PGECons
PostgreSQL Enterprise Consortium

2019年度WG3活動成果報告
クラウド検証編
②実機検証編（AWS）

PostgreSQLエンタープライズコンソーシアム
WG3（課題検討WG）

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。

性能検証結果 -pgbench編-

検証概要

- 目的
  - pgbenchで、Amazon Aurora/Amazon RDS/Amazon EC2の性能を比較検証
  - 検証結果から、企業でクラウド活用を検討するための資料として提供する。さらに、考えられる知見をまとめる。

- 検証内容
  - pgbench(カスタムクエリ)で参照性能、更新性能を測定
  - カスタムクエリは、PGEConsの定点観測で利用しているものを使用

- 検証時期
  - 2019年7月、2020年2月の2回実施

検証環境

- DB種類
  - Amazon Aurora with PostgreSQL compatibility（以降、Auroraと表記）
  - Amazon RDS for PostgreSQL（以降、RDSと表記）
  - PostgreSQL（コミュニティ版） on Amazon EC2（以降、EC2と表記）

- インスタンス
  - PostgreSQLサーバ
    - r5.2xlarge（vCPU:8、メモリ:64GiB）
    - r5.12xlarge（vCPU:48、メモリ:384GiB）
  - PostgreSQLクライアント（pgbench実行環境）
    - r5.2xlarge（vCPU:8、メモリ:64GiB）
  - サーバとクライアントは同一AZに配置

検証構成

- Aurora設定
  - エンジンのオプション：
    Aurora PostgreSQL（compatible with PostgreSQL 10.7）
  - データベースの機能：1つのライターと複数のリーダー
  - テンプレート：本番稼働用
  - DBインスタンスサイズ：db.r5.2xlarge（標準）、db.r5.12xlarge（高性能）
  - マルチAZ：Auroraレプリカを作成しない
  - DBクラスターのパラメータグループ：カスタム
  - DBパラメータグループ：カスタム
  - 自動バックアップの有効化：あり（7日間）＊強制＊
  - 暗号を有効化：あり
  - Performance Insightsを有効化：あり（7日間）
  - 拡張モニタリングの有効化：あり（60s）
  - マイナーバージョン自動アップグレードの有効化：なし

検証構成

- RDS設定
  - エンジンのオプション：PostgreSQL 10.7-R1
  - テンプレート：本番稼働用
  - DBインスタンスサイズ：db.r5.2xlarge（標準）、db.r5.12xlarge（高性能）
  - ストレージ：汎用SSD 100GiB（シングルAZ）
  - DBパラメータグループ：カスタム
  - 自動バックアップの有効化：あり（7日間）
  - 暗号を有効化：あり
  - Performance Insightsを有効化：あり（7日間）
  - 拡張モニタリングの有効化：あり（60s）
  - マイナーバージョン自動アップグレードの有効化：なし

検証構成

- PostgreSQL on EC2設定
  - PostgreSQL：10.7

    ```
    $ initdb -D /home/postgres/data -E UTF-8 --no-locale -X /h1/pg_wal -A md5 -W
    ```
  - インスタンスサイズ：db.r5.2xlarge（標準）、db.r5.12xlarge（高性能）
  - OS：RHEL7.6
  - モニタリング（CloudWatch）：有効
  - テナンシー：共有
  - ストレージ

デバイス	用途	容量	タイプ	暗号化
/dev/sda1	OS他	100GiB	汎用SSD	無し
/dev/sdb1	WAL領域、アーカイブ領域	100GiB	汎用SSD	無し

検証構成

- postgresql.conf設定（r5.2xlarge）

パラメータ	Aurora	RDS	EC2
listen_addresses	*	*	*
max_connections	5000	5000	5000
shared_buffers	43798032kB	16237064kB	16GB
work_mem	1GB	1GB	1GB
maintenance_work_mem	1039MB	1039MB	8GB
checkpoint_timeout	60	3600	3600
max_wal_size	1024	163840	163840
archive_mode	on	on	on
effective_cache_size	43798032kB	32474128kB	32GB
autovacuum	off	off	off

黒：手動設定値
緑：自動設定値
赤：変更不可

検証構成

- postgresql.conf設定（r5.12xlarge）

パラメータ	Aurora	RDS	EC2
listen_addresses	*	*	*
max_connections	5000	5000	5000
shared_buffers	266227128kB	97950976kB	96GB
work_mem	1GB	1GB	1GB
maintenance_work_mem	6272MB	6272MB	48GB
checkpoint_timeout	60	3600	3600
max_wal_size	1024	163840	163840
archive_mode	on	on	on
effective_cache_size	266227128kB	195901960kB	192GB
autovacuum	off	off	off

黒：手動設定値
緑：自動設定値
赤：変更不可

検証方法(１)

- 参照系検証
  - PGEConsで実施している検証方法を踏襲
  - 初期化
    - pgbench -i -s 2000 [dbname]
  - 測定方針
    - 同時クライアント数
      - 1, 4, 16, 32, 48, 64, 80, 96 で実施(スレッド数は半分)
    - 測定前に、キャッシュへの読み込みを実施
      - SELECT pg_prewarm ('pgbench_accounts');
    - 測定はカスタムクエリ

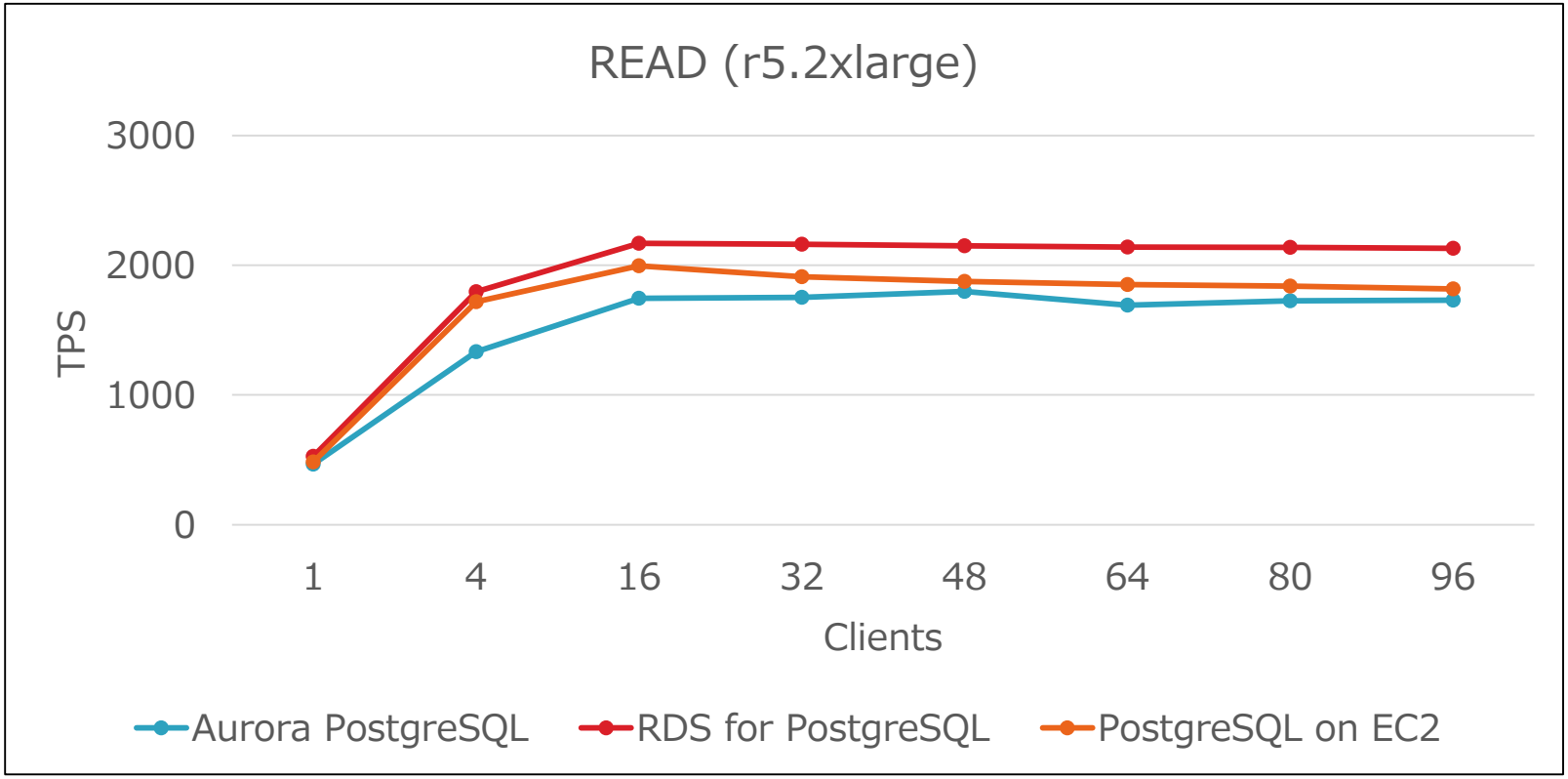
```
¥set naccounts 100000 * :scale
¥set row_count 10000
¥set aid_max :naccounts - :row_count
¥set aid random (1, :aid_max)

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

    - 測定の実行(複数回実行)

```
$ pgbench -r -P 1 -n -c [clients] -j [threads] -f [カスタムクエリ] -T 300 -s 2000
-h [host] -p [port] [dbname]
```

検証結果(１)

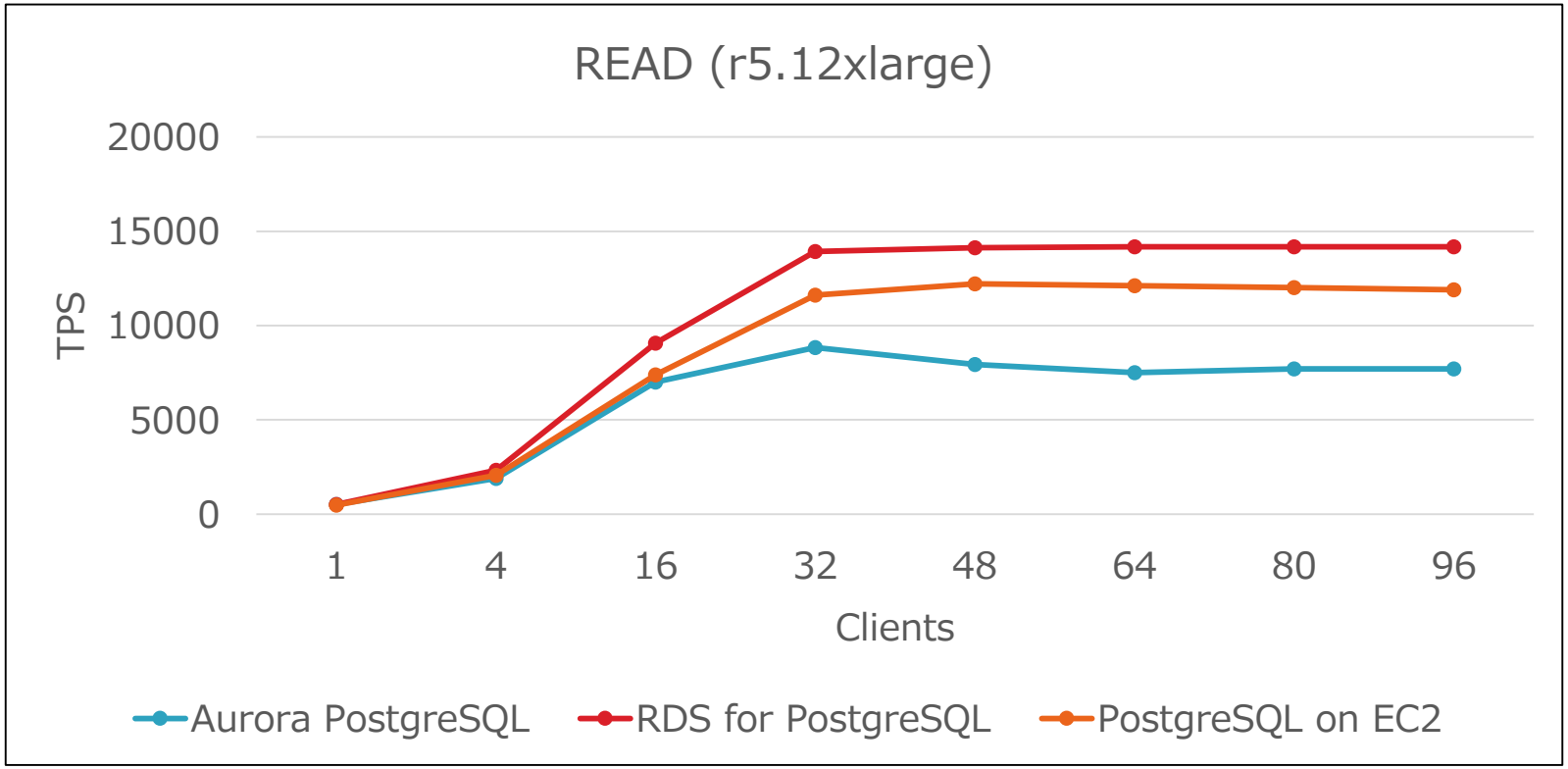
- 参照系検証(標準)



→標準インスタンスでの参照性能結果は、タイプによる性能差はそれほどなく、コア数辺りまで単調増加することが確認できた。

検証結果(１)

- 参照系検証(高性能)



→高性能インスタンスの参照性能結果は、RDSとEC2はコア数辺りまで単調に増加することが確認できた。一方で、Auroraの性能が伸びていないことが確認できており、単純な参照のようなクエリはAuroraのワークロードには適していなかったと考えられる。

検証方法(２)

- 更新系検証
  - PGEConsで実施している検証方法を踏襲
  - 初期化
    - pgbench -i -s 2000 [dbname] -F 80
  - 測定方針
    - 同時クライアント数
      - 1, 4, 16, 32, 48, 64, 80, 96 で実施(スレッド数は半分)
    - 測定前に、キャッシュへの読み込みを実施
      - SELECT pg_prewarm ('pgbench_accounts');
    - 測定はカスタムクエリ

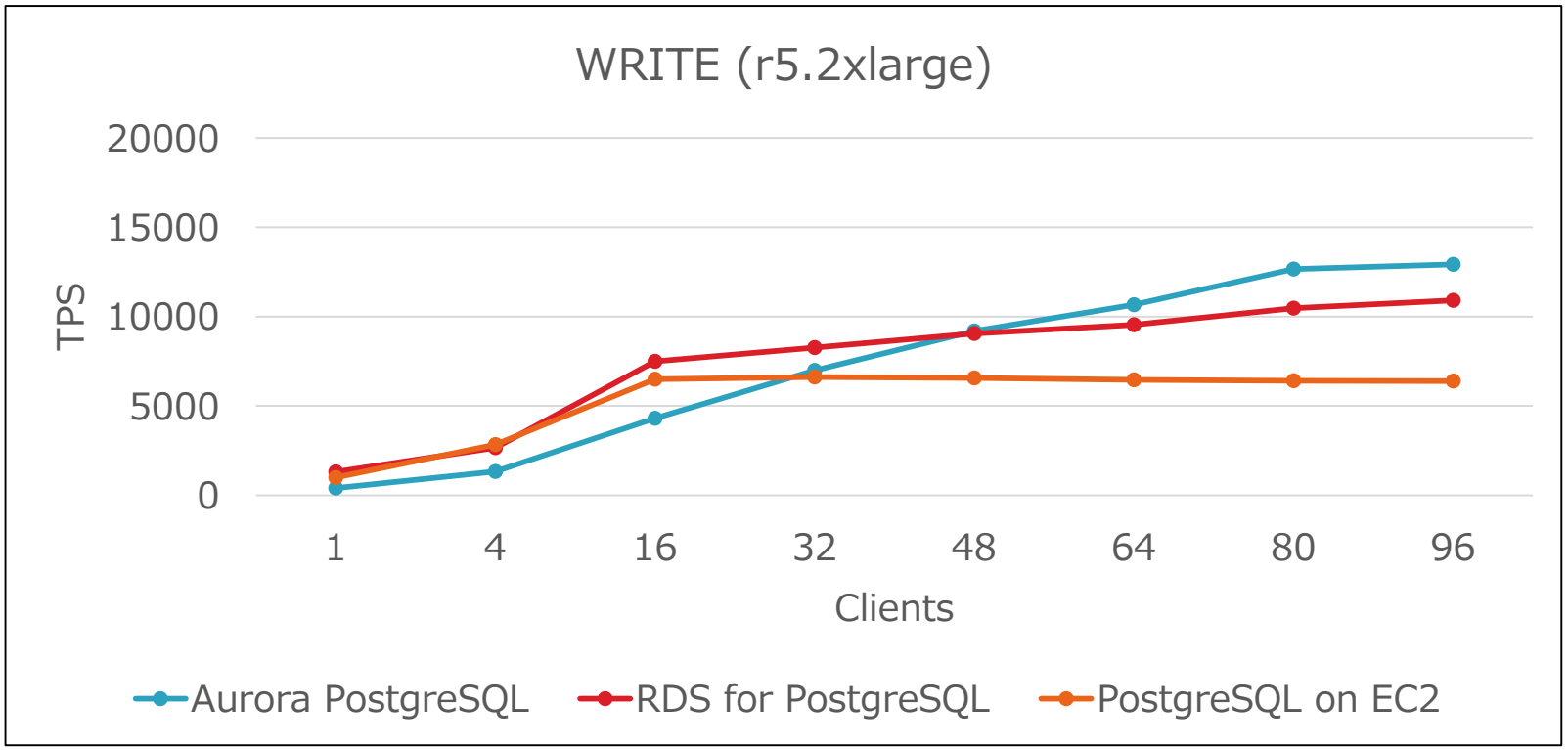
```
¥set naccounts 100000 * :scale
¥set aid_val random (1, :naccounts)
UPDATE pgbench_accounts SET filler=repeat (md5 (current_timestamp::text) ,2) WHERE
aid= :aid_val;
```

    - 測定の実行(複数回実行)

```
$ pgbench -r -P 1 -n -c [clients] -j [threads] -f [カスタムクエリ] -T 300 -s 2000
-h [host] -p [port] [dbname]
```

検証結果(２)

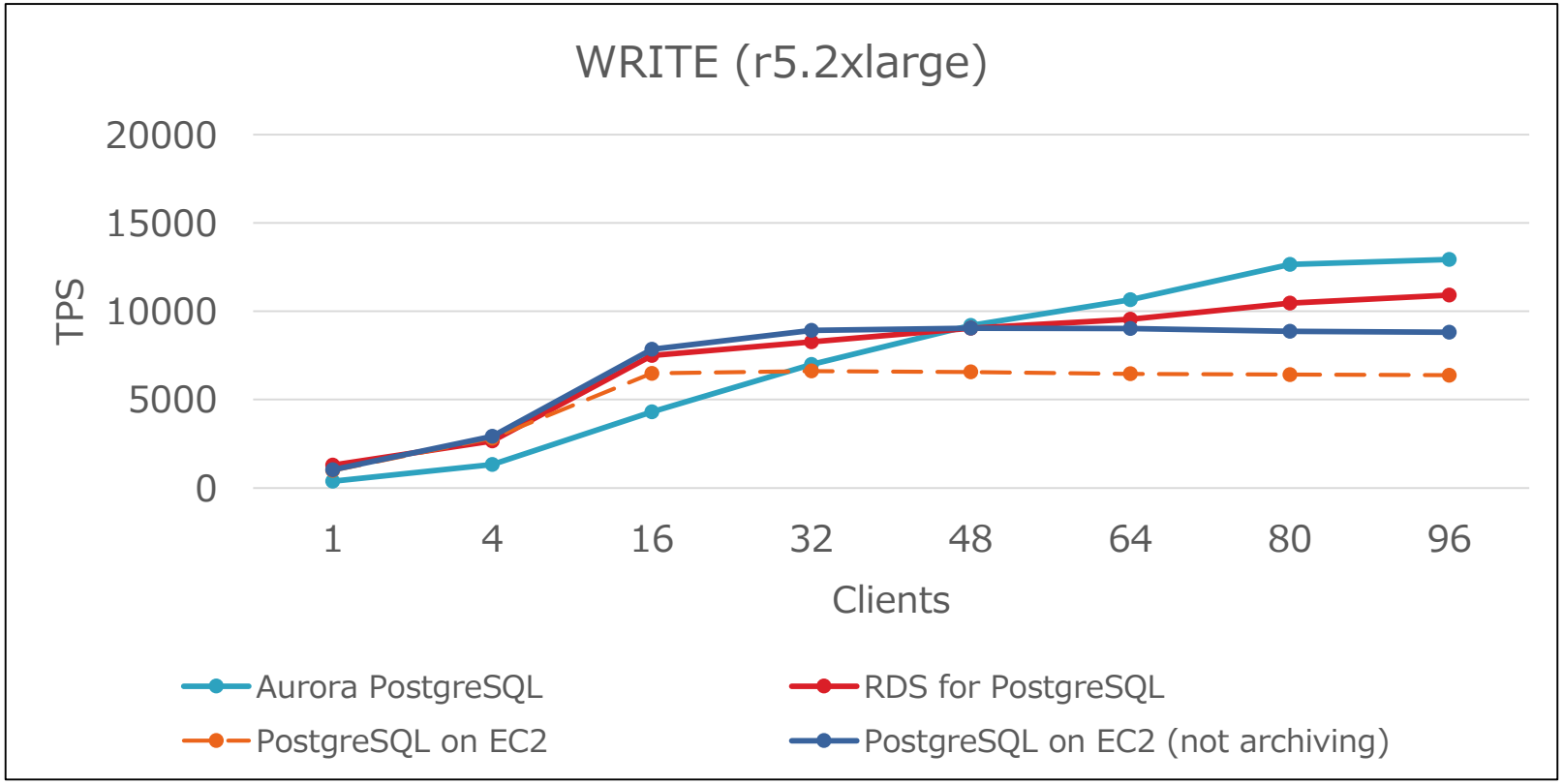
- 更新系検証(標準)



→標準インスタンスでの更新性能結果は、RDSとEC2はコア数辺りまで、Auroraは接続数が多くなるほどリニアに増加することが確認できた。
ただ、RDSとEC2で性能差が確認された。クライアント数が増加するほど差が大きいため、ストレージ書込みによるIOPS制限による影響と考えられたので、アーカイビングせず(archive_mode=off)に検証を行った。

検証結果(２)

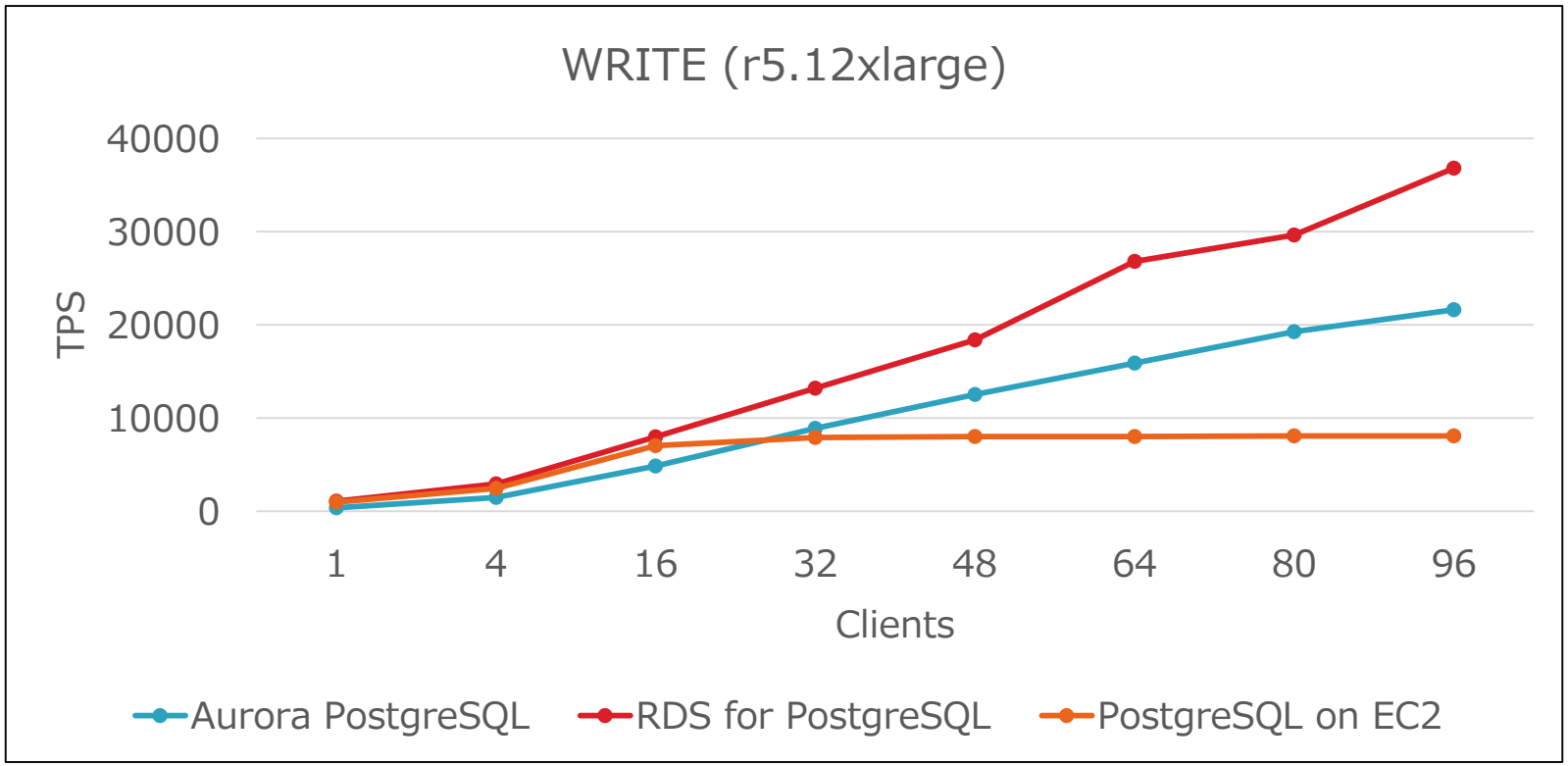
- 更新系検証(標準)



→想定のとおり、RDSとEC2の性能差がほぼなくなった。
配置により、ストレージ書込みの影響が大きいことがわかるので、クラウドの場合でも、設計は気を付けて行うべきである。

検証結果(２)

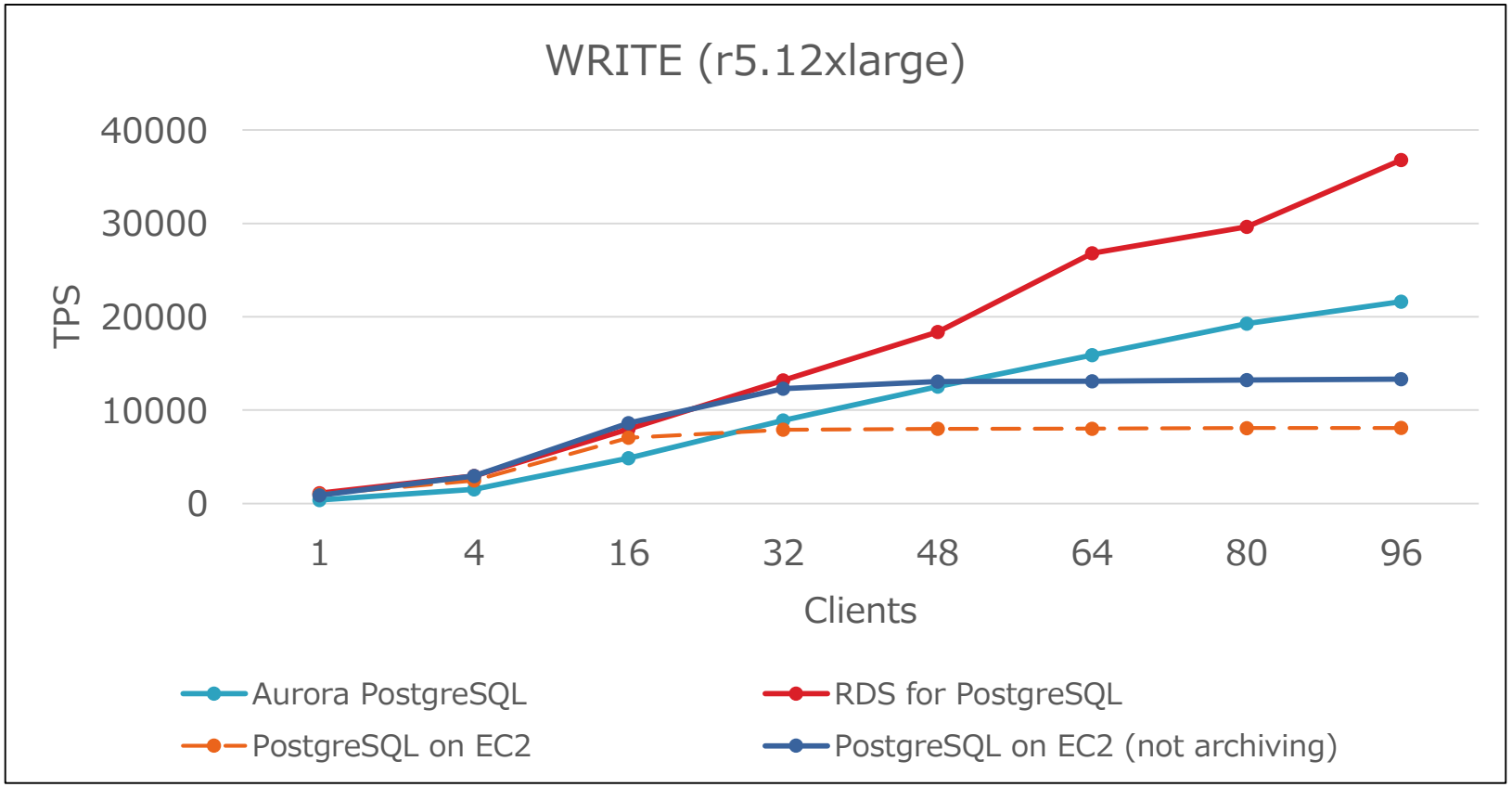
- 更新系検証(高性能)



→高性能インスタンスでの更新性能結果は、RDSとAuroraはリニアに増加することが確認できた。一方で、EC2の性能が伸びていない。標準インスタンスと同様に、書込みのIOPS制限による影響と考えられたので、アーカイビングせずに検証を実施した。

検証結果(２)

- 更新系検証(高性能)



→高性能インスタンスでも、標準インスタンスと同様に性能が改善された。それでも、マネージドと比較するとギャップがある。そのため、IOPSを増やす施策を実施して、再検証を実施した。

検証結果(２)

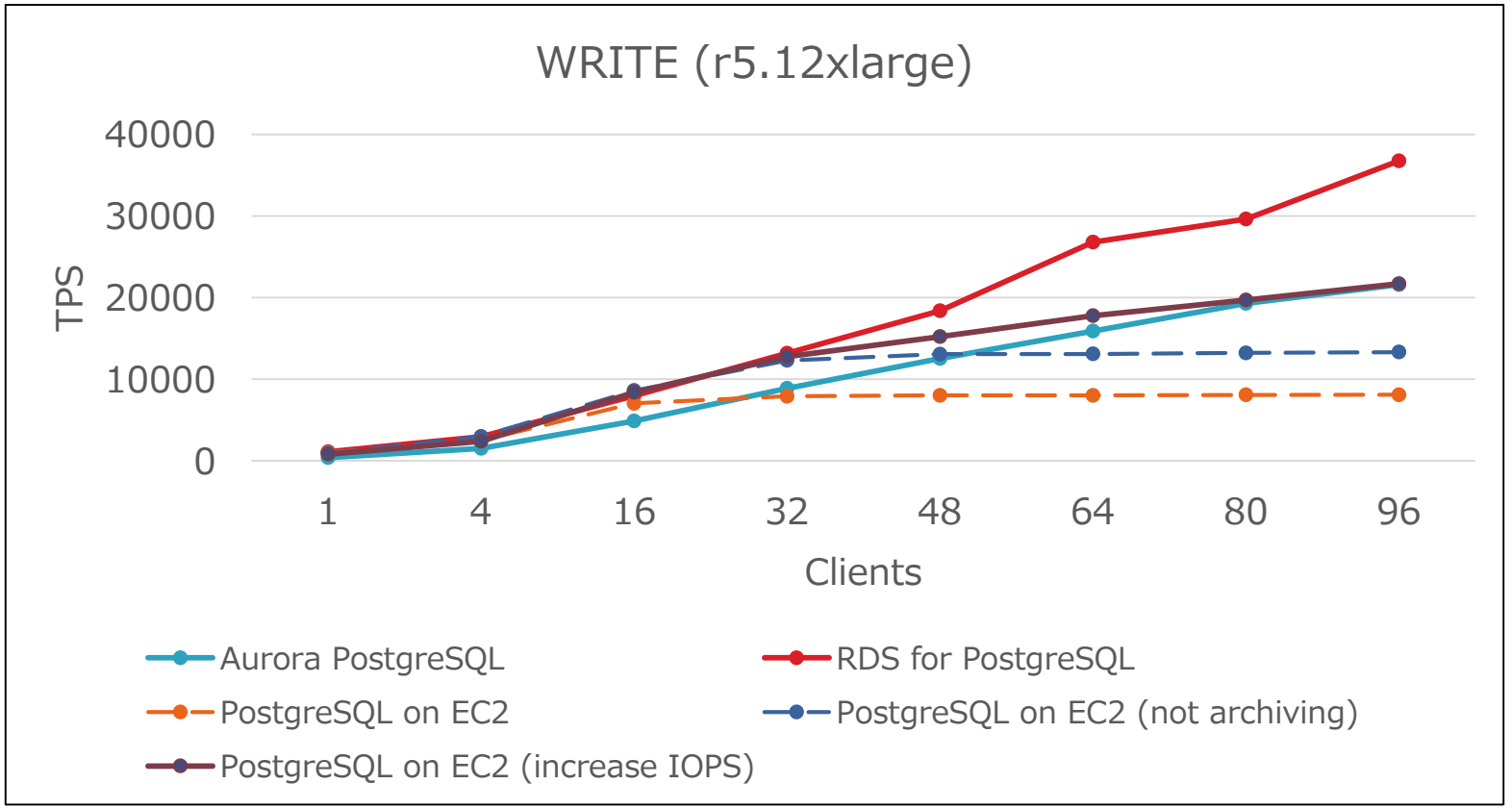
- IOPSを増やす場合の方法
  - AWSでIOPSを増加させたい場合には、次の２種類がある。
    - ✓ ディスクサイズを増やす
    - ✓ プロビジョニングIOPSでIOPSを指定する

      →今回は、ディスクサイズを増やす方式を採用。
  - 理由はコスト面である。

    →1500IOPSとした場合(東京リージョンで算出)
    - ディスクサイズを500GBに増加
      - 0.12(サイズ) x 500 / 30 = 2USD
    - ディスクサイズは100GBのまま、IOPSを1500に指定
      - 0.142(サイズ) x 100 / 30 + 0.074(IOPS) * 1500 / 30 ≒ 4.17USD

検証結果(２)

- 更新系検証(高性能)



→想定した通り、性能の改善がみられた。インスタンスサイズに応じて、適切なストレージを選択すべきである。逆に言うと、事業の規模に応じて、インスタンスサイズだけでなく、ストレージ性能についても目を向けるべきである。

まとめ

- RDS、Aurora、EC2それぞれの性能評価を行ったが、適切に設計すれば、差はそれほど見られなかった(多少、RDSがいいかなと感じた程度)。そのため、運用面など性能以外で選択しても問題はないと感じた、
- ただし、Auroraは、その仕組み上、利用シーンを選ぶと感じたので、自分の利用シーンで予め試したほうが良い。
- 測定結果の図表は、平均をとっているため見えなくなっているが、RDSやAuroraは性能のゆらぎ(2割程度)がみられた。今回は原因追及が途中のため、Performance Insight等各種データは未掲載としたが、今後原因を特定していきたい。ゆらぎが定常的に発生すると仮定すると、RDS やAurora では余裕をもった構成で開始し、実ワークロードでの状況を見ながらScale-Down するなどの対処が必要であると感じた。
- EC2上に構築する場合には、業務の種類によるが、今回の検証ではIOPS制限による影響が大きかったので、注意が必要である。性能が不足している場合には、コストによる解決(WAL領域を大きいディスクを使用、もしくは、プロビジョニングIOPSを使用)して、性能要件を満たすなどの方法が必要である。

性能検証結果
-HammerDB (TPC-C) 編-

検証概要

- 目的
  - TPC-Cワークロード(HammerDB)でEC2/RDS/Auroraの性能を比較検証
  - 一般的なCRUDの混在したOLTPに対するパフォーマンスを以下のような観点/趣旨で確認する
    - インスタンスタイプとサーバの設定内容はなるべく揃える
    - RDSとAuroraによって自動計算されるパラメータはそれに従う
- 検証内容
  - HammerDBでTPC-HのTPMを測定
  - インスタンスタイプは以下2種類
    - r5.2xlarge
    - r5.12xlarge
  - 検証時期:2019年7月

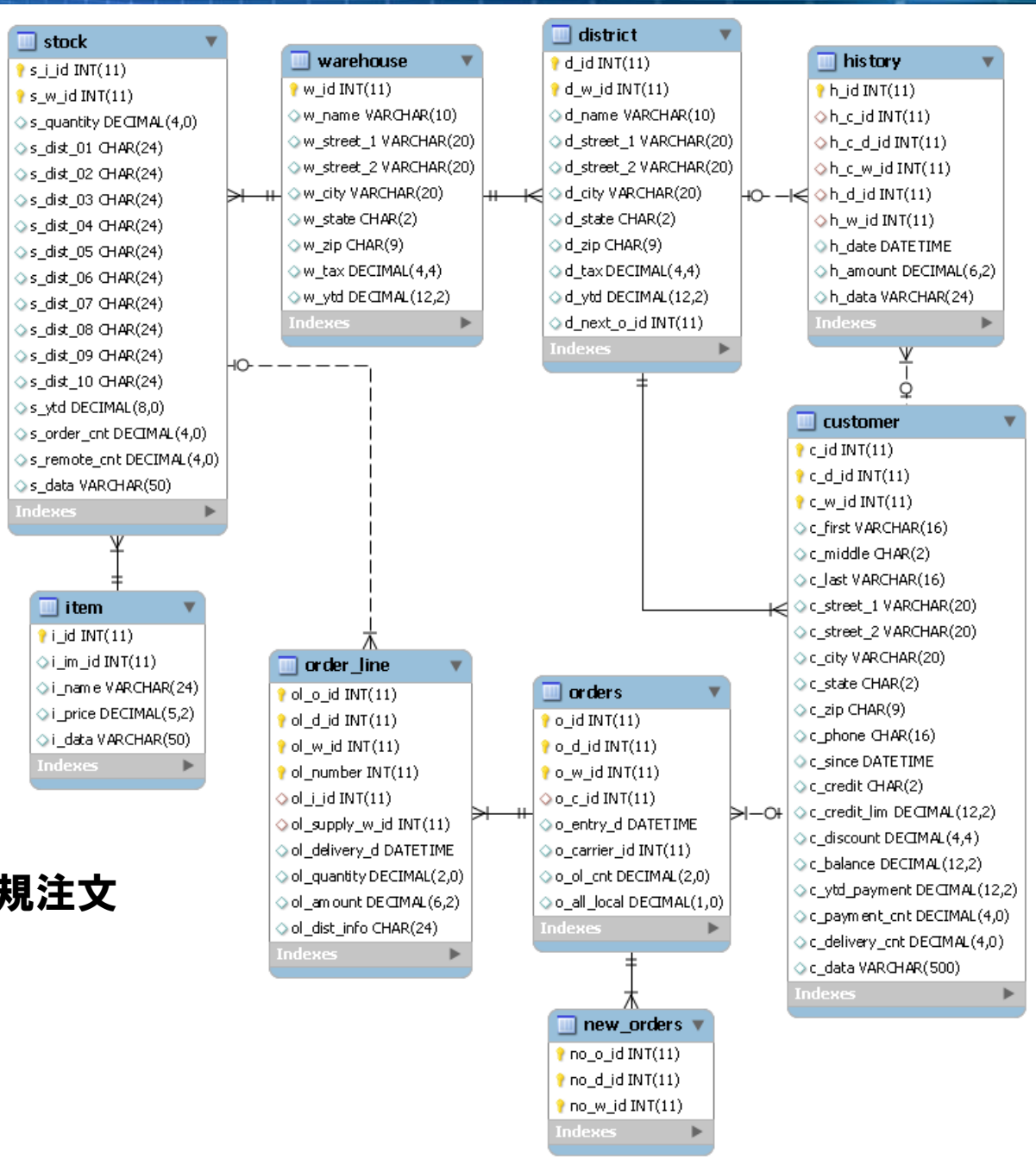
TPC-C概要

- OLTPの性能測定を目的として、卸売業における注文・支払いなどの業務をモデルにしたトランザクションを参照/更新交えて複数種類同時実行する

- 以下の5種類のトランザクションを10:10:1:1:1で連続実行し続け、New-Orderの1分あたりの実行回数(TPM)がスコアとなる
  - New-Order：注文処理(SELECT/UPDATE/INSERT)
  - Payment：支払い処理(SELECT/UPDATE/INSERT)
  - Order-Status：注文状況を確認する処理(SELECT)
  - Delivery：配送処理(SELECT/UPDATE/DELETE)
  - Stock-Level：在庫状況を確認する処理(SELECT)

TPC-Cの参考ER-図

- warehouse：倉庫
- district：配送区域
- customer：顧客
- history：支払い履歴
- item：商品
- stock：在庫
- orders：注文
- order_line：注文明細
- new_orders：未配送の新規注文



TPC-Cの参考SQLとCRUD図

New-Orderの疑似SQL

```
SELECT FROM warehouse JOIN customer;
SELECT FROM district FOR UPDATE;
UPDATE district;
INSERT INTO orders;
INSERT INTO new_orders;

LOOP {
  SELECT FROM item;
  SELECT FROM stock FOR UPDATE;
  UPDATE stock;
  INSERT INTO order_line;
}

COMMIT;
```

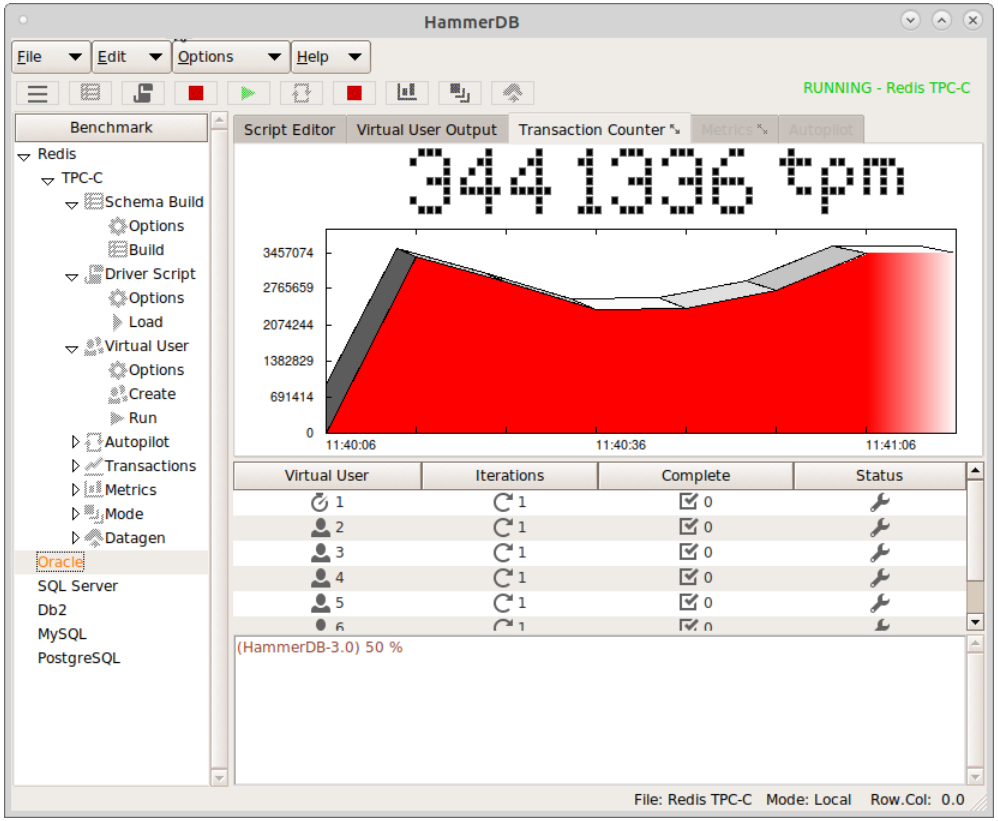
transaction	warehouse	district	customer	history	item	stock	orders	new_orders	order_line
New-Order	R	RU	R		R	RU	C	C	C
Payment	RU	RU	RU	C					
Order-Status			R				R		R
Delivery			U				RU	RD	RU
Stock-Level		R				R			R

測定ツールHammerDBについて

- TPC-CとTPC-Hに相当するベンチマークのデータ生成とクエリ実行/測定までが実施できるツール

- 様々なRDBMSに対応
  - Oracle
  - SQLServer
  - Db2
  - MySQL/MariaDB
  - PostgreSQL
  - Redis

- GUI/CUI両方での利用が可能



検証環境の設定方針

- 全般
  - PostgreSQLのVerは10.7で統一（Auroraの最新に合わせる）
  - EC2/RDS/Auroraでサーバの設定内容はなるべく揃える
- Aurora
  - リードレプリカなし、暗号化は有効
- Aurora/RDS
  - マルチAZはなし、暗号化は有効、ストレージは汎用SSD（120GB）
- EC2
  - ディスクはEBSを汎用SSD（120GB）、暗号化ありにて、1つのみマウント（WALの別ディスク保存設定なし。RDSに寄せている）
  - OSはRHEL 7.6
- インスタンスタイプ
  - r5.2xlarge（8vCPU、64GBメモリ）：標準的なスペックを想定
  - r5.12xlarge（48vCPU、384GBメモリ）：強力なスペックを想定
    ⇒（WG1定点観測に寄せている）

postgresql.conf設定要素

※記載のないものはデフォルト値

手動で変更した設定値
計算式で自動設定される設定値
変更不可な設定値

	EC2	RDS	Aurora	備考
listen_addresses	*	*	*	
max_connections	500	5000	5000	
shared_buffers	16GB 100GB	16237064kB(15G) 48405784kB(46G)	43798032kB(41G) 266227128kB(253G)	上:2xlarge 下:12xlarge
work_mem	64MB 1GB	64MB 1GB	64MB 1GB	上:2xlarge 下:12xlarge
maintenance_work_mem	10GB 20GB	1039MB 3099MB	1039MB 6272MB	上:2xlarge 下:12xlarge
autovacuum	off	off	off	
max_wal_size	1GB 50GB	1GB 50GB	1GB	上:デフォルト 下:未発動用
checkpoint_timeout	5min 60min	5min 60min	1min	上:デフォルト 下:未発動用

HammderDB　設定方針

- データの規模
  - pg_count_ware：200で測定
  - いわゆるScale Factorにあたり、warehouseテーブルのレコード数が200、その他のテーブルは以下のような倍率となる

warehouse	sf x 1
district	sf x 10
customer	sf x 30,000
history	sf x 30,000
item	100,000
stock	sf x 100,000
orders	sf x 30,000
new_orders	sf x 9,000
order_line	sf x 300,000 (approx.)

データベースクラスタのサイズは
約41GBとなる

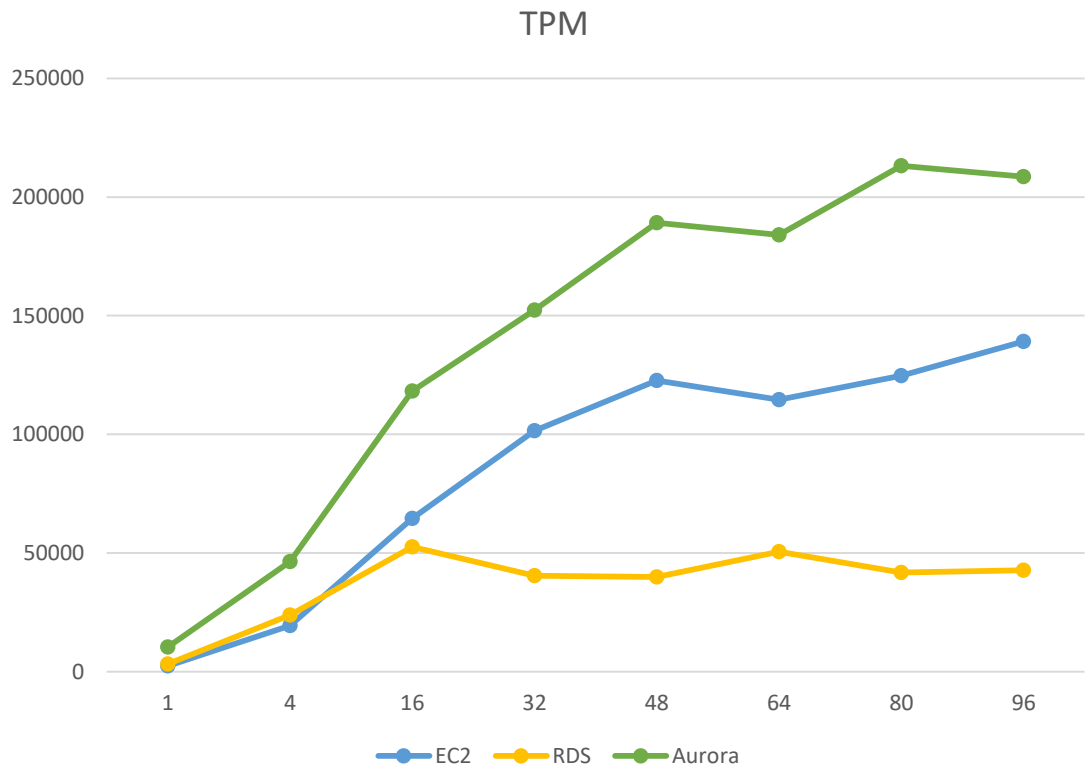
HammderDB　設定方針　その2

- 同時接続クライアント数
  - 1, 4, 16, 32, 48, 64, 80, 96　で実施
  - TPC-Cは更新も実施してしまうため、各測定回ごとにデータ構築済みのスナップショットを復元、起動直後のPostgreSQLサーバに測定を実施
- 測定時間とprewarmにあたる時間
  - pg_rampup ： 3min ⇒実行開始から何分後測定開始するか
  - pg_duration ： 5min
    ⇒測定時間。rampup後この期間のトランザクション処理数をカウントし、1分間の平均に直した後TPMとする
- 実行インスタンス
  - OSはRHEL 7.6
  - インスタンスタイプはr5.2xlarge (8vCPU、64GBメモリ)

測定結果その1

- r5.2xlarge (8vCPU、64GBメモリ)
- EC2とRDSはチェックポイント関連の設定を1GB/5minで測定 (デフォルト値)
- AuroraとEC2は単調増加。RDSが伸び悩み。

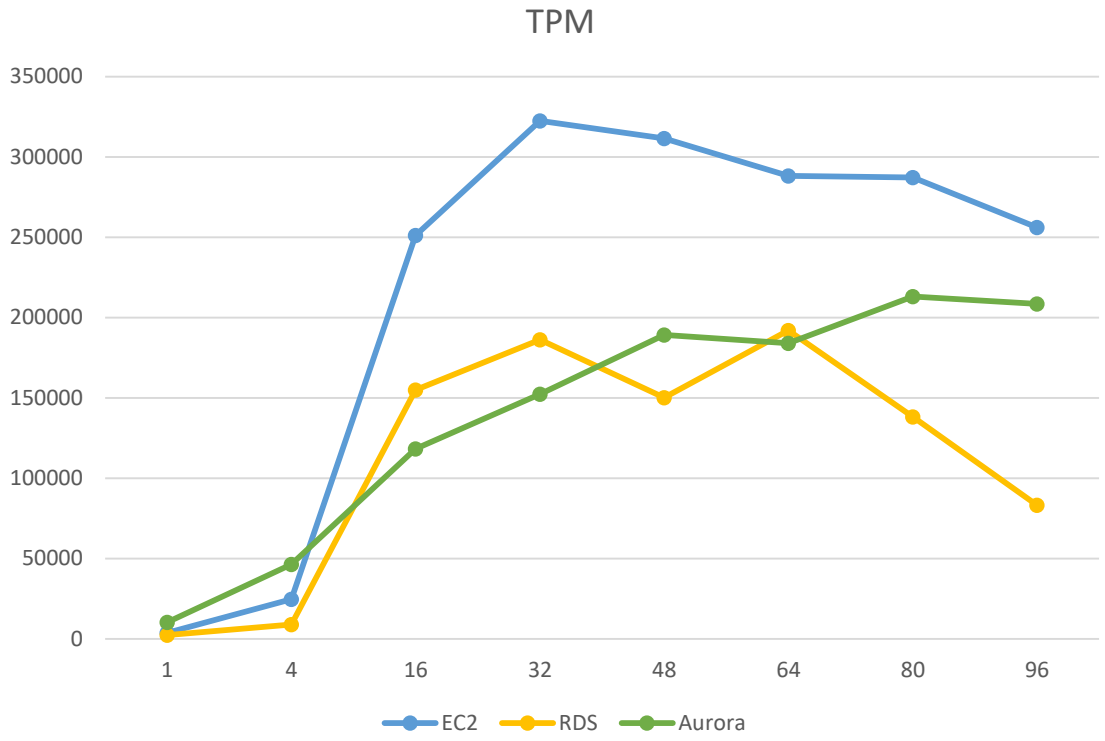
	TPM		
接続数	EC2	RDS	Aurora
1	2366	3182	10355
4	19327	23865	46307
16	64537	52535	118273
32	101560	40452	152394
48	122611	39913	189160
64	114559	50546	183977
80	124663	41782	213088
96	139174	42663	208573



測定結果その2

- r5.2xlarge (8vCPU、64GBメモリ)
- EC2とRDSのチェックポイント関連の設定を未発動用に設定 (50GB/60min)
- Auroraの数値は測定結果その1と同一で比較
- EC2がチェックポイントが発生しない分最もTPMが高くなったが、RDSが伸び悩み

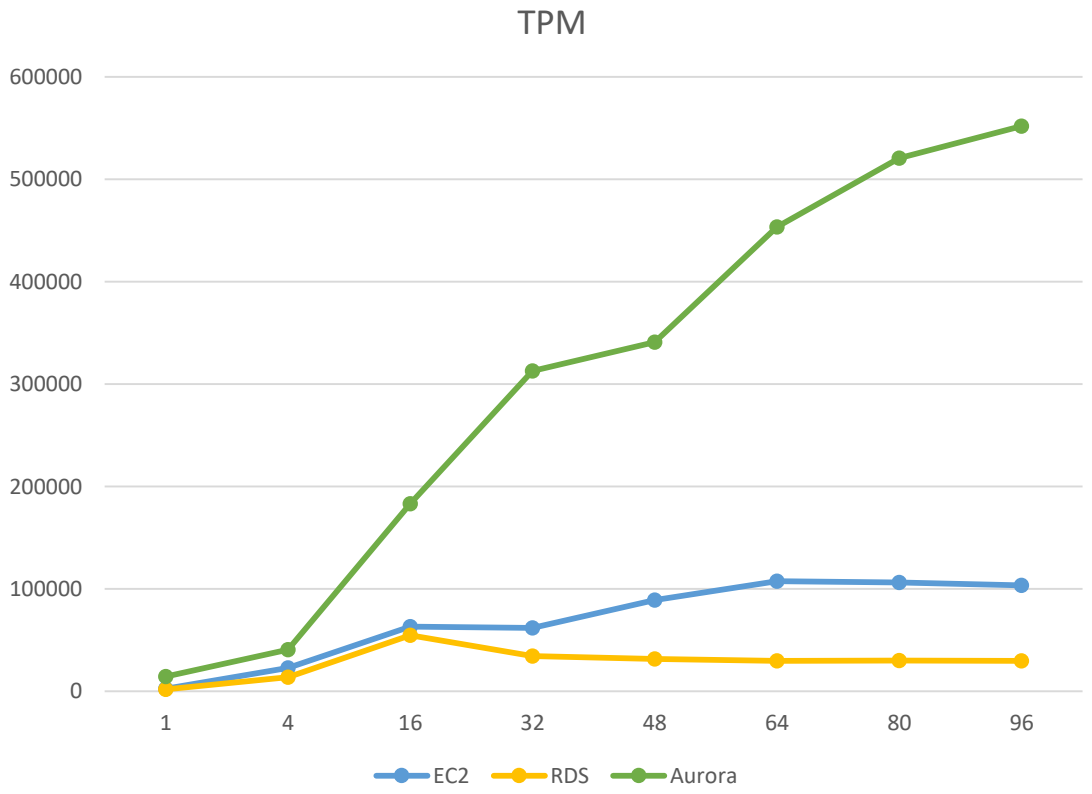
	TPM		
接続数	EC2	RDS	Aurora
1	3677	2329	10355
4	24705	8898	46307
16	251147	154862	118273
32	322378	186157	152394
48	311442	150066	189160
64	288093	191897	183977
80	287157	138244	213088
96	256154	83146	208573



測定結果その3

- r5.12xlarge (48vCPU、384GBメモリ)
- EC2とRDSはチェックポイント関連の設定を1GB/5minで測定 (デフォルト値)
- Auroraが圧倒的かつ単調増加。EC2/RDSはr5.2xlargeと変わらず。

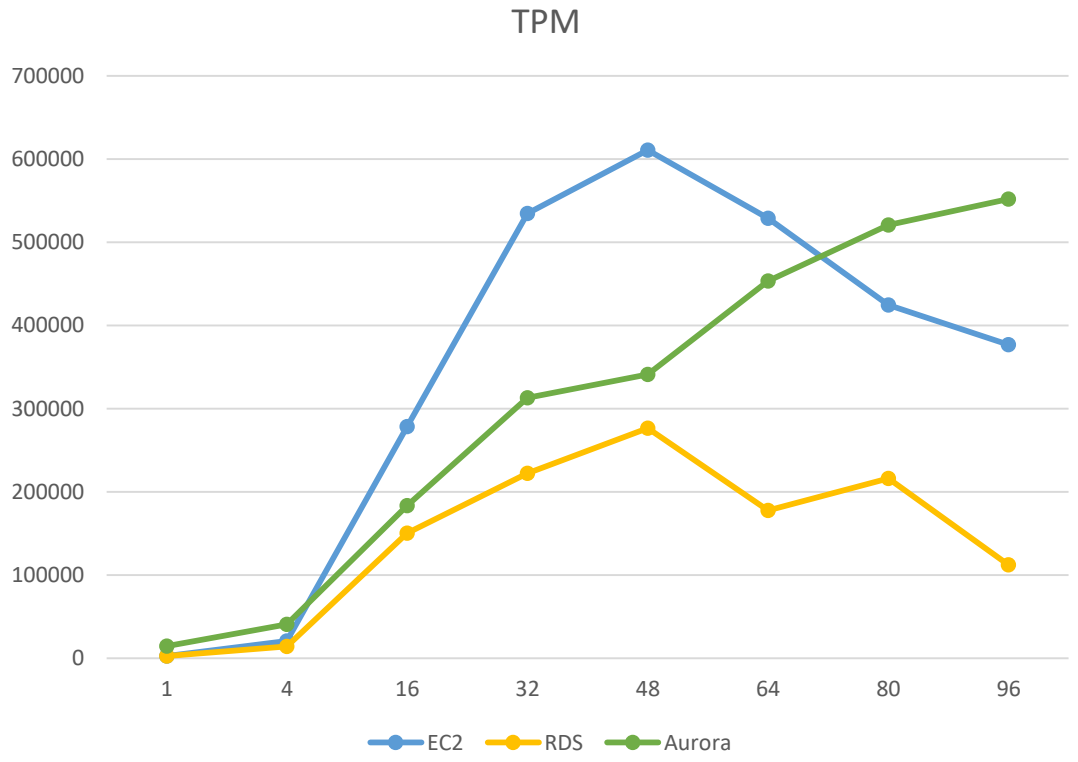
	TPM		
接続数	EC2	RDS	Aurora
1	2461	1850	14461
4	22827	13941	40782
16	63299	54889	183300
32	61879	34536	312879
48	88981	31667	341042
64	107519	29640	453326
80	106346	29932	520742
96	103346	29640	551930



測定結果その4

- r5.12xlarge (48vCPU、384GBメモリ)
- EC2とRDSのチェックポイント関連の設定を未発動用に設定 (50GB/60min)
- Auroraの数値は測定結果その3と同一で比較
- EC2は接続数＞コア数で性能劣化。RDSも同様。

	TPM		
接続数	EC2	RDS	Aurora
1	2350	2528	14461
4	20662	14019	40782
16	278255	150427	183300
32	534690	222200	312879
48	610655	276380	341042
64	528663	177497	453326
80	424607	215905	520742
96	376857	112112	551930



結果の考察と指摘事項　その1

- CloudWatchでメトリクスを見ていると以下の傾向が見られた
  - IOPSに圧倒的な差がある。EC2/RDSは個別に配慮、設定していないため3000どまりでネックとなっていたが、Auroraは2xで約30万、12xでは約100万近くのWriteIOPSを発揮していた
  - なるべく深く考えずに立ち上げ、利用するという趣旨もあったが、検証の指針としては事前の考慮から抜けてしまっていた
  - 上記IOPS制限の違いに起因してか、CPU利用率もAuroraは2x、12xともに100%近くをマークするが、EC2/RDSは2xは30%～50%、12xは10%の水準（特に12xでI/Oネックあり）

- 全般的にI/Oの条件面でEC2/RDSと圧倒的に不利であった
  - 上記3000IOPS制限に加え、スナップショットからの復元 ⇒ 直後の測定による、ファーストタッチペナルティが効いている
  - TPC-Cはデータに変更を加えるため、各測定ではデータ構築済みのスナップショットからの復元 ⇒　測定開始をしており、Aurora以外はファーストタッチペナルティが発生する

結果の考察と指摘事項　その2

- EC2/RDSはでconfにてチェックポイントの発動を抑制し、Writeの I/O要素を抑えた場合は、EC2がTOPとなった。
  - EC2とRDSはReadIOPSが3000どまりとなってしまうが、TPC-Cは参照(Read)より更新の影響が大きいか、チェックポイントなしの性能向上は大きかった。WriteのIOPSは600～2000など
  - CPU利用率については、I/Oネックが多少解消され、接続数によっては2xでは70～80%をEC2/RDSでも使えていた(12xは20～30%)

- EC2はarcivemodeがoffとしていたため、バックアップとHA面のコストでは一番有利であった
  - Auroraはまったく別の仕組みで常時バックアップをしているためそもそも比較はできない
  - RDSについては自動バックアップがONであるとWALarchiveをS3に送るのでここがネックになる　⇒　結果として一番伸び悩んだ
  - WALについてはEC2追加検証の試行で40GB(8分間)の出力が見られた

まとめやその他

- IOPSにおいてAuroraで圧倒的な値が出た
  - 2xlargeで30万近く、12xlaregeでは100万近くの数値をCloudWatchで確認
  - EC2とRDSはプロビジョンドでも最大80000までしか確保できない
- EC2側は最大性能を突き詰める方針としてチェックポイントの抑制（更にはWAL書き込み先の分離）ができるが、その場合でも接続数＞CPUからの壁がある。Auroraは接続数＞CPU後も伸び続けた
- 現実的な可用性も加味した条件の測定として、Auroraの圧倒的な性能を確認することができた（特に、高スペックインスタンス）
  ⇒CPUはほぼ100%を使い切る（I/Oネックにならない？）
  ⇒8vCPUですら、傾きは鈍化しつつも96接続まで線形に増加した
- RDS/Auroraは接続数増加に対する測定結果に多少のブレが出たが、AZの違いも影響しているのではと後から気づいた。こちらも配慮が漏れたがAZ指定なしで復元し、HammerDBの実行インスタンスとRDS/AuroraでAZが分かれた場合、NWネックによるTAT悪化が発生していたのではないか

性能検証結果
-HammerDB (TPC-C) 編　追加検証-

HammerDB　追加検証の計画

- 検証目的
  - RDSとEC2に不利であった要素を改善する
    - プロビジョンドIOPSでIOネックを低減する
    - EC2ではWALアーカイブも実施する
    - ファーストタッチペナルティを回避する
- 検証概要（当初）
  - r5.2xlargeとr5.12xlarge、PostgreSQLのVerやHammerDB、パラメータなどの設定は同一
  - EC2とRDSに10000のプロビジョンドIOPSを設定する
  - EC2に別のEBSをマウントし、WALアーカイブを保存する
  - 測定前にselect count(*)を全テーブルに実施し、ファーストタッチペナルティを回避する
  - Auroraは変更要素がないので、旧検証結果から流用する

追加検証の計画　その2

- プレ測定の実施
  - 2xlargeのRDSにおいて、48接続で約22万TPMの性能を発揮。旧検証時のAuroraの約19万を上回る
    ⇒Auroraを再測定すると、約17万
- 測定方針の変更
  - Auroraは独自のストライピングによる10GB単位でのIO分散があり、40GBのデータではあまり性能を発揮できない。100GBに増加させる
  - Auroraも再測定を実施する
  - 接続数を絞る（2x：32/48/64）（12x：48/64/80）
- 検証時期
  - 2020年2月実施

追加検証の設定方針

- 全般
  - PostgreSQLのVerは10.7で統一
  - リージョンは東京、AZはap-northeast-1aで統一
- Aurora
  - リードレプリカなし、暗号化有効
- RDS
  - マルチAZなし、暗号化有効、ストレージは
    プロビジョンド IOPS SSD(io1)で200GB:10000IOPS
- EC2
  - ディスクはEBSを2つマウント。片方はプロビジョンド IOPS SSD(io1)で
    200GB:10000IOPSとし、PGDATAとWALを格納。もう一つは汎用SSDで
    100GBを確保、WALアーカイブのコピー先とする(両方暗号化あり)
  - OSはRHEL 7.6
- インスタンスタイプ
  - r5.2xlarge(8vCPU、64GBメモリ)
  - r5.12xlarge(48vCPU、384GBメモリ)

postgresql.conf設定要素

※記載のないものはデフォルト値

手動で変更した設定値
計算式で自動設定される設定値
変更不可な設定値

	EC2	RDS	Aurora	備考
listen_addresses	*	*	*	
max_connections	500	5000	5000	
shared_buffers	16GB 100GB	16237064kB(15G) 48405784kB(46G)	43798032kB(41G) 266227128kB(253G)	上:2xlarge 下:12xlarge
work_mem	64MB 1GB	64MB 1GB	64MB 1GB	上:2xlarge 下:12xlarge
maintenance_work_mem	10GB 20GB	1039MB 3099MB	1039MB 6272MB	上:2xlarge 下:12xlarge
autovacuum	off	off	off	
max_wal_size	1GB 50GB	1GB 50GB	1GB	上:デフォルト 下:未発動用
checkpoint_timeout	5min 60min	5min 60min	1min	上:デフォルト 下:未発動用

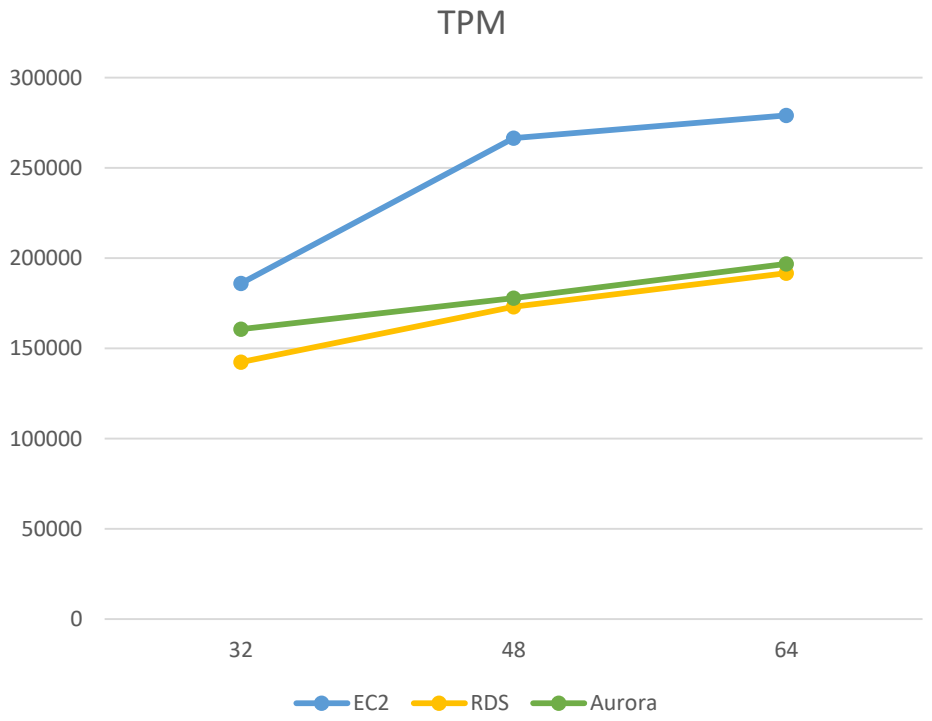
HammderDB　設定方針

- データ規模
  - pg_count_ware：500　PGDATAのディレクトリサイズは約101GB
- 同時接続クライアント数
  - r5.2xlarge：32/48/64　r5.12xlarge：48/64/80で実施
  - 各測定回ごとにデータ構築済みのスナップショットを復元後、DB起動後にselect count(*)とALNAYZEを実施する
- 測定時間とprewarmにあたる時間
  - pg_rampup ： 3min　pg_duration ： 5min
- 実行インスタンス
  - OSはRHEL 7.6
  - インスタンスタイプはr5.2xlarge(8vCPU、64GBメモリ)

測定結果その1

- r5.2xlarge (8vCPU、64GBメモリ)
- EC2とRDSが大きく躍進。旧検証におけるAuroraのTOPを覆す
- バックアップ要素で一番オーバーヘッドの少ないEC2のTPMが最も高い

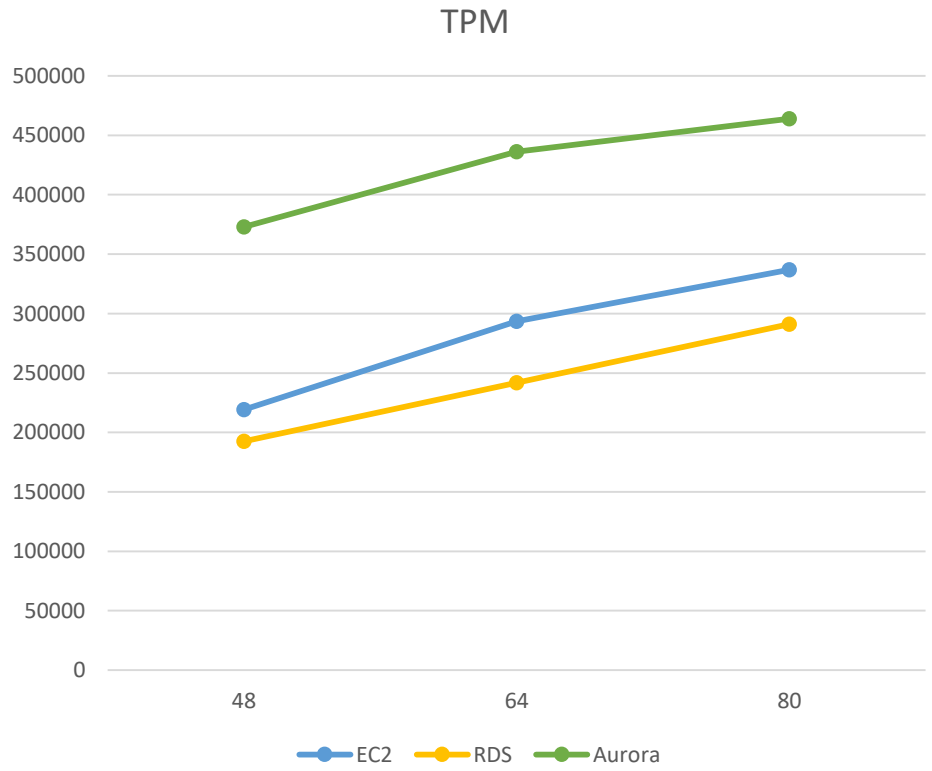
	TPM		
接続数	EC2	RDS	Aurora
32	185963	142304	160673
48	266534	172968	177881
64	279138	191758	196805



測定結果その2

- r5.12xlarge (48vCPU、384GBメモリ)
- こちらもEC2とRDSが躍進はしたが、Auroraが最も高いTPMを維持
- 特にRDSが旧検証と比較し、大きく持ち直した

	TPM		
接続数	EC2	RDS	Aurora
48	219317	192546	372826
64	293387	241912	436220
80	336939	291195	463882



測定結果その1（r5.2xlarge）について

- CPUとIOPSの状況（CloudWatchによる高負荷時の値を抜粋）

	EC2	RDS	Aurora
CPU	70-90%	70-80%	90-95%
WriteIOPS	CloudWatch上はEBSopsとして出ており5分間隔のため比較不可	10000-11000	90000-100000
ReadIOPS		4200-6600	Select count(*)時：5.9-7.1万 測定時：4000-5000

- プロビジョンドIOPSの効果か、CPU使用率含め、EC2とRDSは性能をしっかり出すことができた。
  - I/Oネックがほぼなくなった状態では、このデータ規模とインスタンスサイズ、ワークロードではAuroraが特別有利ではないと示された
  - Auroraはそもそも常時かつ独自のバックアップ/PITR仕様によるハンデがある
- Performance Insightsの傾向（EC2は除く）
  - RDSはWALWriteLockの割合が一番大きく、22-38%を占めた
  - AuroraはXactSyncの割合が一番大きく、20-40%を占めた
  - いずれもIO部分の待機が大きい、ということになる

測定結果その1（r5.2xlarge）について

- CPUとIOPSの状況（CloudWatchによる高負荷時の値を抜粋）

	EC2	RDS	Aurora
CPU	10-30%	13-25%	34-38%
WriteIOPS	CloudWatch上はEBSopsとして出ており5分間隔のため比較不可 sarにおけるはreadは0	12000-14000	22.4-27.8万
ReadIOPS		Select count(*)時： 1400-1500 測定時：0	Select count(*)時： 9.7-13.5万 測定時：0

- メモリ384GBのため、select count（*）でオンキャッシュとなった
  - ReadI/Oが0の条件ながらAuroraがTOPに。WriteIOPSで差がついたか
  - CPUは全体的にあまり使い切れておらず、WriteI/Oがネックになったとも思うが、接続数によるTPMの更なる線形増加も見込めそうである
- Performance Insightsの傾向（EC2は除く）
  - RDSはWALWriteLockの割合が一番大きく、43-67%を占めた
  - AuroraはXactSyncの割合が一番大きく、34-58%を占めた
  - IO待機部分の割合がさらに大きくなった

追加検証の考察

- 旧検証では振るわなかったEC2/RDSに対し、プロビジョンドIOによる3000⇒10000の効果は大きかった
  - sar -dの指標とは違い、I/Oのマージなどが発生する
    ⇒最大で256KiB/1IO
  - 逆に1I/Oが256KiB以上であれば分割されるということ
  - 思っている感覚以上の効果をIOPSブーストで出せる可能性がある

- 同様にファーストタッチペナルティの解消効果も大きかったといえる。性能測定では注意が必要
  - 副次効果として12xlargeではオンキャッシュでの測定となった

- Performance Insightsからネックを探るのであれば、いずれもIO起因の待機イベントが多くを占めた
  - I/Oのブーストにより更に伸びる余地がある

追加検証のまとめ

- 旧検証ではAurora圧倒有利という感覚だが、EC2とRDSについては、AWSならではの仕様を理解した条件設定と検証方針の策定が必要であった
- Auroraはそういったものに気を配らずに性能を出せ、かつバックアップ/HA面も盤石なのは素晴らしいと改めて思う
- CloudWatchとPerformance Insightsは使いこなせれば便利であると感じたが、EC2ではいずれも使えない(CloudWatchは5分間隔固定)ので、自前での性能監視基盤が必要となる
- EC2とRDSは気を配らないと特にIOPS制限で性能に大きく蓋がかかる(インスタンスサイズだけ大きくしても無意味)ので注意が必要。性能面重視ならやはりAurora

備考

ライセンス

本作品はCC-BYライセンスによって許諾されています。ライセンスの内容を知りたい方はこちらでご確認ください。 文書の内容、表記に関する誤り、ご要望、感想等につきましては、PGEConsのサイトを通じてお寄せいただきますようお願いいたします。

- Amazon Web Services、"Powered by Amazon Web Services"ロゴ、Amazon EC2、Amazon S3、Amazon Relational Database Service (Amazon RDS) およびAmazon Auroraは、米国その他の諸国における、Amazon.com, Inc.またはその関連会社の商標です。
- IBMおよびDb2は、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。
- Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
- Red HatおよびShadowman logoは、米国およびその他の国におけるRed Hat,Inc.の商標または登録商標です。
- Microsoft、Windows Server、SQL Server、米国 Microsoft Corporationの米国及びその他の国における登録商標または商標です。
- MySQLは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。 文中の社名、商品名等は各社の商標または登録商標である場合があります。
- Oracleは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。 文中の社名、商品名等は各社の商標または登録商標である場合があります。
- PostgreSQLは、PostgreSQL Community Association of Canadaのカナダにおける登録商標およびその他の国における商標です。
- TPC, TPC Benchmark, TPC-B, TPC-C, TPC-E, tpmC, TPC-H, TPC-DS, QphHは米国Transaction Processing Performance Councilの商標です。
- その他、本資料に記載されている社名及び商品名はそれぞれ各社が 商標または登録商標として使用している場合があります 。

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（2019年度WG3）	NECソリューションイノベータ株式会社	サポートサービス事業部	湯村　昇平
	株式会社日立製作所	Software CoE　OSSソリューションセンタ	稲垣　毅
	株式会社日立製作所	Software CoE　OSSソリューションセンタ	牧田　伸行