PostgreSQLエンタープライズ・コンソーシアム 技術部会 WG#1

2021年度WG1活動報告書

1. 改訂履歴

版	改訂日	変更内容
1.0	2022/04/06	新規作成

2. ライセンス

3. はじめに

3.1. PostgreSQLエンタープライズコンソーシアムとWG1について

PostgreSQLエンタープライズコンソーシアム(略称 PGECons) は、PostgreSQL本体および各種ツールの情報収集と提供、整備などの活動を通じて、ミッションクリティカル性の高いエンタープライズ領域へのPostgreSQLの普及を推進することを目的として、2012年4月に設立された団体です。

PGECons 技術部会ではPostgreSQLの普及に資する課題を活動テーマとし、3つのワーキンググループで具体的な活動を行っています。

- WG1（新技術検証ワーキンググループ）
- WG2（移行ワーキンググループ）
- WG3（課題検討ワーキンググループ）

WG1では、PostgreSQLの新技術に対する技術検証、および新バージョンにおける性能検証を進めるにあたり、様々な観点から個々の検証テーマをたてて、参加企業が集まってチームを作って具体的な検討を進めています。

3.2. 本資料の概要と目的

本資料は、2021年9月にリリースされた PostgreSQL 14の基本的な処理性能について、前バージョンである PostgreSQL 13 と比較する検証を実施した結果を報告します。

3.2.1. 2021年度の活動テーマ

2021年度は、2021年9月30日にリリースされた PostgreSQL 14を対象として以下のようなテーマで検証を実施しました。

- 定点観測

以下、各検証テーマごとの概要を紹介します。

3.2.2. 定点観測

例年通り、PostgreSQL 新旧バージョンにおける性能を比較する検証を行いました。 参照系については、わずかな性能劣化が見られました。FlameGraphの結果および、14のリリースノートから、共有バッファアクセスの品質改善に伴うオーバヘッドが関与しているものと推察されます。 更新系については、性能差は認められませんでした。

3.3. 成果の公開

本資料も含めて、活動成果は報告書の形にまとめて、 PGEConsのWebサイト で公開しています。また、過去の活動成果も含めてテーマ毎に報告書を検索できるように 成果物総索引 も用意しています。

3.4. 実施体制

2021年7月15日に開催された2021年度第1回技術部会より、以下の企業(名前順)が参加してWG1とWG3が合同で実施しています。

- SRA OSS, Inc. 日本支社
- NECソリューションイノベータ株式会社
- NTTテクノクロス株式会社
- 日鉄ソリューションズ株式会社
- 日本電信電話株式会社
- 株式会社日立製作所
- ヤマトシステム開発株式会社

この中で、NTTテクノクロス株式会社、SRA OSS, Inc. 日本支社は、「主査」としてWG1とWG3の取りまとめ役を担当することになりました。

3.5. 謝辞

検証用のAWS環境をSRA OSS, Inc. 日本支社よりご提供いただきました。 この場を借りて厚く御礼を申し上げます。

4. 定点観測

4.1. 検証概要

WG1では、PostgreSQLの新バージョン・新リリースにあわせて、新旧バージョンの性能比較の検証を目的とした定点観測を2012年度から実施してきました。2014年度からは、それまでの参照処理に加えて更新処理についても検証を実施し、検証結果の公開を行うようになりました。2021年度はアマゾン ウェブ サービス（AWS）上の仮想マシン（vCPU: 32、メモリ: 128GB）で、最新のPostgreSQLバージョン14と前バージョンの13との参照性能の比較および更新性能の比較を行いました。

4.2. pgbenchとは

本検証では、 pgbench というベンチマークツールを使用しました。

pgbenchはPostgreSQLに付属する簡易なベンチマークツールです(バージョン9.5より前はcontribに付属)。標準ベンチマークTPC-B（銀行口座、銀行支店、銀行窓口担当者などの業務をモデル化）を参考にしたシナリオに基づくベンチマークの実行のほか、検索クエリのみを実行するシナリオも搭載されています。また、カスタムスクリプトを用意することで、独自のシナリオでベンチマークを実行することも可能です。

pgbenchでベンチマークを実行すると、以下のように1秒あたりで実行されたトラザクションの数（TPS: Transactions Per Second）が出力されます。TPSはデフォルトではPostgreSQLへの接続に要した時間を含まず、トランザクションごとに再接続を行う-Cオプションを指定した場合にはこれを含みます。なお、バージョン13以前はPostgreSQLへの接続に要した時間を含むTPSと、これを含まないTPSが出力され、「including connections establishing」は前者を、「excluding connections establishing」は後者を示します。

```
pgbench (14.2)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10
number of transactions actually processed: 10/10
latency average = 1.620 ms
initial connection time = 12.429 ms
tps = 617.322057 (without initial connection time)
```

pgbenchには「スケールファクタ」という概念があり、データベースの初期化モードでpgbenchを起動することにより、任意のサイズのテスト用のテーブルを作成できます。デフォルトのスケールファクタは1で、このとき「銀行口座」に対応する「pgbench_accounts」というテーブルで10万件のデータ、約15MBのデータベースが作成されます。

以下に、各スケールファクタに対応するデータベースサイズを示します。

表 4.1 スケールファクタに対するデータベースサイズ

スケールファクタ	データベースサイズ
1	15MB
10	150MB
100	1.5GB
1000	15GB
5000	75GB

初期化モードではpgbench_accountsの他にもテーブルが作成されます。作成されるテーブルのリストを以下に示します。

表 4.2 pgbench_accounts(口座)

列名	データ型	コメント
aid	integer	アカウント番号(主キー)
bid	integer	支店番号
abalance	integer	口座の金額
filler	character(84)	備考

表 4.3 pgbench_branches(支店)

列名	データ型	コメント
bid	integer	支店番号
bbalance	integer	口座の金額
filler	character(84)	備考

表 4.4 pgbench_tellers(窓口担当者)

列名	データ型	コメント
tid	integer	担当者番号
bid	integer	支店番号
tbalance	integer	口座の金額
filler	character(84)	備考

スケールファクタが1の時、pgbench_accountsは10万件、pgbench_branchesは1件、pgbench_tellersは10件のデータが作成されます。スケールファクタを増やすとこれに比例して各テーブルのデータが増えます。

pgbenchには、様々なオプションがあります。詳細は PostgreSQL文書 をご覧ください。ここでは、本検証で使用している主なオプションのみを説明します。

表 4.5 ベンチマークテーブル初期化

オプション	説明
-i	ベンチマークテーブルの初期化
-s	スケールファクタ
-F	作成するテーブルのフィルファクタ

表 4.6 ベンチマークの実行

オプション	説明
-c	同時接続クライアント数
-j	pgbench内のワーカスレッド数
-T	ベンチマークを実行する秒数
-n	実行前にバキュームを行わない
-P	指定した秒数ごとのレポートを表示
-r	実行クエリの平均レイテンシを表示

4.3. 検証構成

4.3.1. マシン構成

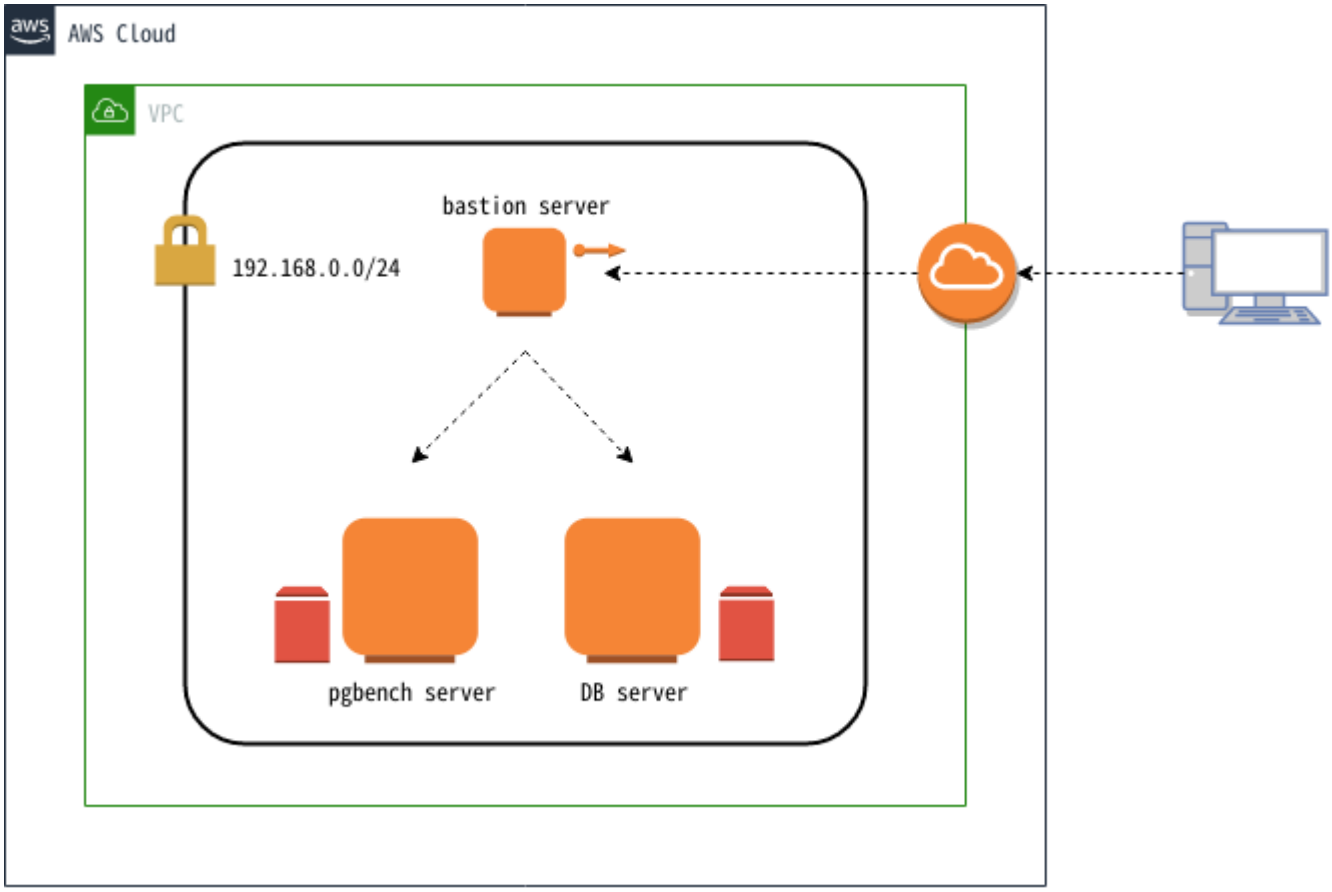

図 4.1 検証マシン構成

本検証では上図構成の通り、AWS上の仮想ネットワーク内に3台の仮想マシンを作成しました。各仮想マシンの用途は、bastion serverがAWS環境へのアクセスに用いる踏み台サーバ、pgbench serverがpgbenchを実行するためのクライアント用サーバ、DB serverがPostgreSQLを稼働させるサーバでした。また、pgbench serverとDB serverには検証作業で作成されるデータを格納するために、ストレージを追加しました。追加ストレージは標準的な用途で使用される汎用SSDタイプでした。DB serverに追加したストレージはIOPSをデフォルトの100 ではなく600を設定しました。各仮想マシンのスペックは以下の通りです。

表 4.7 仮想マシンのスペック

名前	インスタンスタイプ	vCPU	メモリ（GiB）	ルートデバイスサイズ（GiB）/IOPS	追加ストレージサイズ（GiB）/IOPS
bastion server	t2.micro	1	1	10/100	N/A
pgbench server	m5a.8xlarge	32	128	20/100	20/100
DB server	m5a.8xlarge	32	128	20/100	200/600

4.3.2. ソフトウェア構成

検証環境の主要なソフトウェア構成を示します。

表 4.8 bastion server

OS	Red Hat Enterprise Linux 7.9

表 4.9 pgbench server

OS	Red Hat Enterprise Linux 7.9
pgbench	14.0

表 4.10 DB server

OS	Red Hat Enterprise Linux 7.9
PostgreSQL	13.4, 14.0

上記に加え、pgbench serverとDB serverに以下のパッケージをインストールしました。

表 4.11 インストールしたパッケージ

パッケージ名	用途
Development tools（グループパッケージ）, readline-devel, zlib-devel	PostgreSQLのビルド
chrony	時刻同期
perf	検証作業中の性能分析
screen	作業セッションの保持
sysstat	検証作業中のsar取得

4.3.3. カーネル設定

- OS デフォルト

4.3.4. PostgreSQL 設定

PostgreSQL設定ファイル(postgresql.conf)の設定は以下の通りです。これらは参照系検証と更新系検証で共通です。

```
listen_addresses = '*' # クライアント用サーバからの接続用
max_connections = 500  # 多めに設定
shared_buffers = 40GB  # pgbenchのスケールファクタに合わせてデータがすべてメモリに載るように設定
work_mem = 1GB
maintenance_work_mem = 20GB
checkpoint_timeout = 60min  # 試験中にチェックポイントを発生させない
max_wal_size = 160GB        # 試験中にチェックポイントを発生させない
logging_collector = on
log_checkpoints = on
log_lock_waits = on
autovacuum = off # 試験中にI/O処理を発生させない
```

4.3.5. 環境

以下の手順で、データベースクラスタを作成しました。

initdbでデータディレクトリを作成し、上記に示した設定をpostgresql.confに記述します。

```
$ initdb --no-locale --encoding=utf-8
$ vi $PGDATA/postgresql.conf
```

PostgreSQLを起動してベンチマーク用のデータベースを作成します。

```
$ pg_ctl start
$ createdb [dbname]
```

以上の構成で、クライアント接続数を変動させて実行したpgbenchのTPSが、PostgreSQL新旧バージョン間で差があるか否かを検証しました。検証は参照系性能と更新系性能について行いました。

4.4. 検証方法(参照系)

参照系ベンチマークの手順を以下に示します。

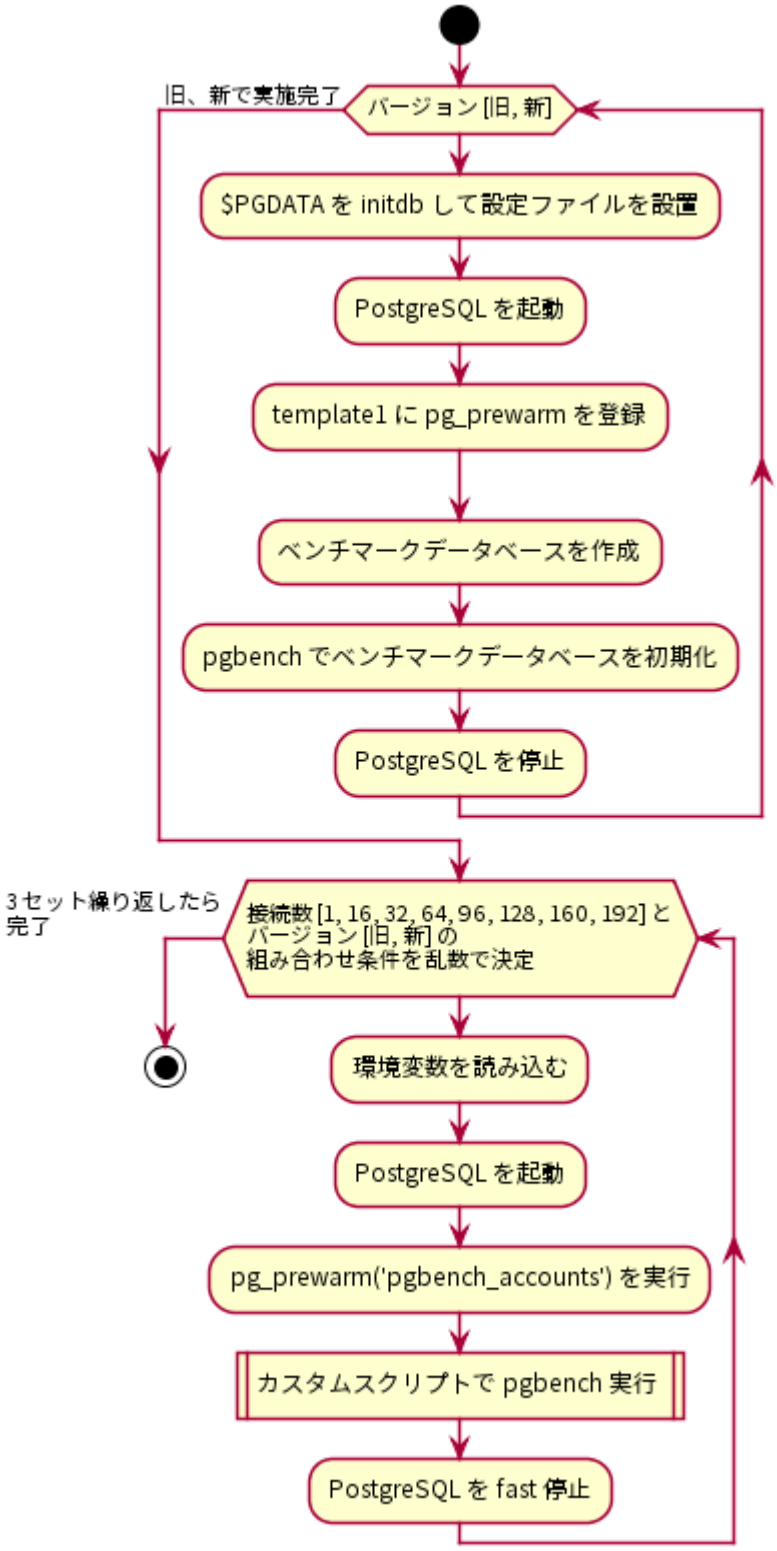

図 4.2 参照系ベンチマークの手順

はじめに、PostgreSQLサーバにデータベースクラスタを作成し、検証構成に従ってpostgresql.confを調整しました。postgresql.confの調整が完了したら、データベースクラスタを起動してtemplate1データベースにpg_prewarmを登録しました。 pg_prewarm はバッファキャッシュにテーブルデータを読み込むためのモジュールで、バッファキャッシュがクリアされているデータベース起動直後の性能低下状態を解消するために用いることができます。pg_prewarmの登録が完了したら、ベンチマークデータベースを作成し、pgbenchコマンドを使ってベンチマーク用データベースを初期化しました。ベンチマーク用データベースはスケールファクタ2000で初期化しました。

```
$ pgbench -i -s 2000 [dbname]
```

以上の作業を、PostgreSQL新旧バージョンで行いました。

その後、クライアント用サーバからベンチマークを実施しました。参照系ベンチマークではクライアント接続数を1、16、32、64、96、128、160、192の8条件とし、PostgreSQL新旧バージョンのそれぞれでTPSを取得しました。各ベンチマーク試行で決定されるクライアント接続数とPostgreSQL新旧バージョンはランダマイズしました。これは各要因における順序効果を打ち消すことを意図していました。ベンチマーク試行条件に当てはまる環境変数を読み込みんで、PostgreSQLを起動したら、測定スクリプト実行前にpg_prewarmを実行しました。これによりテーブルデータはすべてバッファキャッシュに格納されます。

```
=# SELECT pg_prewarm('pgbench_accounts');
```

pg_prewarmを実行したら、以下の参照系カスタムスクリプトをpgbenchで実行し、適度な負荷がかかるようにしました。これは、pgbenchの標準シナリオ（pgbench -S）ではCPUに充分な負荷がかからないためです。具体的には、ランダムに10000行を取得しています。

```
\set naccounts 100000 * :scale
\set row_count 10000
\set aid_max :naccounts - :row_count
\set aid random(1, :aid_max)

SELECT count(abalance) FROM pgbench_accounts WHERE aid BETWEEN :aid and :aid + :row_count;
```

クライアント用サーバで実行したpgbenchコマンドは以下の通りです。

```
$ pgbench -n -h [host] -p [port] -c [clients] -j [threads] -f [参照系カスタムスクリプト] -T 300 -s 2000 -P 1 -r [dbname]
```

SELECTのみであるためVACUUMを実行せず(-n)、pgbench クライアント数(-c)とスレッド数(-j)を変動させながら、300秒ずつ(-T)実行しています。スレッド数はクライアント数の半分としています。スケールファクタ(-s)にはデータベース初期化時と同じ2000を指定します。また、1秒毎の進捗レポート取得(-P)と各クエリの平均レイテンシ(-r)も取得する指定にしました。ただし、-P, -r の指定によって得られた結果は今回の考察に直接用いることはありませんでした。

クライアント用サーバからのベンチマークはここまでを1セットとし、3セット実施して得られたTPSの中央値を結果としました。

4.5. 検証結果(参照系)

参照系ベンチマークのTPSのグラフを以下に示します。TPSは接続確立にかかった時間を無視した値(excluding connections establishing)を用いています。(以降同様)

13, 14 はともにクライアント接続数の増加につれてTPSが増加し、コア数を超えた辺りのクライアント数でTPSは頭打ちとなっています。また、新旧バージョン間でTPSに差がありました。13の平均TPSは6916.72、14の平均TPSは6754.68と、14の平均TPSは13よりも2.34%下回っていました。

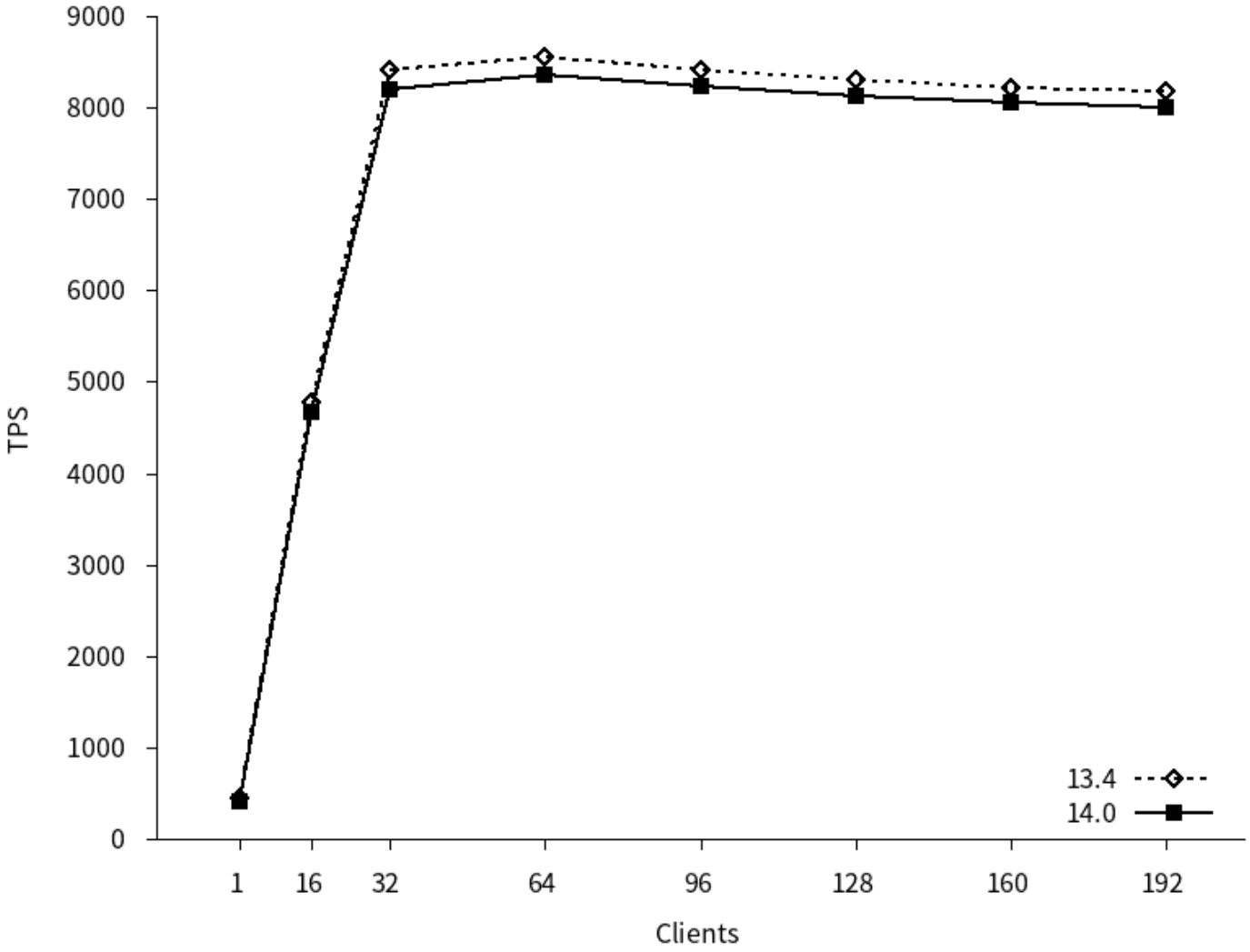

図 4.3 各クライアント数に対するTPS (参照系)

参照系ベンチマークのレイテンシのグラフを以下に示します。

13, 14 はともにクライアント接続数の増加につれてレイテンシがほぼ直線的に増加しています。また、新旧バージョン間でレイテンシにわずかな差がありました。14の平均レイテンシは11.08 ms、13の平均レイテンシは10.82 msと、14の平均レイテンシは13よりも2.3%上回っていました。

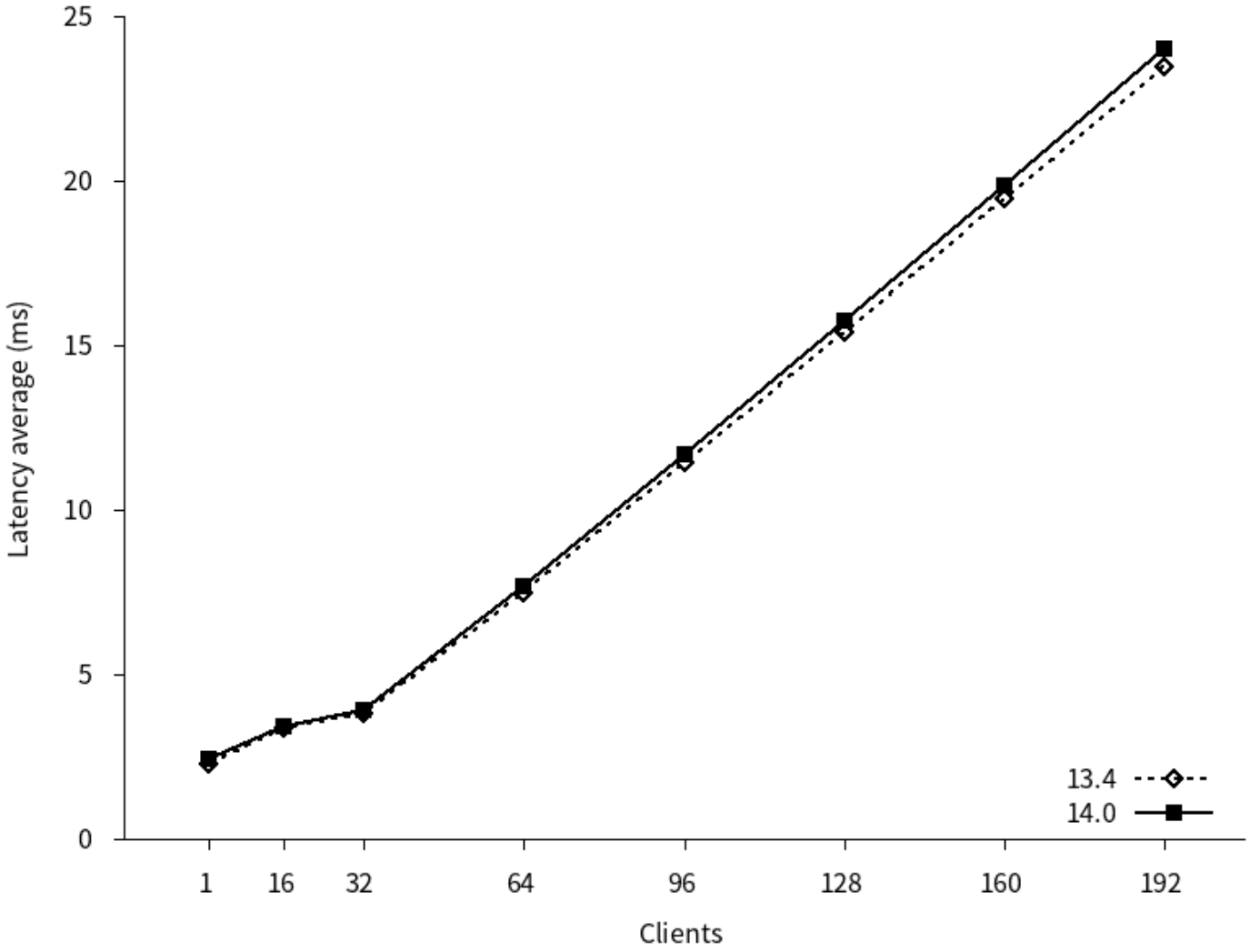

図 4.4 各クライアント数に対するレイテンシ (参照系)

続いて、バージョン間の性能差異をもたらした原因を追求するため追加検証を行いました。

4.6. 追加検証(参照系)

バージョン13と14の参照系検証結果から、14にわずかな性能劣化が確認されました。そこで、両バージョンのTPS差が最も大きかった、クライアント数1の条件でperfを取得し、FlameGraphを生成しました

表 4.12 各クライアント数に対するTPSの前バージョン比 (参照系)

クライアント数	TPS		前バージョン比
	13.4	14.0	
1	437.241	412.937	**94%**
16	4778.467	4668.131	98%
32	8404.637	8193.761	97%
64	8546.134	8361.549	98%
96	8405.358	8229.714	98%
128	8310.025	8120.542	98%
160	8218.915	8055.056	98%
192	8188.156	8000.038	98%

4.7. 追加検証結果(参照系)

FlameGraphの結果は以下の通りです。

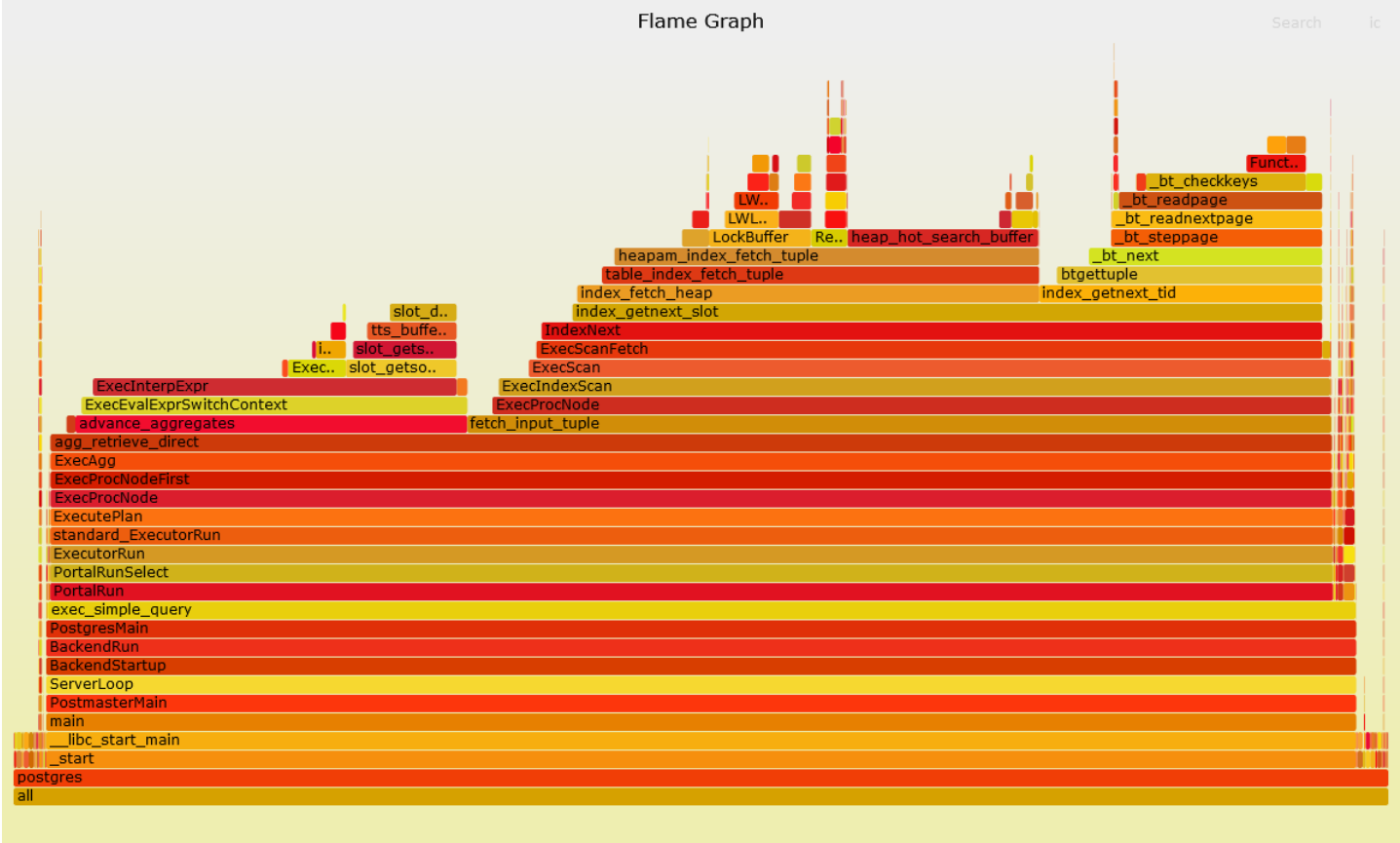

図 4.5 バージョン13.4のFlameGraph

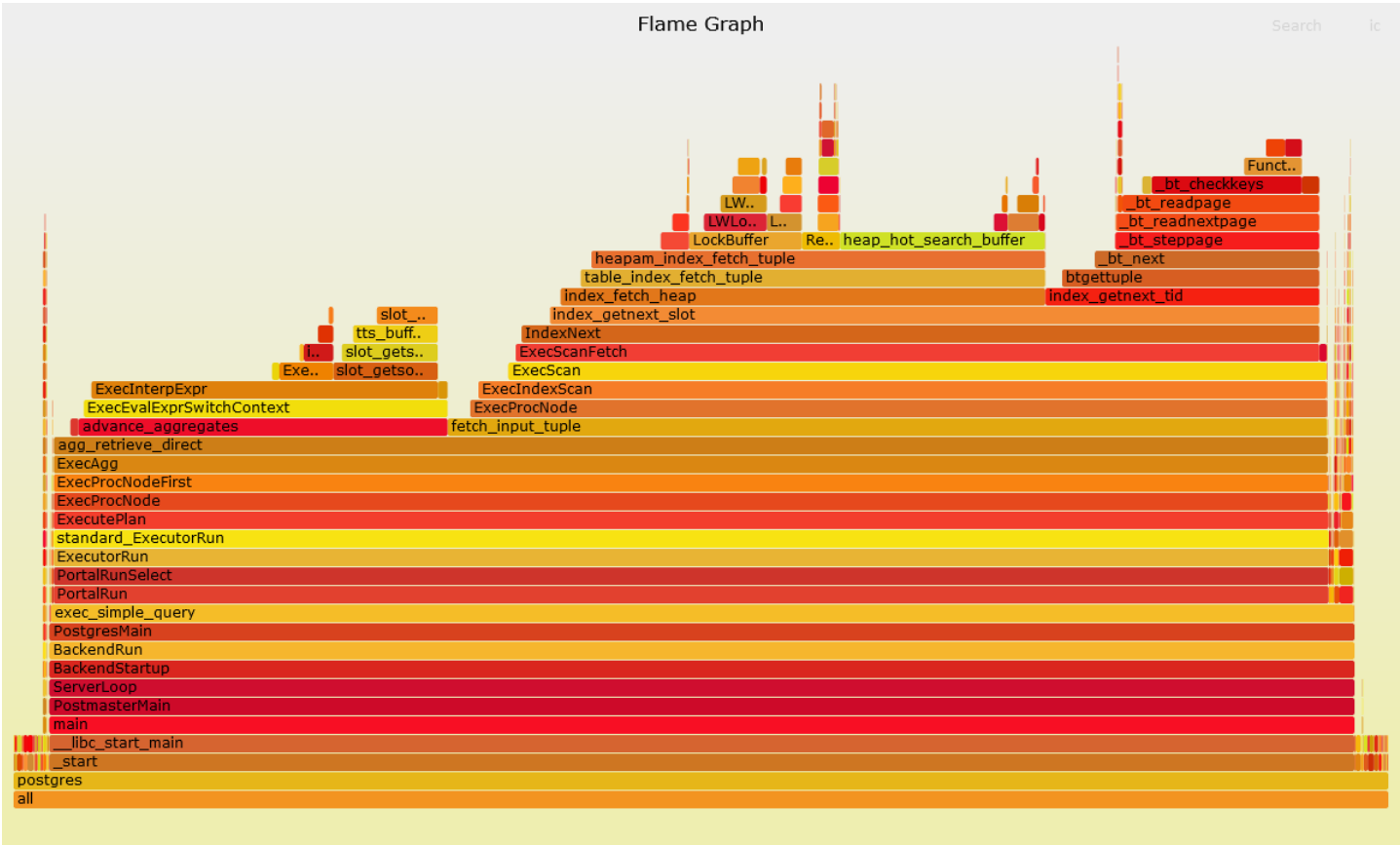

図 4.6 バージョン14.0のFlameGraph

両者のコールグラフに着目すると、呼び出される関数に大きな違いはありませんが、バージョン14ではindex_fetch_heap関数の割合が増えています。PostgreSQL 14の リリースノート によると、共有バッファアクセスの品質改善がなされています。

> Various improvements in valgrind error detection ability (Álvaro Herrera, Peter Geoghegan)

参照系ベンチマークは充分な負荷をかけるため、10000行を取得していて、共有バッファへのアクセスが多く、かつ、それが占める割合も高いため、この変更が新旧バージョンのTPSおよびレイテンシに影響を与えたものと推察されます。

続いて、更新性能について検証します。

4.8. 検証方法(更新系)

更新系ベンチマークの手順を以下に示します。

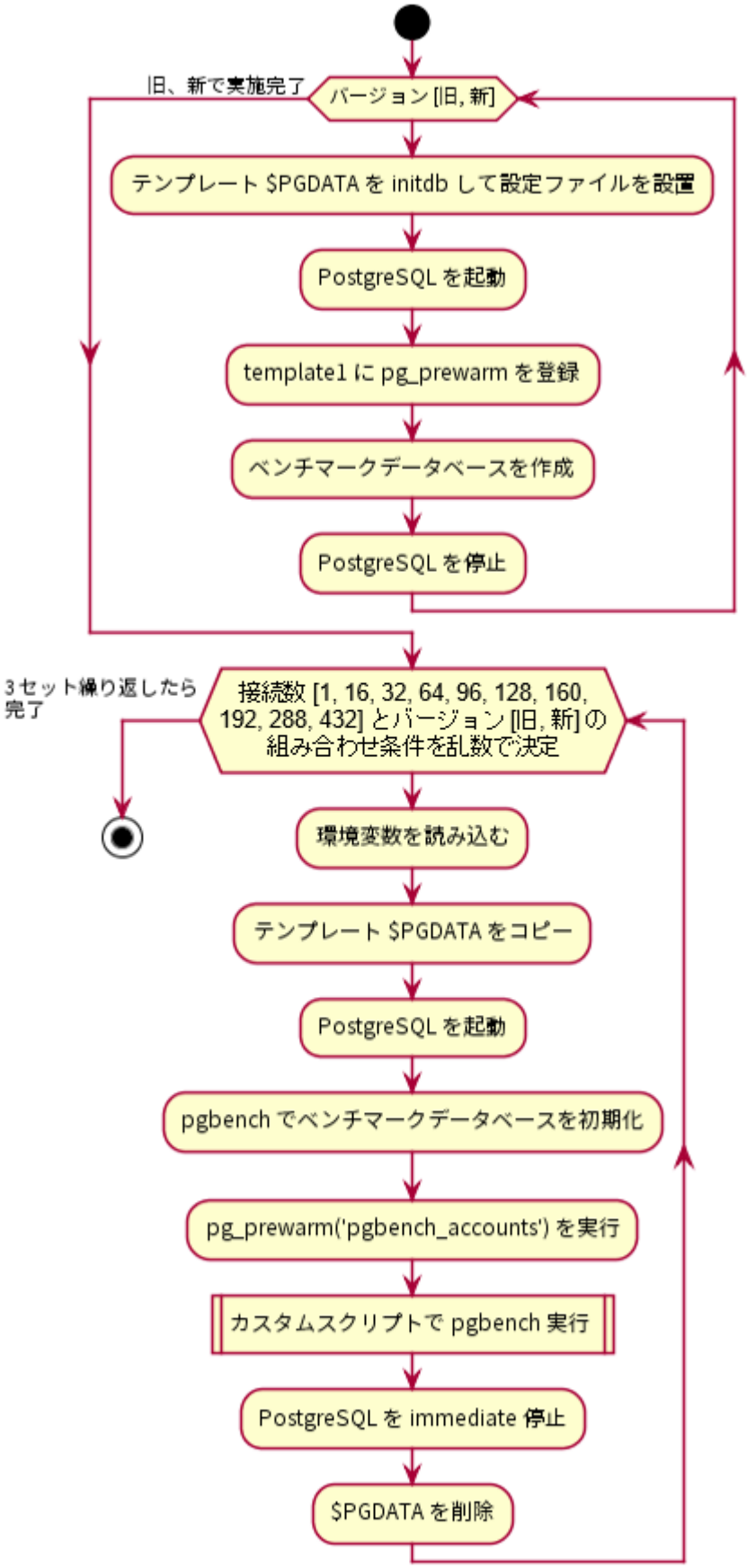

図 4.7 更新系ベンチマークの手順

はじめに、PostgreSQLサーバにデータベースクラスタを作成し、検証構成に従ってpostgresql.confを調整しました。postgresql.confの調整が完了したら、データベースクラスタを起動してtemplate1データベースにpg_prewarmを登録し、ベンチマークデータベースを作成しました。

以上の作業を、PostgreSQL新旧バージョンで行いました。

その後、クライアント用サーバからベンチマークを実施しました。更新系ベンチマークではクライアント接続数を1、16、32、64、96、128、160、192、288、432の10条件とし、PostgreSQL新旧バージョンのそれぞれでTPSを取得しました。各ベンチマーク試行で決定されるクライアント接続数とPostgreSQL新旧バージョンはランダマイズしました。これは各要因における順序効果を打ち消すことを意図していました。ベンチマーク試行条件に当てはまる環境変数を読み込みんで、予め作成しておいたテンプレートデータベースクラスタをコピーし、PostgreSQLを起動し、pgbenchコマンドを用いてベンチマーク用データベースをスケールファクタ2000で初期化しました。このとき、フィルファクタは80としました。

```
$ pgbench -i -s 2000 [dbname] -F 80
```

その後、pg_prewarmを実行し、以下の更新系カスタムスクリプトをpgbenchで実行し、適度な負荷がかかるようにしました。

```
\set naccounts 100000 * :scale
\set aid_val random(1, :naccounts)
UPDATE pgbench_accounts SET filler=repeat(md5(current_timestamp::text),2) WHERE aid = :aid_val;
```

これを、クライアント用サーバから

```
$ pgbench -n -h [host] -p [port] -c [clients] -j [threads] -f [更新系カスタムスクリプト] -T 300 -s 2000 -P 1 -r [dbname]
```

として実行しました。VACUUMを実行せず(-n)、pgbench クライアント数(-c)とスレッド数(-j)を変動させながら、300秒ずつ(-T)実行しています。スレッド数はクライアント数の半分としています。スケールファクタ(-s)にはデータベース初期化時と同じ2000を指定します。また、1秒毎の進捗レポート取得(-P)と各クエリの平均レイテンシ(-r)も取得する指定にしました。（ただし、-P, -r の指定によって得られた結果は今回の考察に直接用いることはありませんでした）

クライアント用サーバからのベンチマークはここまでを1セットとし、3セット実施して得られたTPSの中央値を結果としました。なお、更新系検証の場合は実行後のテーブルを使い回さず、毎回初期化しています。

4.9. 検証結果(更新系)

更新系ベンチマークのTPSのグラフを以下に示します。

13, 14 はともにクライアント接続数の増加につれてTPSが増加し、192接続までTPSの増加傾向は均一でした。288接続以降もTPSの増加はやや認められましたが、192接続までの増加傾向よりも緩やかでした。新旧バージョン間でTPSに差はありませんでした。

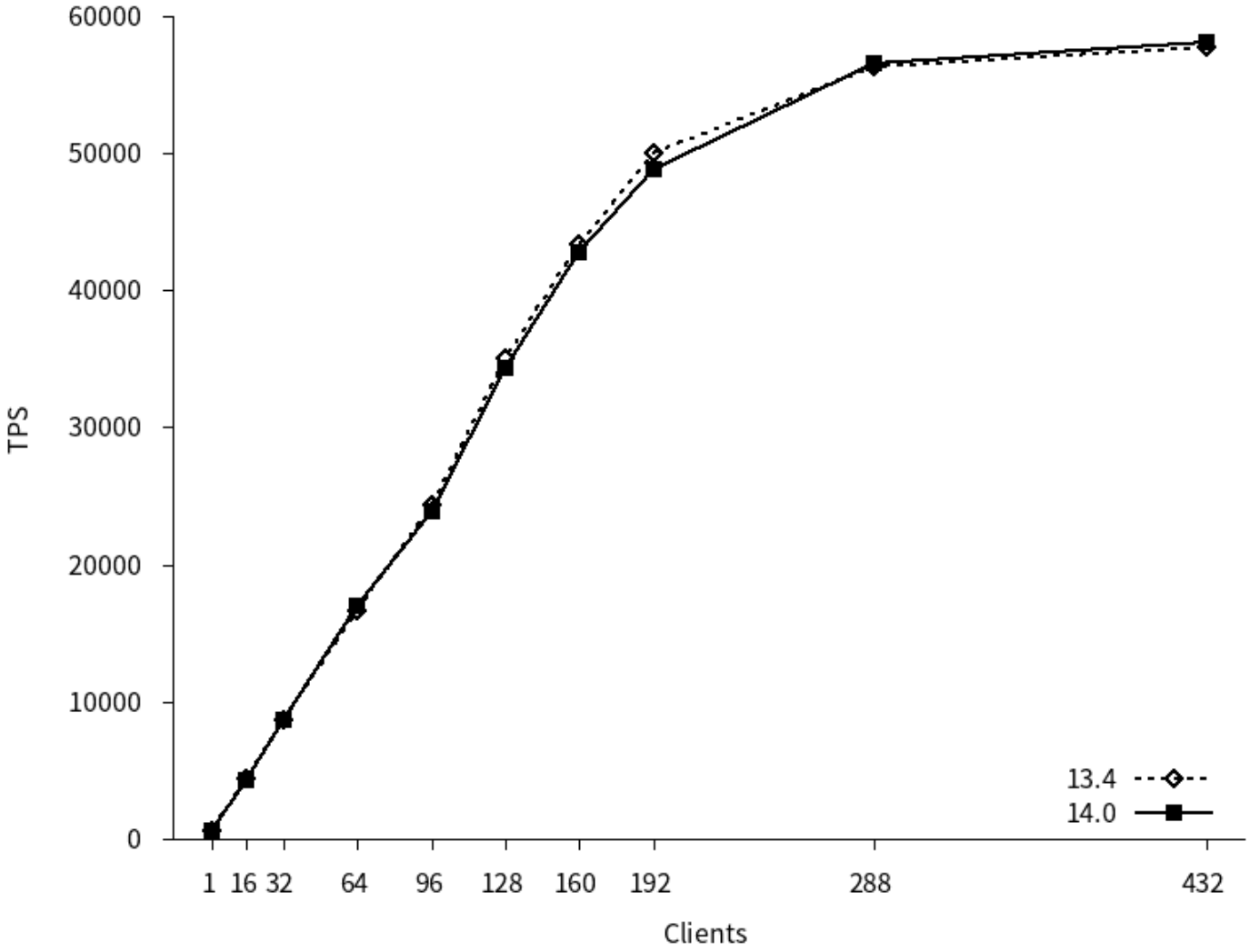

図 4.8 各クライアント数に対するTPS (更新系)

更新系ベンチマークのレイテンシのグラフを以下に示します。

13, 14 はともにクライアント接続数 16 の時点から192までレイテンシがほぼ横ばい、それ以降は増加しました。また、新旧バージョン間でレイテンシに差はほとんどありませんでした。

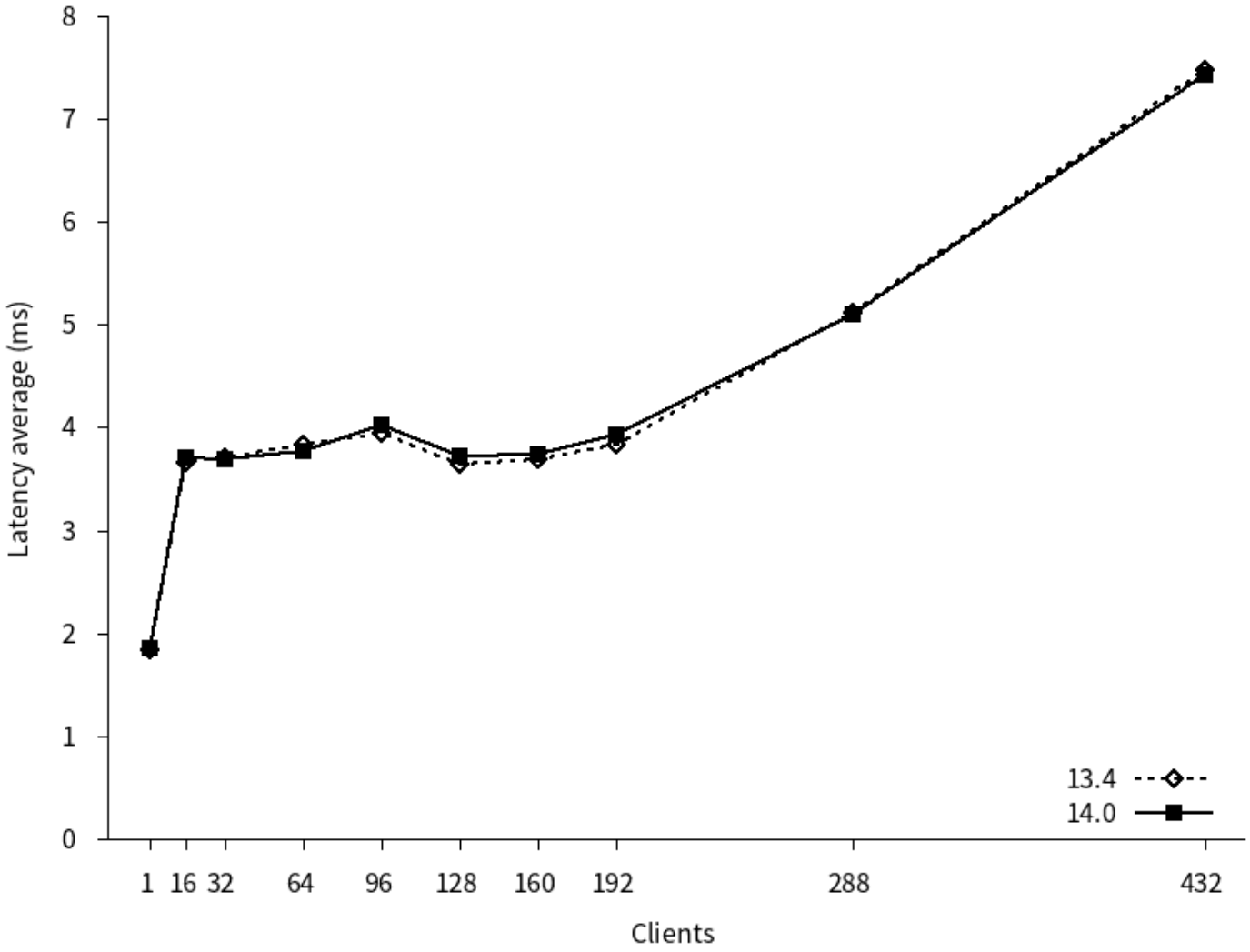

図 4.9 各クライアント数に対するレイテンシ (更新系)

4.10. まとめ

本検証では例年通り、PostgreSQL最新バージョンと旧バージョンとの性能比較検証を行いました。

参照系については、わずかな性能劣化が見られました。FlameGraphの結果および、14のリリースノートから、共有バッファアクセスの品質改善に伴うオーバヘッドが関与しているものと推察されます。

更新系については、性能差は認められませんでした。

5. 著者

（企業・団体名順）

版	所属企業・団体名	部署名	氏名
第1.0版 (2021年度WG1&WG3)	SRA OSS, Inc. 日本支社	OSS事業本部 技術部	佐藤 友章
		OSS事業本部 技術部 データベース技術グループ	北山 貴広
			正野 裕大
	NECソリューションイノベータ株式会社	第一PFソフトウェア事業部	近藤 太樹
			湯村 昇平
	NTTテクノクロス株式会社	デジタルツイン事業部 第三ビジネスユニット	上原 一樹
	日鉄ソリューションズ株式会社	流通・サービスソリューション事業本部 アドバンストテクノロジー部	伊藤 春
			永井 光
			秋山 暉佳
	日本電信電話株式会社	NTT OSSセンタ	坂田 哲夫
	株式会社日立製作所	サービスプラットフォーム事業本部 OSSソリューションセンタ	稲垣 毅
	ヤマトシステム開発株式会社	システム本部 技術開発部	鳥居 英明
		システム本部 技術開発部 開発技術推進	毛呂 良寛
		IT基盤本部 運用技術部 大阪インフラ	藤井 大和