PGECons
PostgreSQL Enterprise Consortium

2024年度WG3活動成果報告
Amazon Aurora PostgreSQL Limitless Database 検証
～実機検証結果～

PostgreSQL エンタープライズコンソーシアム
WG3 パブリッククラウド検証チーム

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。

実機検証概要

- 以下2つのデータベースに対し、ツール（HammerDB）を用いて
  TPC-Cベンチマークを実行し、スループットを比較することで、
  Limitless Databaseの性能特性を確認した。
  - Amazon Aurora PostgreSQL（以降、Provisionedとする）
  - Amazon Aurora PostgreSQL Limitless Database（以降、Limitless Databaseとする）

- 検証結果の概要は以下の通り
  - TPC-CにおいてProvisionedと同スペック相当のLimitless Databaseでは、
    スループットはProvisioned ＞ Limitless Databaseとなった
    - 今回のケースにおいてはLimitless Database特有のチューニングに加え、
      Provisionedの8倍相当のスペック（maxACU）を設定したところ、
      高い同時アクセス数においてProvisionedに近いスループットが出たと思われる
  - Limitless Databaseの性能を引き出すためには特有のチューニングが必要

- 次スライド以降で検証内容や結果、考察を説明

実機検証概要

- 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
  - 以下5種類のトランザクションを10:10:1:1:1で実行し続け、New-Orderの1分毎の実行回数(New-Order Per Minutes 以降、NOPMとする)がスコアとなる。
    - New-Order : 注文処理
    - Payment : 支払い処理
    - Order-Status : 注文状況を確認する処理
    - Delivery : 配送処理
    - Stock-Level : 在庫状況を確認する処理
  - それぞれのトランザクションは書き込み、読み込み処理が混在 (以下はCRUD表)

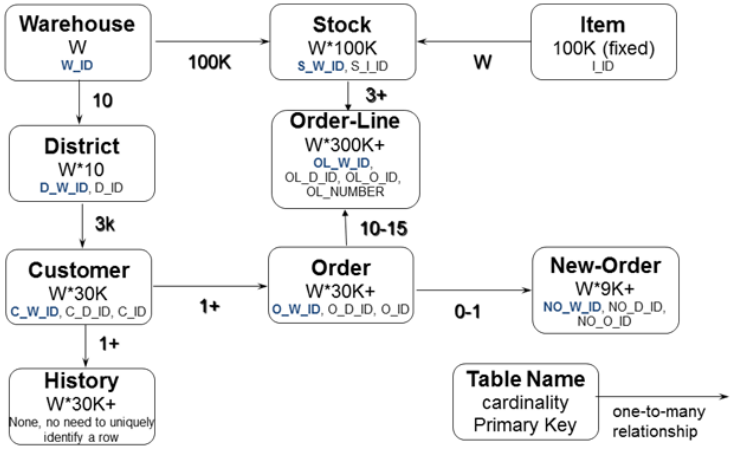
	Warehouse	District	Customer	History	Item	Stock	Orders	New_orders	Order_lines
New-order	R	RU	R		R	RU	C	C	C
Payment	RU	RU	RU	C					
Order-Status			R				R		R
Delivery			U				RU	RD	RU
Stock-Level		R				R			R

実機検証概要

- データの規模
  - Scale Factor (sf) : 200で測定
    - 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.)

テーブル一覧



ER図[1]

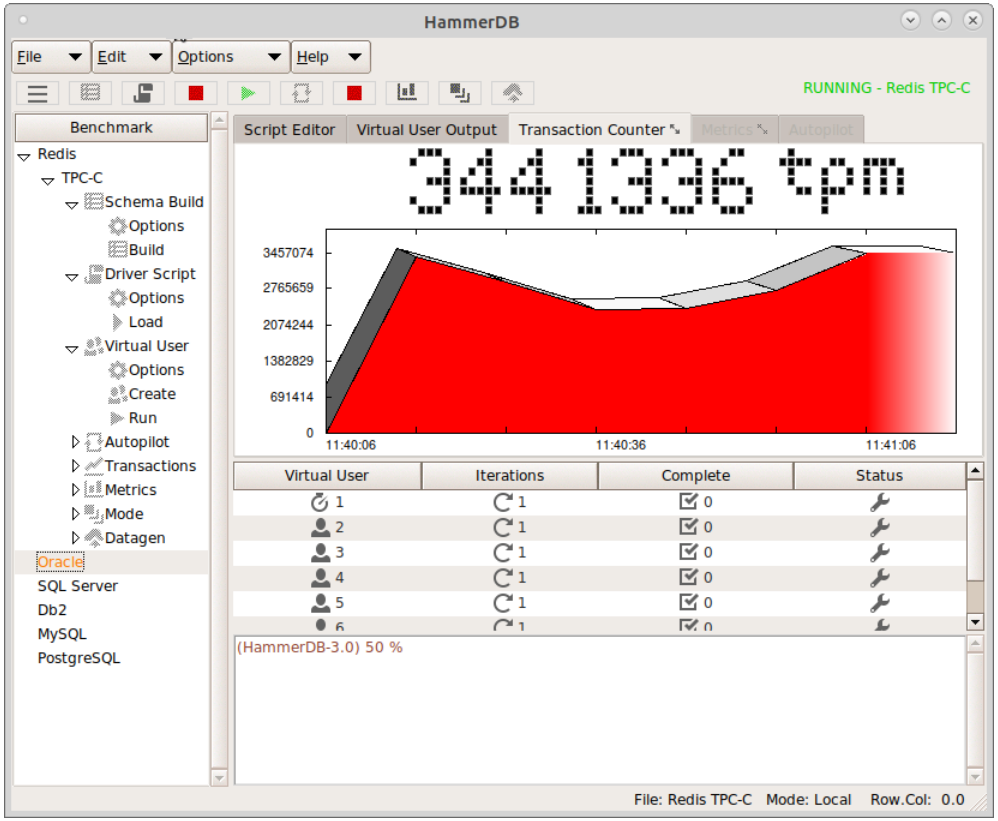
[1] : https://www.hammerdb.com/docs/ch03s05.html#d0e1184

実機検証概要: 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



実機検証概要：HammerDB実行方針

- 比較にはTPC-Cのベンチマーク指標の一つである NOPM (New Orders per minute) を利用
  - TPC-Cのテーブルから取得されるメトリクスで特定のデータベース実装に依存しないため、本検証ではこれを採用
  - 詳細は、4. Comparing HammerDB results を参照のこと

- 測定時間とprewarmにあたる時間
  - pg_rampup ： 4min ⇒ 実行開始から何分後測定開始するか
  - pg_duration ： 5min ⇒ 測定時間。rampup後この期間でNOPMを取得する

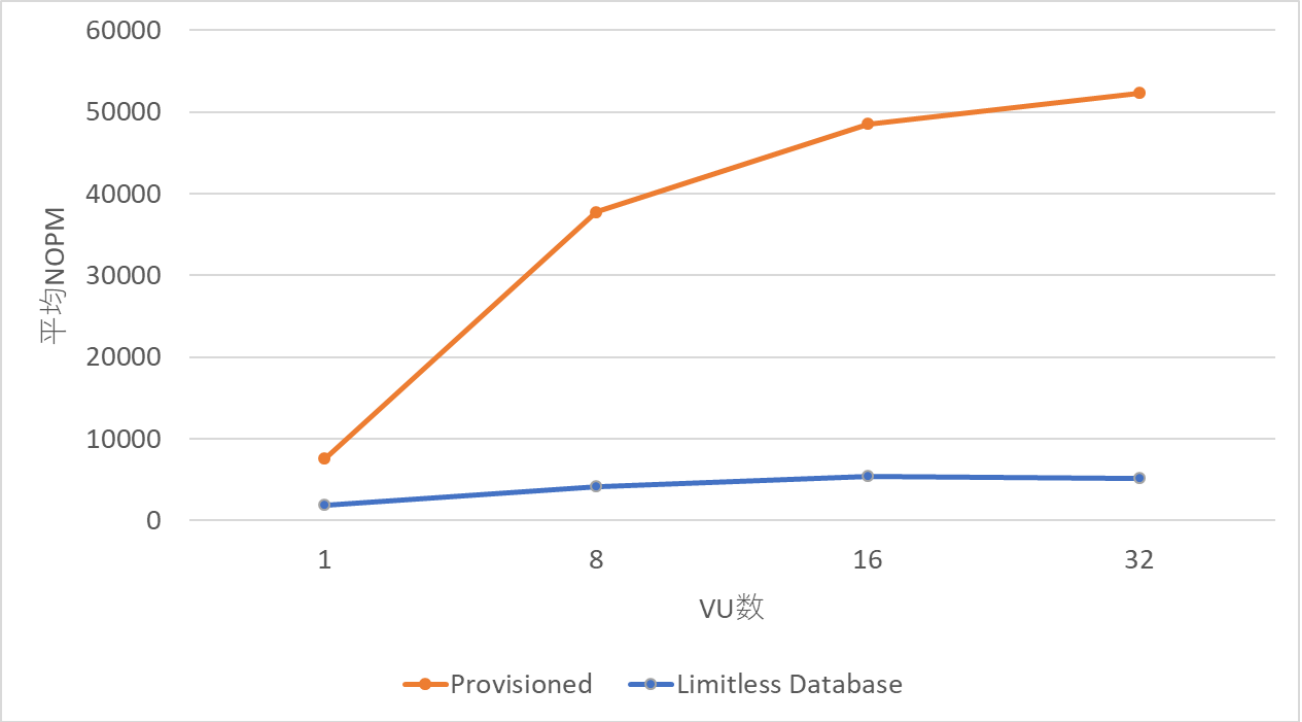
- 実行インスタンス
  - 負荷掛けサーバのスペック: r5.2xlarge (vCPU:8/メモリ:64GiB)
  - HammerDB 4.12 をこれにインストールし、GUIにて実施

実機検証概要：データベース側設定事項

- Provisioned側設定事項
  - インスタンスサイズ（db.r7g.xlarge）・・・vCPU:4/メモリ:32GiB
  - PostgreSQLバージョンは16.4
  - クラスタパラメータグループは、default.aurora-postgresql16を内容変更せず利用
- Limitless Database側設定事項
  - minACU16,maxACU16 … ルーター/シャード数=2/2
    - minACU/maxACUはシャードグループ構築時に設定する、この範囲で利用状況によりACUが増減する
    - 設定するminACUによりルーターとシャードの数が決められており、ルーター/シャード数によってmaxACUの上限が決まる
    - 詳細は、Aurora PostgreSQL Limitless Database を使用する DB クラスターの作成 - Amazon Aurora を参照のこと
    - ACU1単位は約2Gibのメモリに対応するとされており、Provisionedと合わせる方針で決定
  - PostgreSQLバージョンは16.4
  - クラスタパラメータグループは、default.aurora-postgresql16を内容変更せず利用
  - 可用性設定は、AZレベルでのシャードのスタンバイがない状態
    - 構築時 [DB シャードグループのデプロイ] にて「コンピューティングの冗長性なし」を選択
    - 参考：既存の Aurora PostgreSQL Limitless Database DB クラスターに DB シャードグループを追加する - Amazon Aurora
  - テーブル設定: 右図の様にTPC-Cのテーブルの種別（table_type）と分散キー（distribution_key）を設定
    - TPC-Cにおける最適なテーブル設定だと思われるHammerDBの分散DB設定を参考にした

```
tpcc=> SELECT * FROM rds_aurora.limitless_tables;
 table_gid | local_oid | schema_name | table_name | table_status | table_type | distribution_key
-----------+-----------+-------------+------------+--------------+------------+-----------------
     16008 |     68742 | public      | order_line | active       | sharded    | HASH (ol_w_id)
     16000 |     68654 | public      | customer   | active       | sharded    | HASH (c_w_id)
     16001 |     68666 | public      | district   | active       | sharded    | HASH (d_w_id)
     16002 |     68678 | public      | history    | active       | sharded    | HASH (h_w_id)
     16003 |     68681 | public      | item       | active       | reference  |
     16004 |     68693 | public      | warehouse  | active       | sharded    | HASH (w_id)
     16005 |     68705 | public      | stock      | active       | sharded    | HASH (s_w_id)
     16006 |     68718 | public      | new_order  | active       | sharded    | HASH (no_w_id)
     16007 |     68730 | public      | orders     | active       | sharded    | HASH (o_w_id)
```

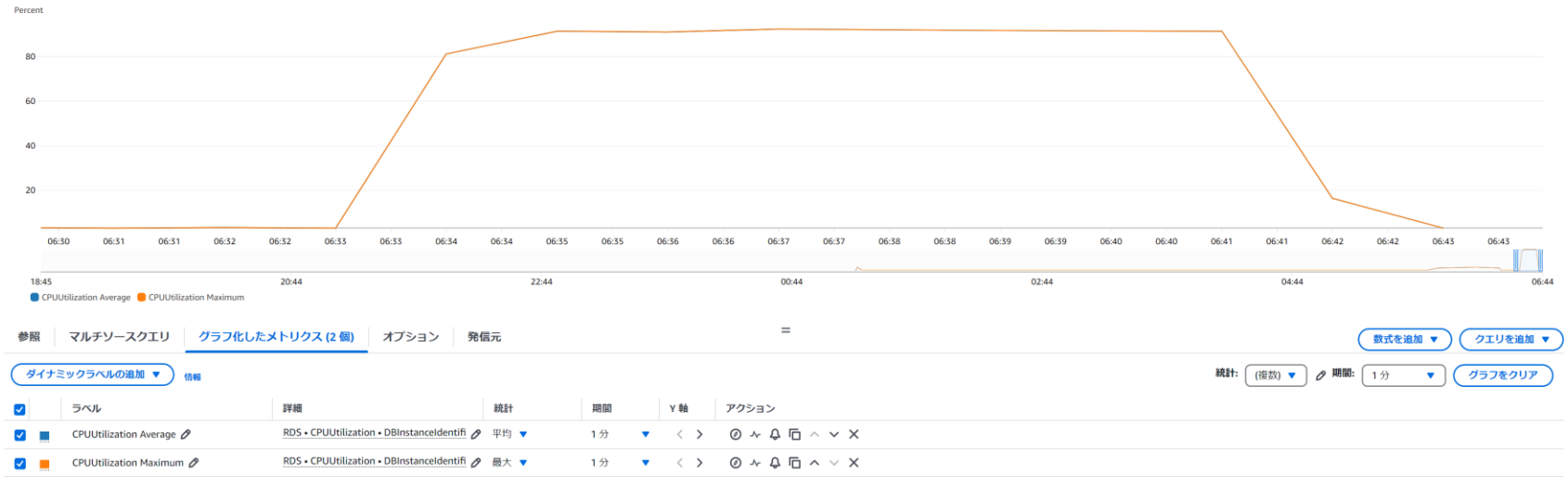
実機検証 計測結果



- **傾向**
  - Limitless DatabaseはProvisionedと比較すると大きく劣るスループット
    - VU数（同時アクセス数）を増やしてもスループットはあまり増加せず

実機検証：結果考察

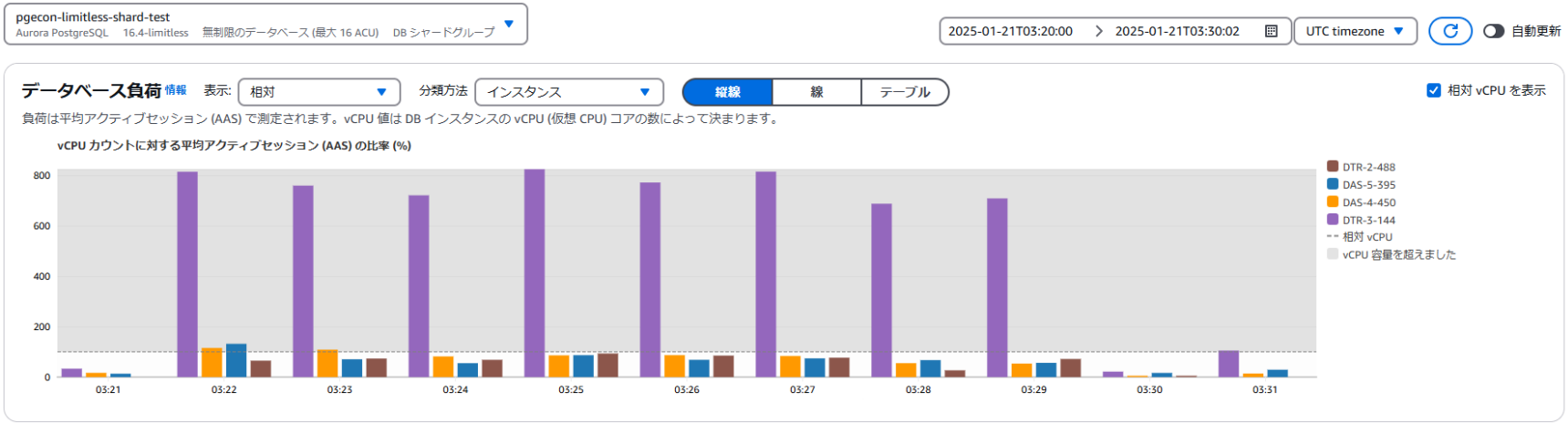
- ProvisionedのCPU使用率 (CPUUtilization)
  - VU：32で100%近くまで上昇しているが、これはリソースを正しく活用している結果と想定され、その他問題となる箇所は見当たらない



Provisioned VU数 32のCPU使用率

実機検証：結果考察

- Limitless Databaseは全てのリソースを使い切れていないことが性能が悪い原因と推測
  - VU数：8の時点で一部リソースが逼迫している
    - 下図の接頭辞:DTRがルーター、DASがシャードの負荷状況
    - 特にルーターの負荷の偏りが大きい
  - Limitless Databaseはルーター,シャードのDBLoadの値を確認
    - Performance Insightsで確認可能な負荷状況のメトリクス
    - CPU使用率を直接表したメトリクスではないことには注意



Limitless Database VU数：8のDBLoad

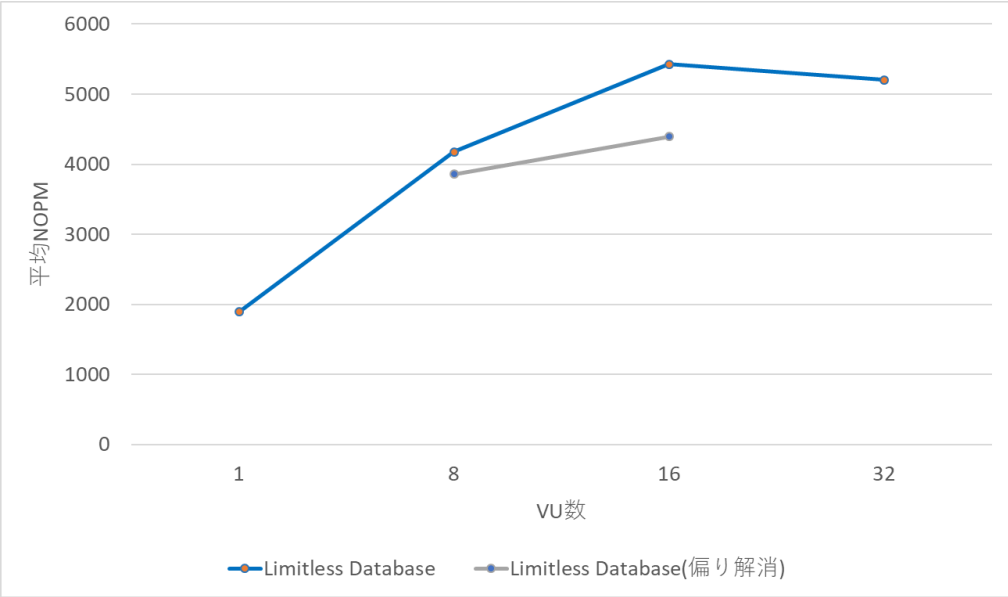
実機検証：結果考察

- ルーター間で負荷が偏る原因、および対処策
  - Limitless Databaseのエンドポイントは、接続タイミング（時間判定）ごとにラウンドロビン方式で接続先ルーターを切り替える仕様となっており、計測開始と同時に一斉にDBへのアクセスが発生する方法だと、同じルーターにアクセスが集中的に振り分けられてしまう
  - ルーターのエンドポイントに対して直接アクセスし接続先ルーターが均等になるように設定
    - シャードグループのエンドポイントは、複数のルーターの個々のFQDNがDNSルーティングされている
    - HammerDBに使われるlibpqの機能を用いて、個々のルータのFQDNをすべて羅列し、load_balance_hostsオプションを付与
    - 詳細は、34.1. データベース接続制御関数 を参照のこと

- シャード側は？
  - Limitless DatabaseはテーブルのPrimary Keyのハッシュ値でデータ格納先シャードを決定
  - Warehouseテーブルは今回のケースだと200のPK:倉庫ID（w_id）を持っているが、HammerDBのデフォルト設定だとVU毎に1つの倉庫IDしか利用されないため、負荷が偏る
  - これをすべての倉庫IDを利用するオプションを利用することで解決
    - HammerDBの”All Warehouse”オプションを有効化

- 上記2つの設定を反映後、再計測を実施

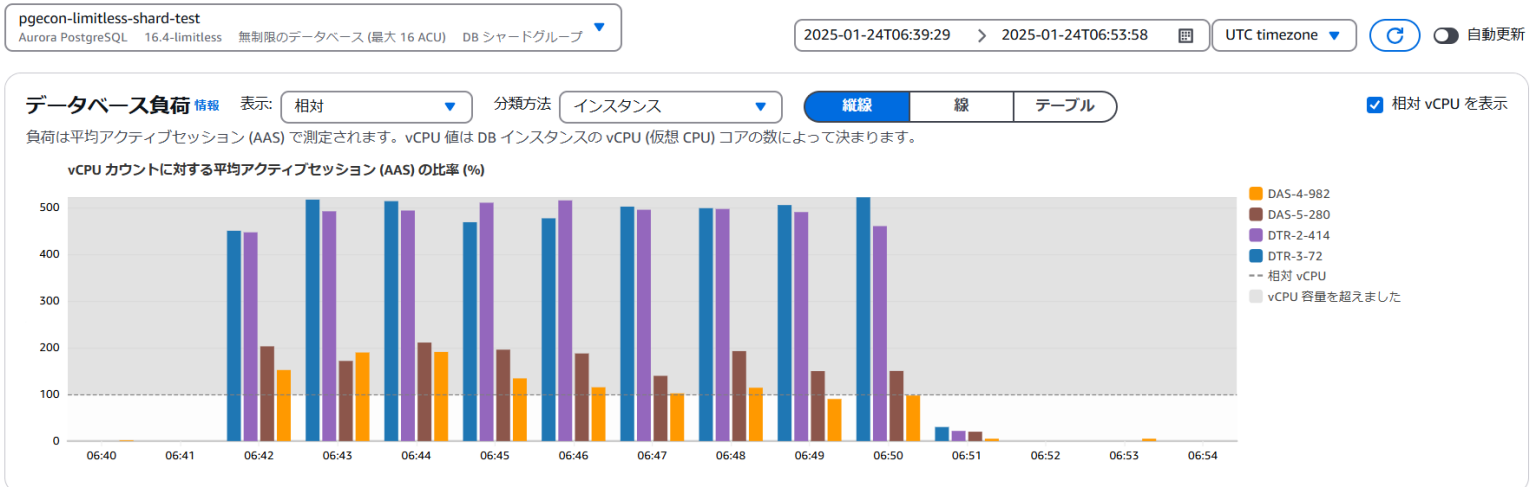
実機検証：結果考察



- 傾向
  - 偏り解消後のNOPMが、もともとのLimitless Databaseの結果より劣化
    - 劣化傾向が見られたため、VU数：8,16のみの計測で中止

実機検証：結果考察

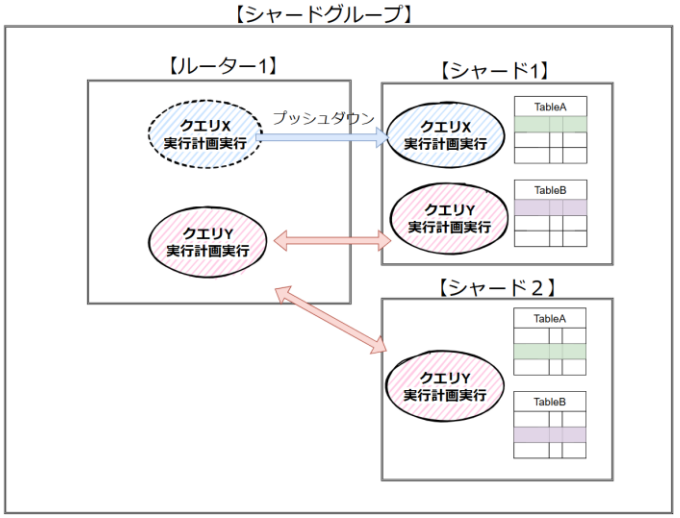
- ルータ/シャードの偏りは是正されていることは確認
  劣化の原因は”All Warehouses”オプションによってアクセスするデータが多岐にわたり、キャッシュが効かなくなったことと推測
- 依然としてProvisionedより大きく性能が悪いため、
  Limitless Databaseの性能を引き出すべく、以下を検討
  - スペックアップ
    - Provisionedと同じメモリ（32GiB RAM）相当のスペックだが、ルーター/シャードの合計4つに分散して割り当てられるため、個々のスペックが小さいことが原因と推察
  - Limitless Database特有のチューニング
    - 次スライド以降で説明



Limitless Database VU数：8のDBLoad

Limitless Database チューニング手法: SSO

- Limitless Databaseでは、クエリの解析～実行～結果抽出に至るまで、ルータ・シャード間で処理が分担されており、各々の間で情報のやり取りも数多く発生する
- ルーターでは、実行計画の算出前に実行するクエリが参照・アクセスするシャードを特定する、分析フェーズがあるが、この分析で関連するテーブル群やデータがすべて同じシャードに存在しているとわかった場合、ルーター側のクエリ計画実行がスキップされ処理の大半を対象単一シャードで分担するようになる。（プッシュダウン）
  このクエリの実行最適化を Single-Shard Optimization (SSO) とよぶ
  - 単一シャードで実行できないクエリを、分散クエリ (Distributed queries) と呼ぶ
- SSOが適用されると、ルータとシャード間のラウンドトリップ回数が減り、多くの場合パフォーマンスが向上する。そのため、SSOが適用されるための設計やチューニングが、Limitless Databaseでは非常に重要となる
  - SSOかどうかは実行計画で確認可能



- SSOを実施するための設定は不要で、テーブル設計・制約（次スライドで説明）・クエリなどを検討する必要がある
- 詳細は Aurora PostgreSQL Limitless Database のクエリ - Amazon Aurora を参照のこと

Limitless Database チューニング手法: SSO

- SSO実行の制限
  - SSOが適用されない条件はいくつかあり、注意が必要
    - 詳細は、Aurora PostgreSQL Limitless Database の単一シャードクエリ - Amazon Aurora を参照のこと
  - この条件の一つに、クエリにPL/pgSQL変数が含まれている場合 があり、HammerDBからのTPC-C実行には、いくつかこの条件に当てはまる箇所がある（下記はその一例）

```
3179            proc fn_prep_statement { lda } {
3180                set prep_neword "prepare neword (INTEGER, INTEGER, INTEGER, INTEGER, INTEGER) as select neword(\$1,\$2,\$3,\$4,\$5,0)"
3181                set prep_payment "prepare payment (INTEGER, INTEGER, INTEGER, INTEGER, INTEGER, INTEGER, NUMERIC, VARCHAR) AS select payment(\$1,\$2,\$3,'
```

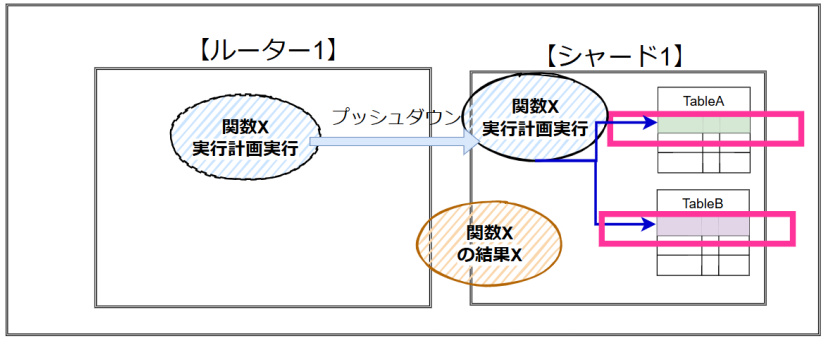
引用:https://github.com/TPC-Council/HammerDB/blob/v4.12/src/postgresql/pgoltp.tcl#L3180

  - そのため、これまでのHammerDBからの計測時にはSSOが適用されず、Limitless Databaseの性能が十分発揮できていなかったことが想定される

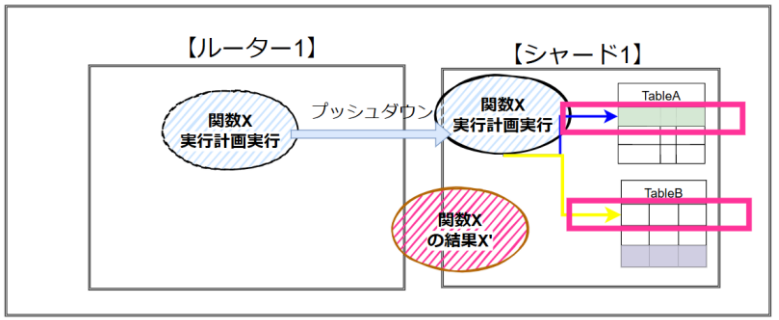
Limitless Database チューニング手法：Function distribution設定

- LimitlessにはSSO実行と同様に関数（Function）をシャードにプッシュダウンできる機能があり、これを関数の分散（Function distribution）と呼ぶ
- この利用は手動で設定することが必要（自動に設定されない）
- 設定は下記の構文にて対象の関数・分散キー・併置テーブルを指定し実行する
  - *SELECT rds_aurora.limitless_distribute_function ('function_prototype', ARRAY ['shard_key'], 'collocating_table');*
- この機能を利用すると、SSOの場合と同様の理由でパフォーマンスが改善される可能性が高い
- ただし、分散化をした関数の実行に必要なデータが対象のシャードにすべて揃っていない場合、エラーなど発生せずに結果が返され結果が期待と異なる可能性があるので利用の際はテストが必須
- 詳しくは、Aurora PostgreSQL Limitless Database の DDL 制限とその他の情報 - Amazon Aurora を参照のこと

■関数Xに必要なデータがプッシュダウンしたシャードに**すべて揃っている場合**



■関数Xに必要なデータがプッシュダウンしたシャードに**すべて揃っていない場合**



Limitless Database チューニング手法：Function distribution設定

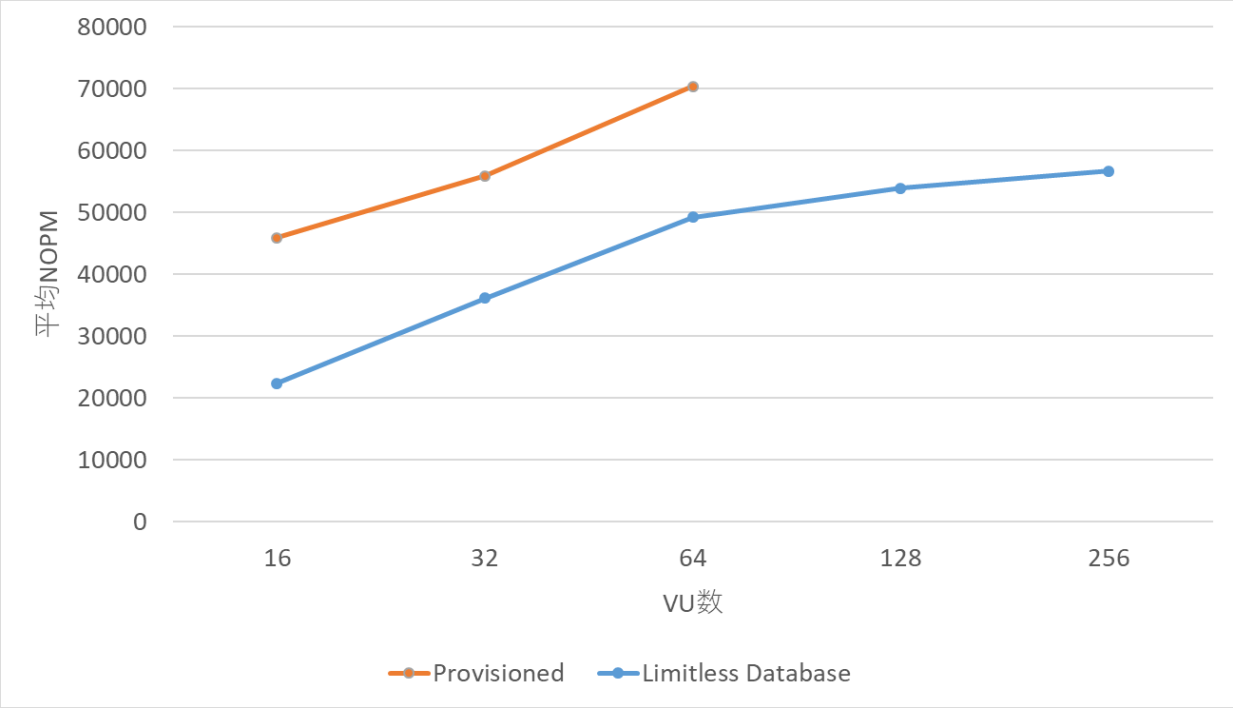
- 本検証のHammerDBのTPC-C実行においては、下記のような内容で行った
- 対象の関数
  - 計測用関数（NEWORD,DELIVERY,OSTAT,SLEV）
    - PAYMENT関数は単一シャードにデータが揃っていない可能性が高く、設定から除外
  - DBMS_RANDOM（NEWORD内で使われる）
- 対象のスクリプト
  - HammerDB/src/postgresql/pgoltp.tcl at v4.12 · TPC-Council/HammerDB · GitHub
- 関数の修正: 引数に具体的な変数名を追記する（例：NEWORD）
  - 変更前：*CREATE OR REPLACE FUNCTION NEWORD (INTEGER, INTEGER, INTEGER, INTEGER, INTEGER, INTEGER) RETURNS NUMERIC AS '*
  - 変更後：*CREATE OR REPLACE FUNCTION NEWORD (no_w_id INTEGER,no_max_w_id INTEGER,no_d_id INTEGER,no_c_id INTEGER,no_o_ol_cnt INTEGER,no_d_next_o_id INTEGER) RETURNS NUMERIC AS '*
- 関数の分散化（例：NEWORD/DBMS_RANDOM）
  - *SELECT rds_aurora.limitless_distribute_function ('neword (integer,integer,integer,integer,integer,integer) ', ARRAY ['no_w_id'], 'warehouse');*
  - *SELECT rds_aurora.limitless_distribute_function ('dbms_random (integer, integer) ');*
  - rds_aurora.limitless_distribute_function構文は第一引数のみの指定の場合、全シャードにプッシュダウンされる
    ※2025/04時点AWSユーザガイドに未記載の内容

実機検証：追加計測

- Limitless Databaseの性能をなるべく引き出すべく、
  以下の設定変更を行ったうえで再度計測を実施

- Limitless Database
  - スペックアップ
    - minACU：16 , maxACU：16→128
      - ACU：128はProvisionedの8倍のスペック相当
      - minACU :16は据え置きのため、ルーター/シャード数は変わりなし
  - 特有のチューニング
    - Paymentを除くベンチマーク実行関数のFunction Distribution設定
  - その他
    - P12. のルーター、シャードの偏り緩和は引き続き実施
    - PostgreSQLのバージョンを16.4→16.6に変更
- Provisioned
  - 負荷偏り解消のために実施した”All Warehouses”オプションを有効化

実機検証：追加計測結果



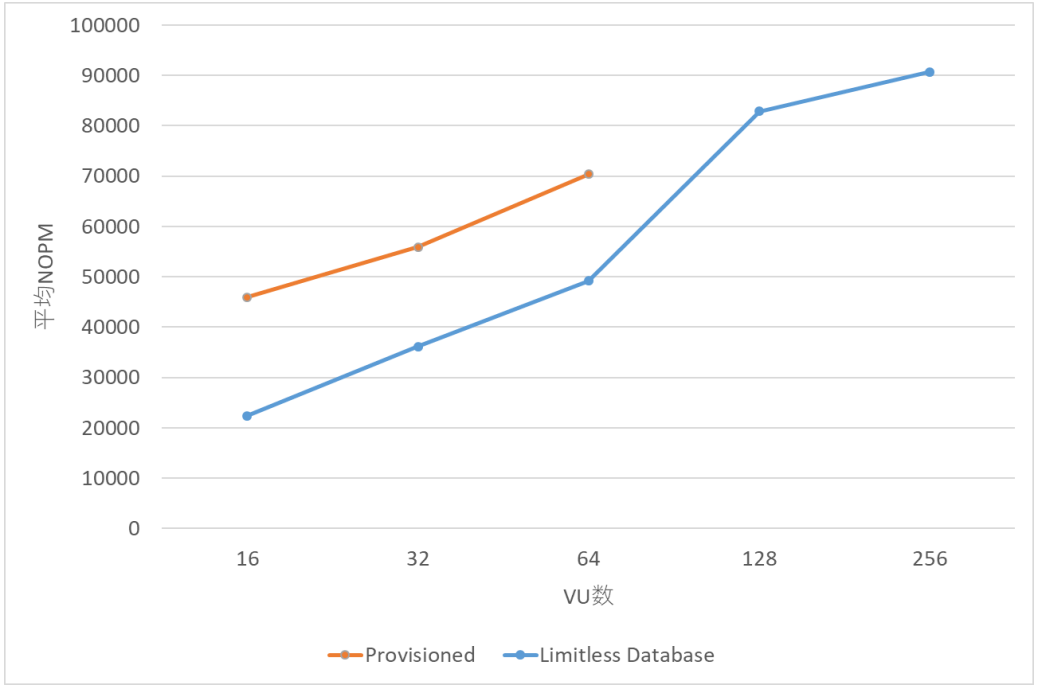
- **傾向**
  - ProvisionedはVU数:32,64でCPU使用率が100%になりVU128,256は未実施
  - Limitless Databaseのスループットは前回の計測より大きく向上したがProvisionedには及ばず
    - スペックアップ (MaxACU:16→128) が大きく寄与したと考えられる
    - VU数：128,256におけるスループットの伸びが悪い

実機検証：追加計測 結果考察

- リソースを使い切れてなくて、余裕がある状態と思われる
  - VU数：64におけるACU使用率が計測期間中最大60%程度なのに対し、VU数：128におけるACU使用率も同程度となっており、Limitless Databaseは負荷状況によって自動的にスケールアップ（ACU増加）を行うが、スケールアップの速度が足りず、計測期間中に上がりきらなかったことが原因と推測
    - Performance insightsによるDBLoadではなく、CloudwatchメトリクスであるACU使用率（ACUUtilization）を確認

- rampup時間を4min→8minに変更し、VU128,256の再計測を実施

実機検証：追加計測 結果考察



- 傾向
  - Limitless Database VU数:128,256のスループットが大きく改善
  - 高いVU数では、Provisionedに近いNOPMが出ていると想定される
    - ProvisionedはVU数64の時点でCPU使用率100%となっており、これ以上伸びにくいと思われる
    - ただし、ProvisionedとLimitless Databaseは同スペック相当ではないことに注意

実機検証 検証結果整理

- TPC-CにおいてProvisionedと同スペック相当のLimitless Databaseでは、スループットはProvisioned > Limitless Databaseとなった
  - Limitless Databaseはルーター、シャード間で通信を行う必要がありオーバヘッド大
  - 最初の計測では小さすぎるスペックで実施したため、差が大きくなった可能性も大きい

- Limitless Databaseの性能を引き出すためには特有のチューニングが必要
  - SSO (Single-shard Optimization) ,Function Distribution設定
  - SSOは手動設定の必要はないが、制約により実行されない可能性がある
    Function Distribution設定はパフォーマンスが向上する可能性があるが、
    関数の実行が正しい結果を得られなくなる可能性があるため注意が必要

- スケールアップ速度が間に合わない (スループットが上がりきらない) ケースも確認
  - 一時的に急激にアクセスが増えるシステムにおいては、ACUのレンジを小さく必要がある
    - minACUを高くすることで、平常時に置いても安定した性能となると思われる

- 今回のケースにおいてはFunction Distribution設定に加え、
  Provisionedの8倍相当のmaxACUを設定したところ、
  高いVU数においてProvisionedに近いスループットが出たと思われる

所感

- アクセス数、処理量が小さいワークロードでは、
  Limitless Databaseのメリットが発揮しにくいと考えられる
  - 小さすぎるACUだと性能がかなり劣化してしまう可能性があるため、
    ある程度大きいACU,ワークロードの環境で利用すべき
    - 最低でもルーター/シャードは2台ずつになるので、
      ノード数に応じて割り当てリソースが分割されてしまうことに注意が必要

- Limitless DatabaseはProvisionedより圧倒的に高いスペックが選択可能なため、
  アクセス数、処理量が非常に大きいワークロードでは有力な選択肢となる。

- Limitless Databaseを利用するには専用のDB設計が必須と思われる
  - テーブル設計
  - Limitless Databaseで動作させるアプリケーションがSSO,Function Distribution実行可能かどうか
    - Function Distribution設定が必要な場合は設定＋テストが必要となり、実装コスト大

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。
- 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 （2024年度 WG3）	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
			野崎　幹男
			永井　光
			磯部　凌

PGECons

PostgreSQL Enterprise Consortium