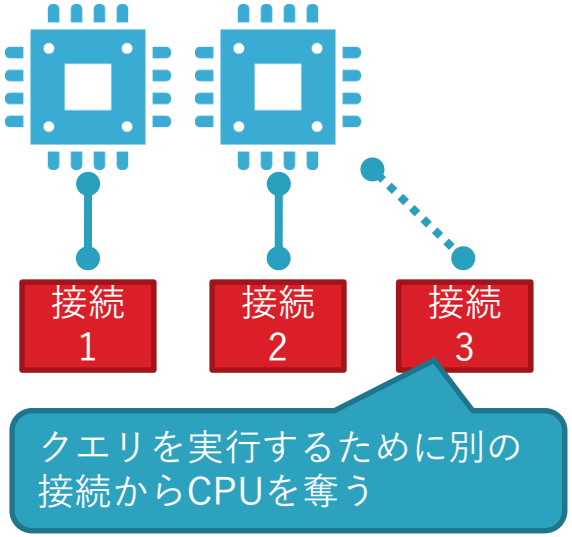
- 接続数が増加し、同時にクエリが実行されると
  次の理由から性能劣化を引き起こす状況が発生する

キャッシュヒット率低下
接続に割り当てられる
メモリによってバッファ
領域が逼迫される。

コンテキストスイッチ多発
CPUコア数を超える接続
により、CPUの奪い合い
が発生する。

接続時のオーバーヘッド発生
接続/切断が多発することで
リソースへの負荷が増える。





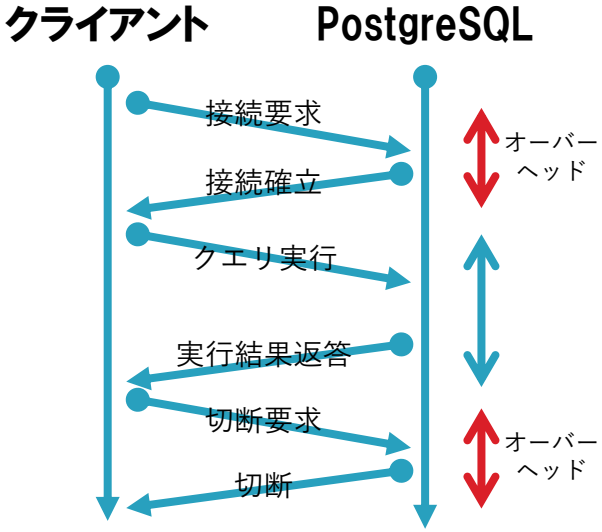


図1: 接続数増により性能劣化を引き起こす状況

解決策例: コネクションプーリング導入

- クライアントとDBサーバの中間に立ち、DBとのコネクションを維持し、かつコネクション数を一定数とすることで課題を解決する
- PostgreSQLの代表的なプーリングツールとして、PgBouncerが存在する

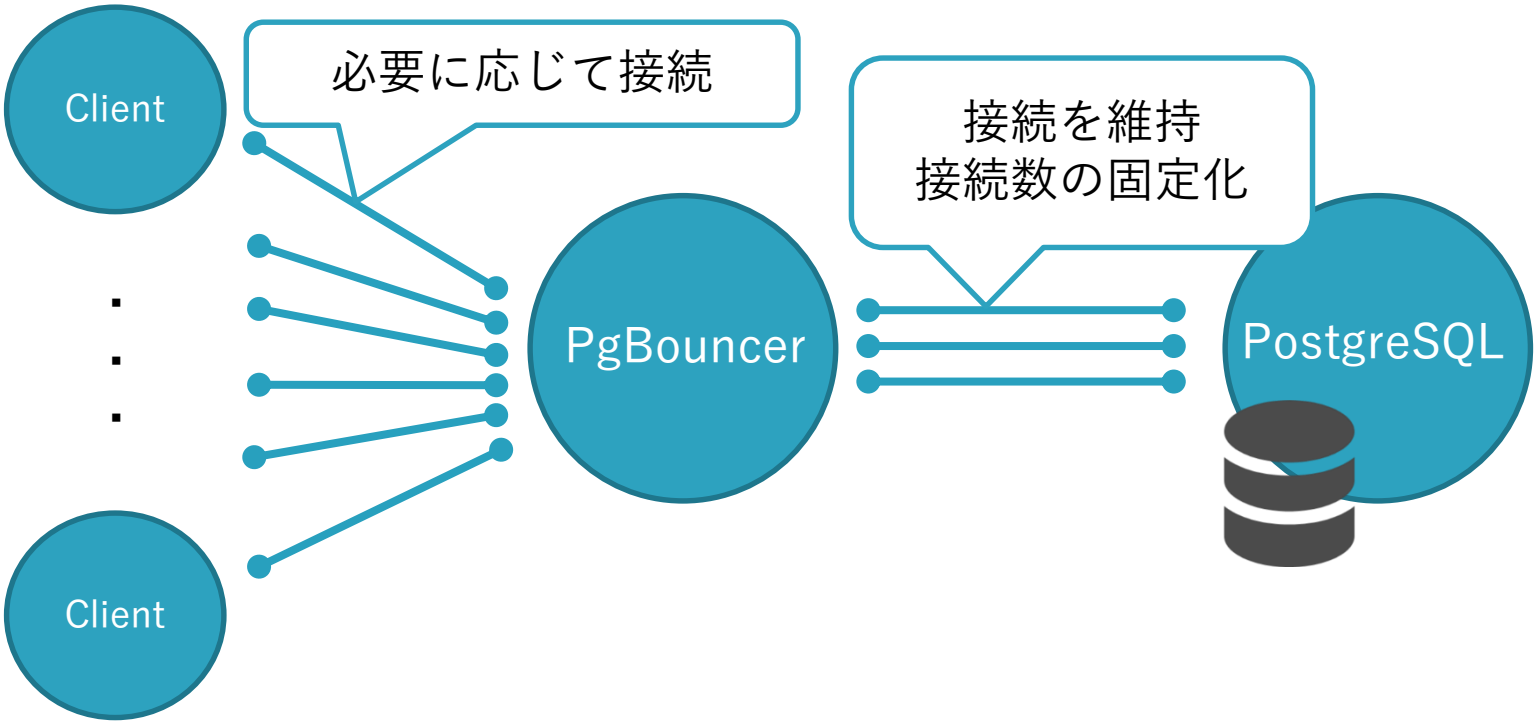


図2: コネクションプールのイメージ

本検証の目的

- PgBouncerを導入するシチュエーション、
  また性能検証を通じて導入時の注意事項を報告する

2. PgBouncerの仕様・使いどころ

用意されるプーリングモード

- プールされたコネクションを、クライアントに払い出す/戻すタイミング別に、3つのプーリングモードが用意されている
  - **セッションプーリング**
    - デフォルトのプーリングモード
    - クライアントからのトランザクション要求時にプールしているコネクションを払い出す。接続要求時はPgBouncerに接続されるがコネクションは払い出されない
    - クライアントからの切断要求時にコネクションがプールに戻される
    - PostgreSQLへの直接接続と同じ挙動となり、PgBouncerを意識したアプリケーションの設計は必要ない。使用を推奨。
  - **トランザクションプーリング**
    - トランザクション開始時にプールしているコネクションを払い出す
    - トランザクション終了時にコネクションがプールに戻される
    - セッションごとに管理されるPostgreSQLの機能が、期待通りに動作しない可能性があるため、これを意識したアプリケーション設計とする必要がある。
      ※ 次ページにて、影響を与えるアプリケーションについて記載
  - **ステートメントプーリング**
    - 発行するクエリ1文ごとプールしているコネクションの払い出し/戻しを行う
    - オートコミットが強制されるため、業務システムでの使用は推奨されない。

トランザクションプーリング時に注意が必要な機能

- セッション毎に管理される機能について、トランザクションプーリング時にコネクションが切り替わり意図せぬ動作となる可能性があるため、注意が必要

表1: セッションごとに管理される機能一覧

機能	説明
SET/RESET	実行時パラメータの変更。SET SESSIONの場合に影響が発生。 SET LOCALの場合、トランザクション内での変更のため影響しない。
LISTEN/NOTIFY	通知を監視する。セッション単位で設定される。
WITH HOLD CURSOR	トランザクションが正常にコミット処理を行った後も、そのカーソルの使用を続けられることを指定する。
PREPARE/DEALLOCATE	実行する文を準備する。準備された文はセッションごとに保持される。
PRESERVE/DELETE ROWS temp tables	一時テーブルのデータ保持期間。デフォルトではセッション単位。
LOAD statement	共有ライブラリファイルの読み込みを行う。読込先はセッションごとのメモリ。
Session-level advisory locks	セッションレベルの勧告的ロック。

https://www.pgbouncer.org/features.html

PgBouncerの挙動

- PgBouncerが持つプールサイズを超える数のクライアント接続が要求された場合
  - 現在使用しているコネクションが解放され、割り当てられるまで待つ
  - max_client_connを超えるクライアント接続は、即時エラーとなる

```
psql: error: connection to server at "localhost" (127.0.0.1), port 6432 failed: FATAL:  no more connections allowed (max_client_conn)
```

- セッションスコープの設定値を変更した場合
  - 変更した内容が、次に割り当てられたクライアント先でも有効となってしまう
  - 他のコネクションプールツールでも同じ挙動であり、
    プールに戻す際は設定を戻す必要あり

Pgpool-IIとPgBouncerの違い

- PostgreSQL向けコネクションプーリングツールは PgBouncerの他にPgpool-IIが存在する
- PgBouncerはコネクションプールに特化したツールだが、 Pgpool-IIはクラスタ管理機能など豊富な機能を持つ

表2: Pgpool-IIとPgBouncer比較

	PgBouncer	Pgpool-II
サーバ負荷	単一プロセスで稼働するため オーバヘッドは少ない。	接続に応じて子プロセスを作成するため PgBouncerに比べてオーバヘッドが大きい。
プーリング モード	セッションプーリング、 トランザクションプーリング、 ステートメントプーリングの3種類。	セッションプーリングのみ。
データベース 高可用性	接続先となるデータベースサーバは 1台のみで、DBの冗長化には 対応していない。	複数のデータベースサーバを登録し、 サーバ間のレプリケーションや負荷分散 などの機能を持つ。
ツール 高可用性	冗長化の機能はないため、 単一障害点になりやすい。	複数のPgpool-IIサーバを 冗長化させることができる。

使用が推奨されるシチュエーション

- PgBouncerは、以下のようなシチュエーションでの利用が想定される
  - **アプリケーション側でコネクションプール機能を持たないシステム**
    - クライアントサーバ型のシステム、CGIなど
    - 導入効果: 接続オーバーヘッドの軽減・同時実行数を一定とすることによる性能劣化の軽減
  - **複数のアプリケーションが1つのDBを共有している**
    - 共有のプールを使用する事で、DBへの接続数を一定にし、かつアプリケーションの利用に応じてDBコネクションを有効に利用できる（下図参照）

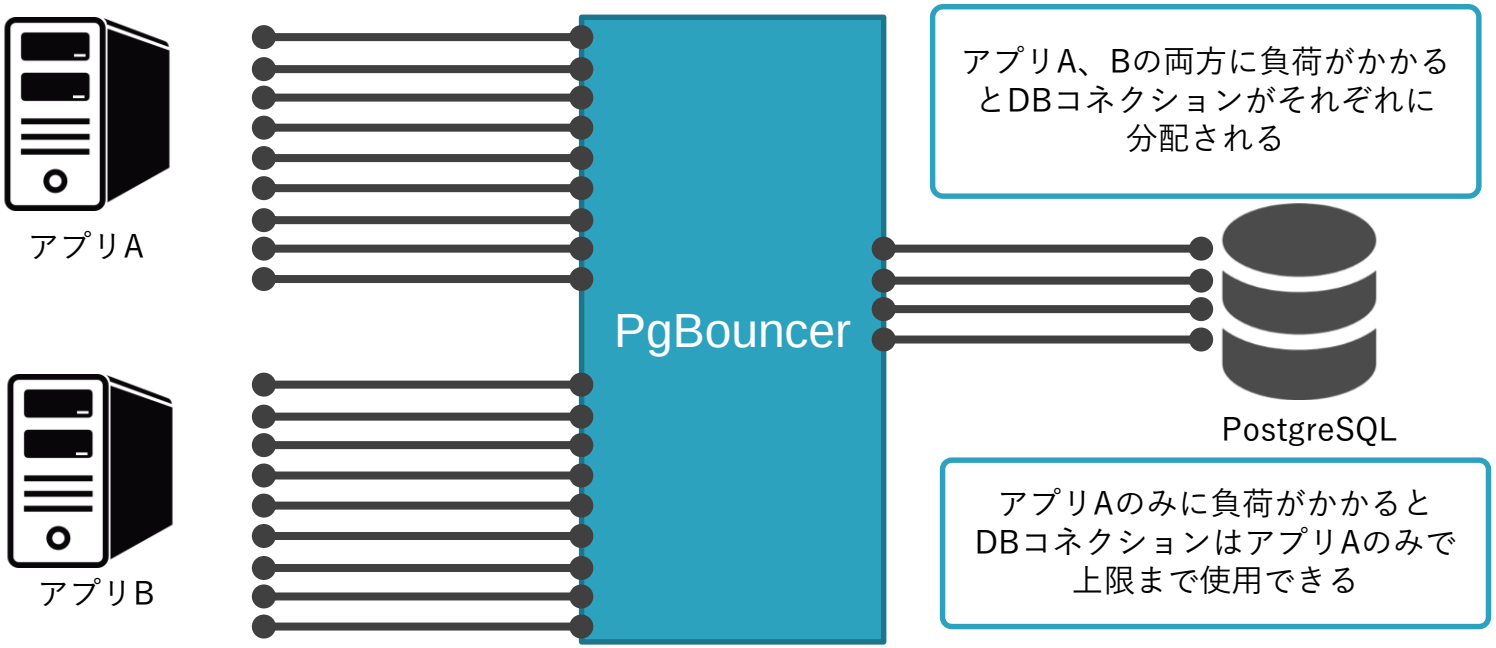


図3: 複数アプリケーションがDBを共有するイメージ

PgBouncer性能検証

検証目的 – 想定されるメリットとデメリット

- PgBouncer利用によるメリット / デメリットのうち
  性能に関する項目として以下を想定している
  - **1.メリット**
    - 1.1. 接続 / 切断に関わるオーバーヘッドの省略
    - 1.2. PostgreSQLのコネクション数を一定にし、同時実行数が大量となった場合の性能劣化軽減
  - **2.デメリット**
    - 2.1. コネクションプールを経由する事によるオーバーヘッドの発生
    - 2.2. 暗号通信時の暗号化 / 復号化処理が2回発生

これらがどの程度の影響となるか、実機にて検証し評価する

検証サマリ

表3: 検証サマリ一覧

CASE	メリット/デメリット	確認内容	検証方法	結果
1	メリット	接続 / 切断に関わるオーバーヘッドの省略	接続/切断をトランザクション毎に実施しPgBouncerあり/なしの結果を比較する	メリット大
2	メリット	PostgreSQLのコネクション数を一定にし、同時実行数が大量となった場合の性能劣化軽減	コア数を超える同時実行数を発生させ、PgBouncerあり/なしの性能劣化割合を比較する	メリット大
3	デメリット	コネクションプールを経由する事によるオーバーヘッドの発生（PgBouncer処理の劣化）	接続/切断を検証前後に1回だけ実施しPgBouncerあり/なしの結果を比較する	デメリット小
4	デメリット	コネクションプールを経由する事によるオーバーヘッドの発生（PgBouncer接続処理による劣化）	PgBouncerあり環境に対して、接続/切断をトランザクション毎と検証前後に1回実施した結果を比較する	デメリット中
5	デメリット	暗号通信時の暗号化 / 復号化処理が2回発生	クライアントからPostgreSQLまでの通信経路を暗号化し、PgBouncerあり/なしの結果を比較する	デメリット 小～中
6	-	[番外編: 動作確認] バースト転送時の性能劣化	PgBouncerあり時にバースト転送されることを確認する	デメリット 小～中

検証環境

- **マシンスペック**
  - CPUコア数: 4
  - メモリ: 16GB
  - OS: Ubuntu 20.04.5 LTS (5.4.0-91-generic)
  - PostgreSQL: 15.1
  - PgBouncer: 1.18.0
  - pgbench: 15.1
- **検証環境構成**
  - pgbench、PostgreSQL、PgBouncerを同一ノード上に構成
    - 今回の検証環境で各要素を別ノードに構成したところ、ネットワークレイテンシが大きく、PostgreSQLのサーバーリソースを使い切る負荷をかけることができなかったため、同一ノード上に構成している
- **検証方式**
  - 前述のメリット / デメリットについて、PgBouncerの影響を計測する。各検証ではPgBouncerを使用した場合 / 使用しない場合のレイテンシとスループットを評価する。

主な設定項目

表4: 各要素の主な設定

設定対象	設定値	補足
PostgreSQL	autovacuum = off	AutoVacuum発生抑制
	checkpoint_timeout = 1d	Checkpoint発生抑制
	max_wal_size = 30GB	Checkpoint発生抑制
	log_min_messages = panic	ログ出力抑制
PgBouncer	pool_mode = session	セッションプーリングを使用
	max_client_conn	検証時の同時接続数を設定
	min_pool_size	検証時の同時接続数を設定
	max_db_connections	検証時の同時接続数を設定
	max_user_connections	検証時の同時接続数を設定
pgbench	--protocol=simple	
	--time=300	
	--connect	接続/切断をトランザクション毎に実施する場合に指定。 接続/切断を検証の前後に1回実施する場合は指定しない。
	実行クエリ	1: BEGIN; 2: SELECT 1; 3: COMMIT;

※記載されていない項目はデフォルトの値とする

CASE1. 接続オーバーヘッド省略(概要)

- PgBouncerによる接続オーバーヘッド省略の効果を確認する
  - 1.1. 接続 / 切断に関わるオーバーヘッドの省略
- 検証設定
  - 接続/切断をトランザクション毎に実施する
  - 同時実行数を1とする
  - 暗号化通信なしとする
  - pgbenchを使用しクエリの性能を計測、PgBouncerのあり/なしで比較する
  - 検証に使用するクエリは、処理時間への影響を最小限に抑えるため、「BEGIN;」「SELECT 1;」「COMMIT;」とする

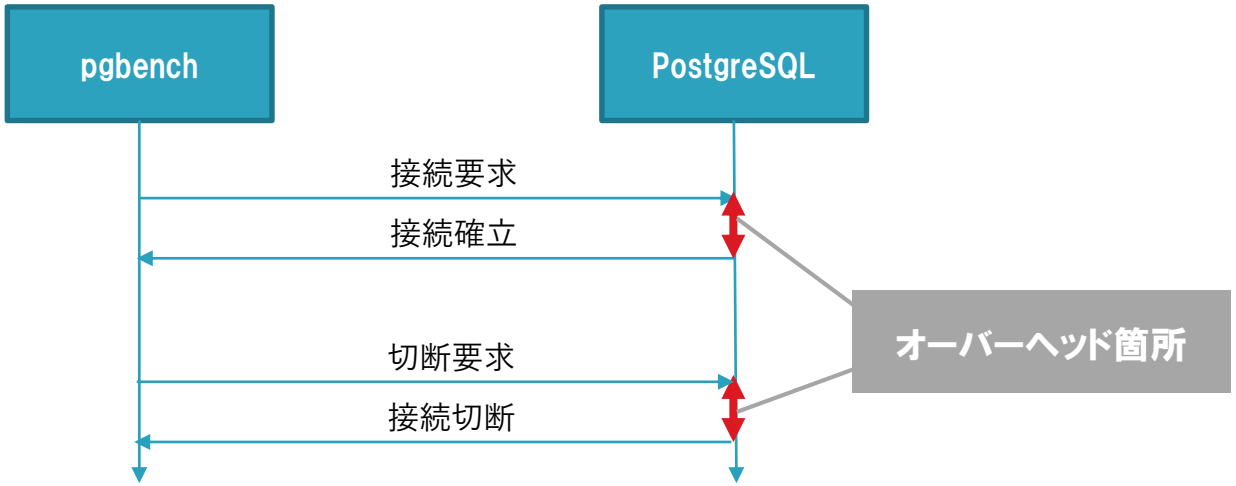


図4: PostgreSQL接続オーバーヘッド箇所

CASE1. 接続オーバーヘッド省略(結果)

- 評価: メリット大
  - 1接続あたり2.6msec程度を削減。ワークロードとして大きな削減につながる

表5: PostgreSQL接続オーバーヘッド省略検証結果

パターン	スループット(TPS)	処理時間(msec)※1
PgBouncerあり	795.8	1.257
PgBouncerなし	256.9	3.893
(あり-なし)	-	-2.636

※1: 接続開始/切断を含む

CASE2. 同時実行数増加(概要)

- 同時実行数の増加による、性能劣化を比較する
  - 1.2. PostgreSQLのコネクション数を一定にし、
    同時実行数が大量となった場合の性能劣化軽減
- 検証設定
  - PgBouncerなし: 接続/切断を検証の前後に1回実施とする
  - PgBouncerあり: 接続/切断をトランザクション毎に実施する
    - セッションプーリングのため試行毎に接続/切断を行う
  - PgBouncerのコネクションプール数は4とする
    - PgBouncerなしでの最大性能が同時実行数4であったため
  - 暗号化通信はなしとする
  - pgbenchを使用し、スループットを計測。同時実行数が少ない状態の
    スループットと多い状態のスループットを計測し、劣化した比率を算出する
  - 検証に使用するクエリは、処理時間への影響を最小限に抑えるため、
    「BEGIN;」「SELECT 1;」「COMMIT;」とする

CASE2. 同時実行数増加(結果①)

- 評価: 多重度の増加に対する性能劣化の面でメリット大
  ただし絶対的な性能が大幅に低下している。
  - 同時接続数増加時の劣化率を約86%⇒約97%程度に軽減。
  - ただしPgBouncerありの場合はPgBouncerなしと比べて
    スループットが約1/8に低下している。
  - CASE1から、PgBouncerなしで接続/切断をトランザクション毎に行い
    同時実行数を増加させた場合にはPgBouncerありよりも劣化すると推測
    - 今回は実施できなかったため、次の機会に確認したい

表6: 同実行数増加検証結果

パターン	最大TPS	同時実行数64でのTPS	劣化率（実行数64でのTPS /最大TPS）
PgBouncerなし（同時実行数: 4）	17,029.7	14,679.4	86.2%
PgBouncerあり（同時実行数: 4）	2,529.8	2,446.9	97.6%
PgBouncerあり（同時実行数: 8）	2790.4	2,721.9	97.5%

CASE2. 同時実行数増加(結果②)

- CPU使用時間を比較すると、PostgreSQLがCPUを使い切れていない
  - クライアント⇒PgBouncerの接続待ちが発生していると推測
  - プロセスごとのCPU使用時間を比較すると、PostgreSQLプロセスはPgBouncerありの方が大幅に短くなっている

表7: プロセスごとのCPU使用時間割合

プロセス	PgBouncerなし	PgBouncerあり
pgbench	42.0%	43.4%
PgBouncer	-	20.3%
PostgreSQL	56.7%	19.1%
合計	98.7%	82.8%

  - PgBouncerとPostgreSQLを同一ノードで稼働させた事により、PostgreSQLに十分なCPUリソースが割り当てられなかったと推測
  - 別ノードで稼働させることで性能の改善を確認したかったが、今回の環境ではネットワークレイテンシが非常に大きく、負荷を掛け切れなかった
    - ネットワークレイテンシが低い環境で、pgbench/PgBouncer/PostgreSQLを別ノードに構築した検証は、次の機会に実施したい

CASE3. PgBouncer内部処理影響(概要)

- PgBouncerが仲介する事でのオーバーヘッドを確認する
  - 2.1. コネクションプールを経由する事によるオーバーヘッドの発生
- 検証設定
  - 接続/切断は検証の前後に1回実施する
  - 同時実行数を1とする
  - 暗号化通信はなしとする
  - pgbenchを使用しクエリの性能を計測、PgBouncerのあり/なしで比較する
  - 検証に使用するクエリは、処理時間への影響を最小限に抑えるため、「BEGIN;」「SELECT 1;」「COMMIT;」とする

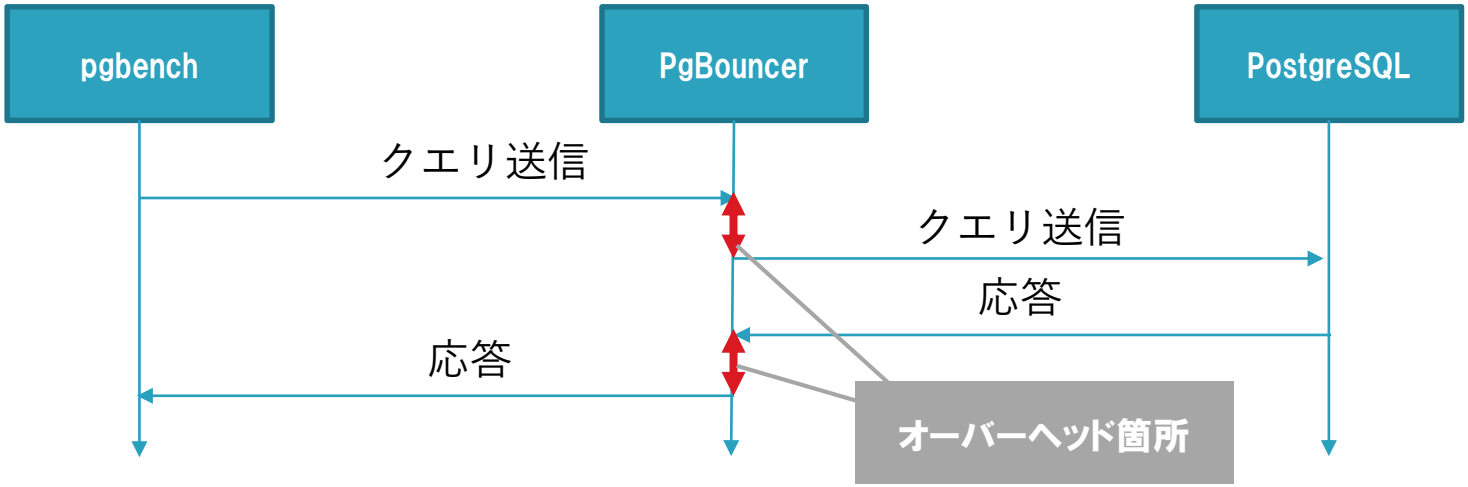


図5: PgBouncer内部処理のオーバーヘッド箇所

CASE3. PgBouncer内部処理影響(結果)

- 評価: デメリット小
  - 同一ノードにPostgreSQLとPgBouncerを構築した場合、1クエリ当たり、0.06msec程度のオーバーヘッドが発生する
  - ワークロードとして数ミリ秒程度のクエリであれば、問題ないと考える
  - PostgreSQLとPgBouncerを別ノードに構築する場合、ネットワークレイテンシがかかることを考慮する必要がある

表8: PgBouncer内部処理計測結果

パターン	スループット(TPS)	処理時間(msec)	標準偏差	クエリ1 (BEGIN;)	クエリ2 (SELECT 1;)	クエリ3 (COMMIT;)
PgBouncerあり	3,216.2	0.310	0.096	0.093	0.118	0.099
PgBouncerなし	7,921.6	0.126	0.019	0.034	0.053	0.038
(あり-なし)	-	0.184	-	0.059	0.065	0.061

CASE4. PgBouncer接続オーバーヘッド(概要)

- PgBouncerに対する接続/切断オーバーヘッドの影響を確認する
  - 2.1. コネクションプールを経由する事によるオーバーヘッドの発生
- 検証設定
  - 接続/切断をトランザクション毎と、検証の前後に1回の2パターンで実施する
  - 同時実行数を1とする
  - 暗号化通信なしとする
  - pgbenchを使用しクエリの性能を計測する
  - 検証に使用するクエリは、処理時間への影響を最小限に抑えるため、「BEGIN;」「SELECT 1;」「COMMIT;」とする

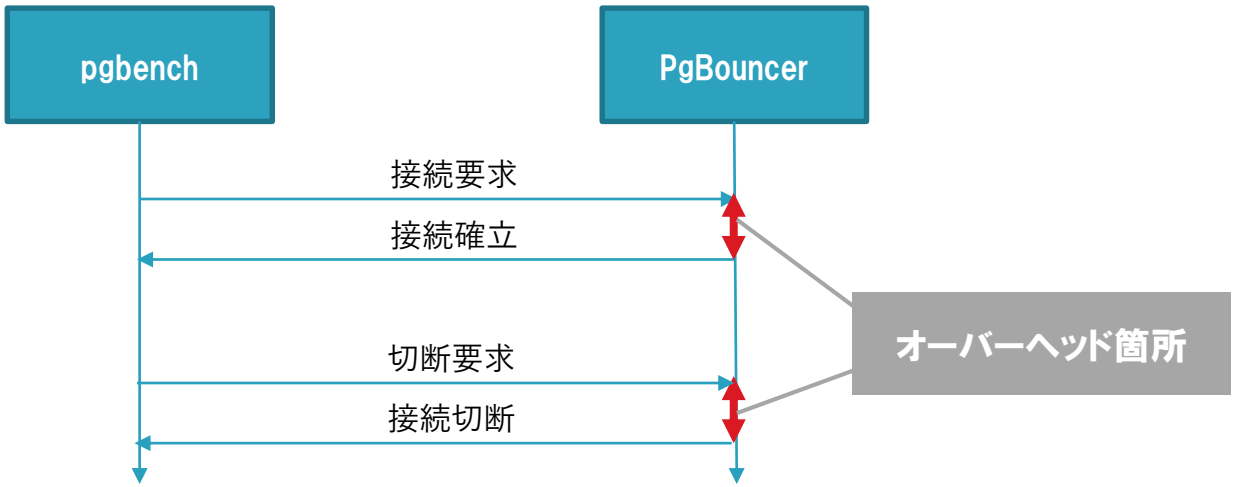


図6: PgBouncer接続オーバーヘッド箇所

CASE4. PgBouncer接続オーバーヘッド（結果）

- 評価: 影響中
  - 0.946msecの接続オーバーヘッドがあることを確認
  - PostgreSQLの接続オーバーヘッド（2.6msec）より少なく済んでいる
  - クエリ1、クエリ2がCASE1より時間がかかっている
    - クエリ1（BEGIN;）は、この箇所でコネクションの割り当てが行われることの影響
    - クエリ2（SELECT 1;）は、CPUキャッシュの効果を推測している（確証を得る検証は未実施）

表9: PgBouncer接続オーバーヘッド検証結果

パターン	スループット（TPS）	処理時間※1（msec）	クエリ1 (BEGIN;)	クエリ2 (SELECT 1;)	クエリ3 (COMMIT;)
試行前後に接続/切断	3,216.2	0.311	0.093	0.118	0.099
試行毎に接続/切断	795.8	1.257	0.281	0.207	0.161
（差分）	-	0.946	0.188	0.089	0.062

※1: 接続開始/切断を含む

CASE5. 暗号通信時の性能劣化(概要)

- クライアントとPostgreSQLの間を暗号化する場合、PgBouncerを使用すると2度の暗号化/復号化処理が発生する
  - 2.2. 暗号通信時の暗号化 / 復号化処理が2回発生
- 検証設定
  - 接続/切断を検証の前後に1回実施する
  - 同時実行数を1とする
  - 暗号化通信ありとする
  - pgbenchを使用しクエリの性能を計測する
  - 検証に使用するクエリは、処理時間への影響を最小限に抑えるため、「BEGIN;」「SELECT 1;」「COMMIT;」とする

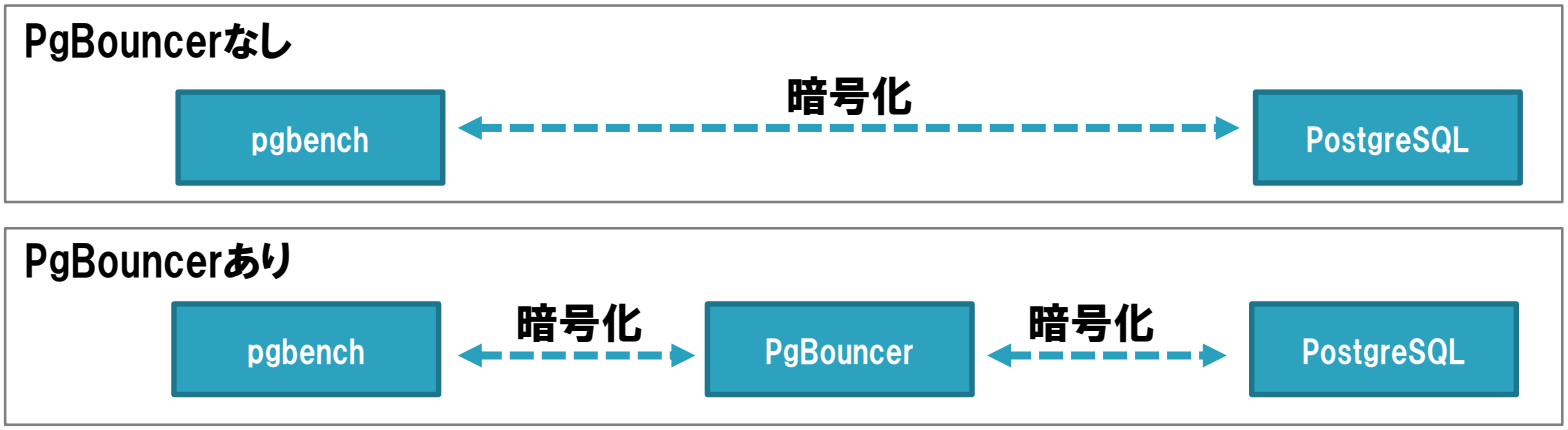


図7: 通信経路暗号化箇所

CASE5. 暗号通信時の性能劣化(結果)

- 評価: デメリット小～中
  - 今回の検証では1クエリあたり0.09msec程度劣化した
  - 本検証ではクエリによる結果セットが極小のため大きな劣化とはならなかったが、大量のデータをやり取りする場合には暗号化/復号化の影響が大きくなる
    - 論理バックアップ取得/論理バックアップからのリカバリなど
  - 実際の業務システムで利用する際には、該当する処理にて許容できる処理時間であることの検証を推奨する

表10: 暗号化通信時の性能劣化検証結果

パターン	スループット（TPS）	処理時間（msec）	標準偏差	クエリ1 (BEGIN;)	クエリ2 (SELECT 1;)	クエリ3 (COMMIT;)
①PgBouncerあり 暗号化有効	2,237.5	0.448	0.252	0.139	0.165	0.144
②PgBouncerなし 暗号化有効	5,611.2	0.178	0.028	0.051	0.071	0.055
①-②	-	0.270	-	0.088	0.094	0.089

CASE6. バースト転送時の性能劣化(概要①)

- TCP/IP通信では、複数のパケットにまとめて1つのACKを返す転送方式（バースト転送機能）があり、PostgreSQLでは実現されている
- バースト転送により大容量のクエリ実行結果を効率的にパケットをやり取りすることができるが、PgBouncerを利用する場合でも行われるかを検証する
  - メリット/デメリットとして挙げていないが動作検証として実施

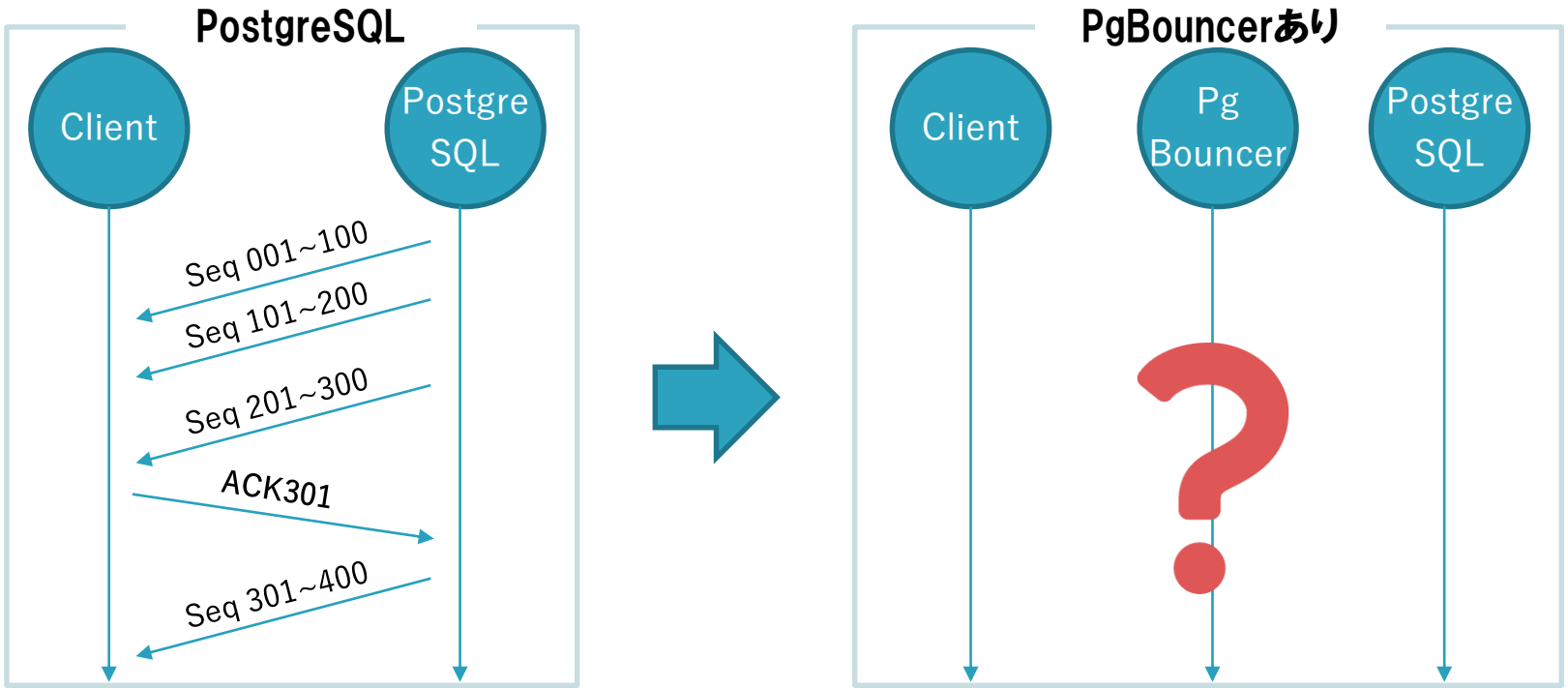


図8: バースト転送イメージ

CASE6. バースト転送時の性能劣化(概要②)

- 検証設定
  - 接続/切断は検証の前後に1回実施する
  - 同時実行数を1とする
  - 暗号化通信なしとする
  - psqlを使用しクエリを実行し、
    tcpdumpを使用してパケットフローを確認する
  - 検証に使用するクエリは、大量の結果を返すため
    10,000行のテーブルをSeq Scanで取得する

CASE6. バースト転送時の性能劣化(結果①)

- **PgBouncerなし**
  - クライアントとPostgreSQL間で2パケットごとにバースト転送されている事を確認。
    ※ラウンドトリップ毎に多少の差あり
  - 1度のラウンドトリップで確認応答されるウィンドウサイズ（転送されるデータ長）は16,384Bytesだった。

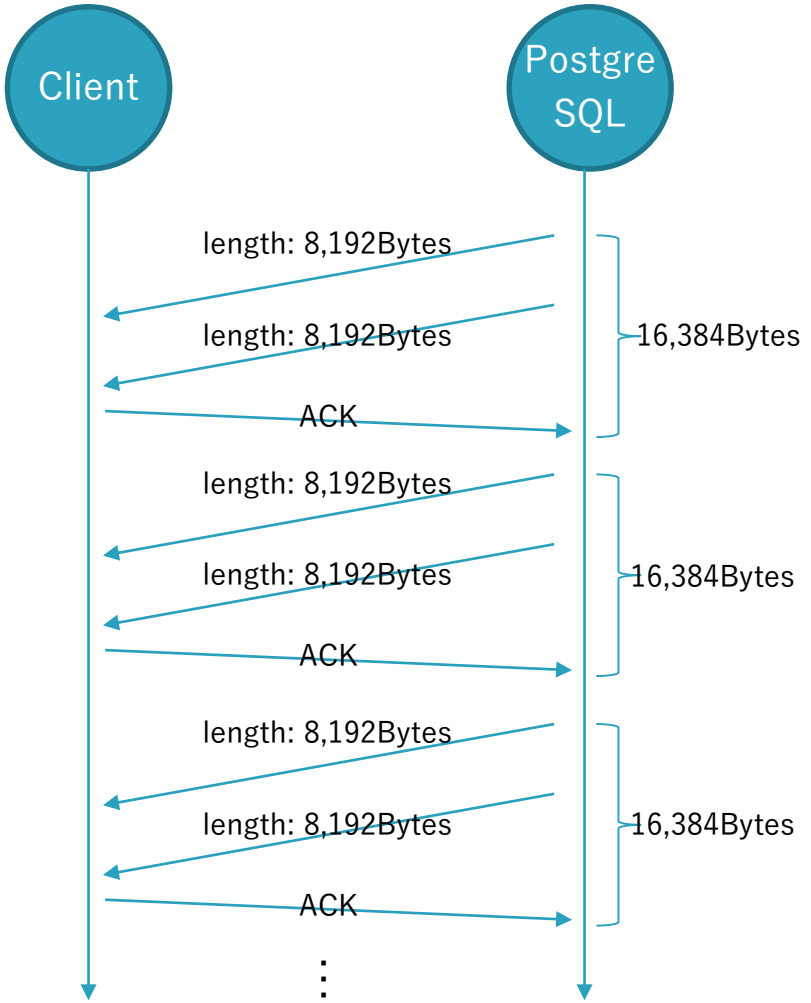


図9: PgBouncerなし時のパケットフロー

CASE6. バースト転送時の性能劣化(結果②)

- **PgBouncerあり**
  - PostgreSQLからPgBouncerに転送されたパケットを、PgBouncerからクライアントへ分割転送している
  - PostgreSQLからPgBouncerへの転送は、PgBouncerからClientへの転送を待つ同期処理となっている
  - PgBouncerとPostgreSQLの間での1度のラウンドトリップで転送されるウィンドウサイズは8,192Byteであった

- 評価: デメリット小～中
  - PgBouncerを使用した状態では、使用しない場合と比べて通信効率が大きく劣化する
  - 論理バックアップなど、大量データの転送時に影響が出る可能性を考慮する必要がある

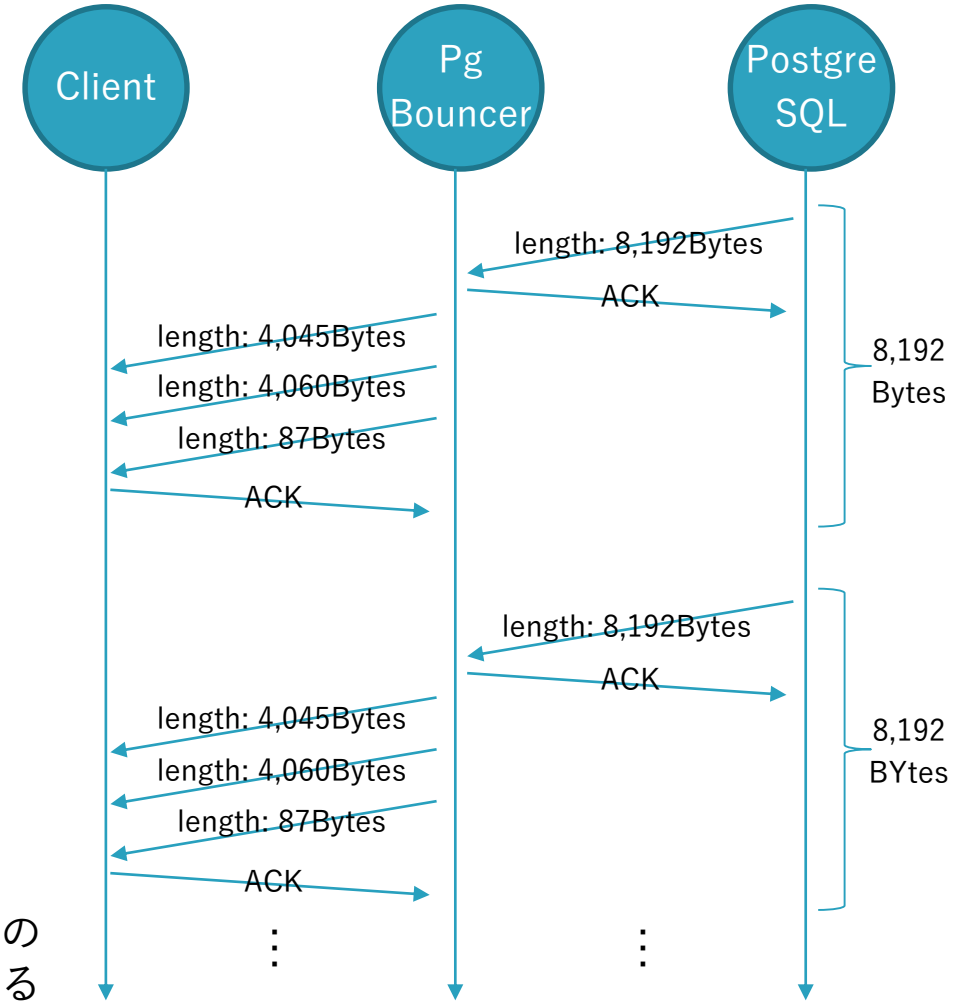


図10: PgBouncerあり時のパケットフロー

まとめ

まとめ

- **推奨環境・構成**
  - クライアントサーバシステムなど、都度接続要求が同時に多数発生する環境でPgBouncerは効果を発揮しやすい
    - 「多数」は、コア数を大幅に超える場面を想定
  - PgBouncerのプロセスがPostgreSQLのCPU使用を圧迫させないよう、異なるノードへの分離やCPU使用の上限値設定が推奨される
- **推奨設定**
  - アプリケーション動作への影響を避けるため、プールモードはセッションプーリングの利用を推奨する
- **利用時の確認事項**
  - ワークロードによってはPgBouncerを導入する事でパフォーマンスが低下してしまう可能性がある
  - PgBouncerを導入する際は、以下の点を事前に確認しシステム全体での性能劣化の状況を把握しておく
    1. 通常利用時、およびピーク時の性能
    2. 大量データ転送時の性能
    3. 短いクエリが大量に発生する場合の性能
       ※ PgBouncerとPostgreSQLを別のマシンで動かす場合に影響が出る可能性あり

PGECons

PostgreSQL Enterprise Consortium