NTT DATA
Global IT Innovator

PostgreSQL10を導入！大規模データ分析事例からみるDWHとしてのPostgreSQL活用のポイント

2017/12/5

株式会社NTTデータ

はじめに

- 近年のPostgreSQLは、パラレルクエリをはじめとして、大量データに対して分析クエリを流すようなDWHとしての用途で活用できる機能が強化されています。

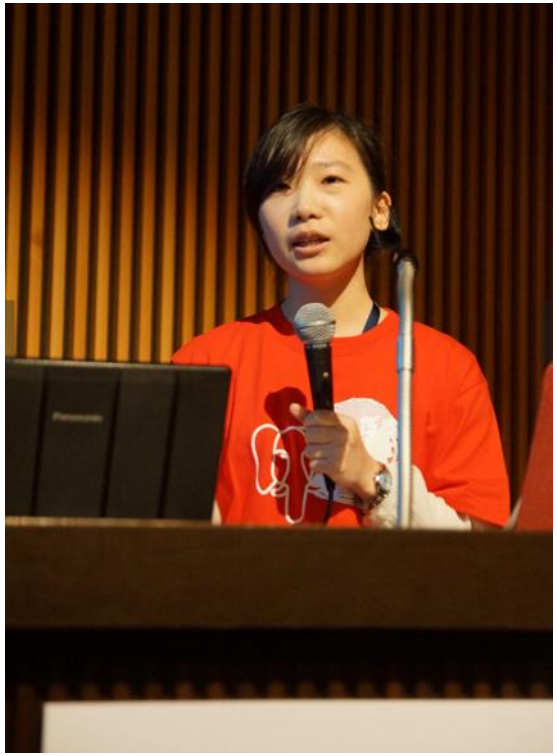
- 本講演では、DWHとしてPostgreSQLを扱うときに、PostgreSQLエンジニアが知っておくとよいポイントについてNTTデータでの事例を踏まえて紹介します。

目次

- はじめに
- システムの概要
- ミッションと課題
- 課題突破のポイント
- DWHの落とし穴
- まとめ

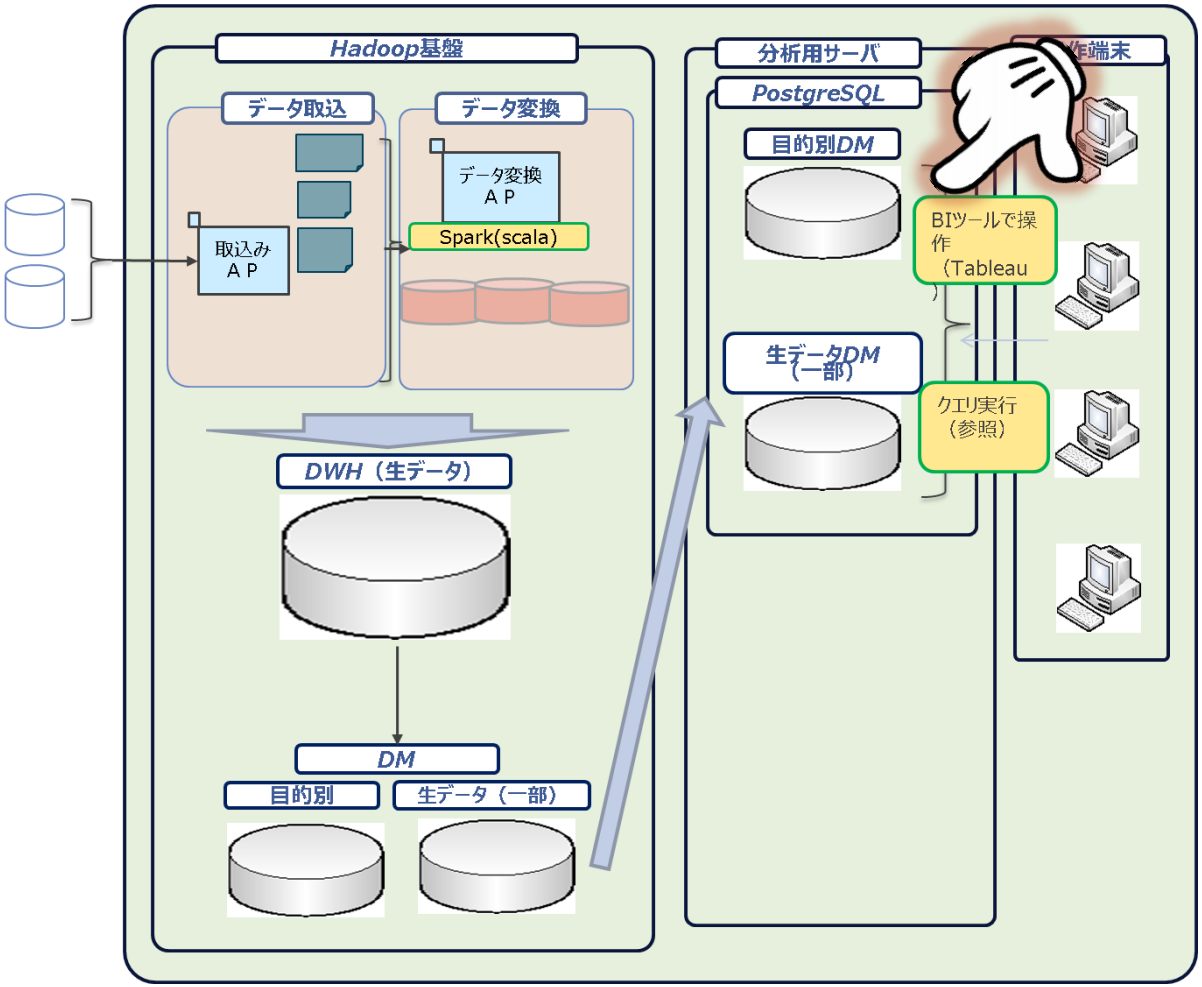
自己紹介

- 石井愛弓（いしいあゆみ）
- NTT データのPostgreSQL チームとして、社内プロジェクトへ技術支援を実施。



プロジェクト概要

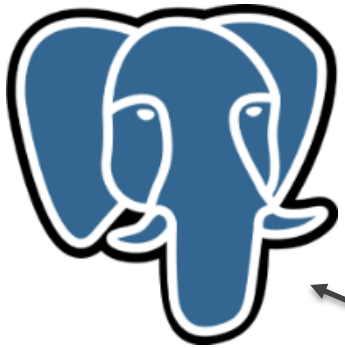
ヘルスケア業界、大規模データ分析システム
BIツールを使ってインタラクティブにデータ分析を行う基盤



NTT DATA

プロジェクト概要

- BIツールのバックエンドとしてPostgreSQLを選択
- データサイズ：約**10TB**（約2年分）、
- 最大テーブル：**20億件**の大規模データを対象に分析を行う

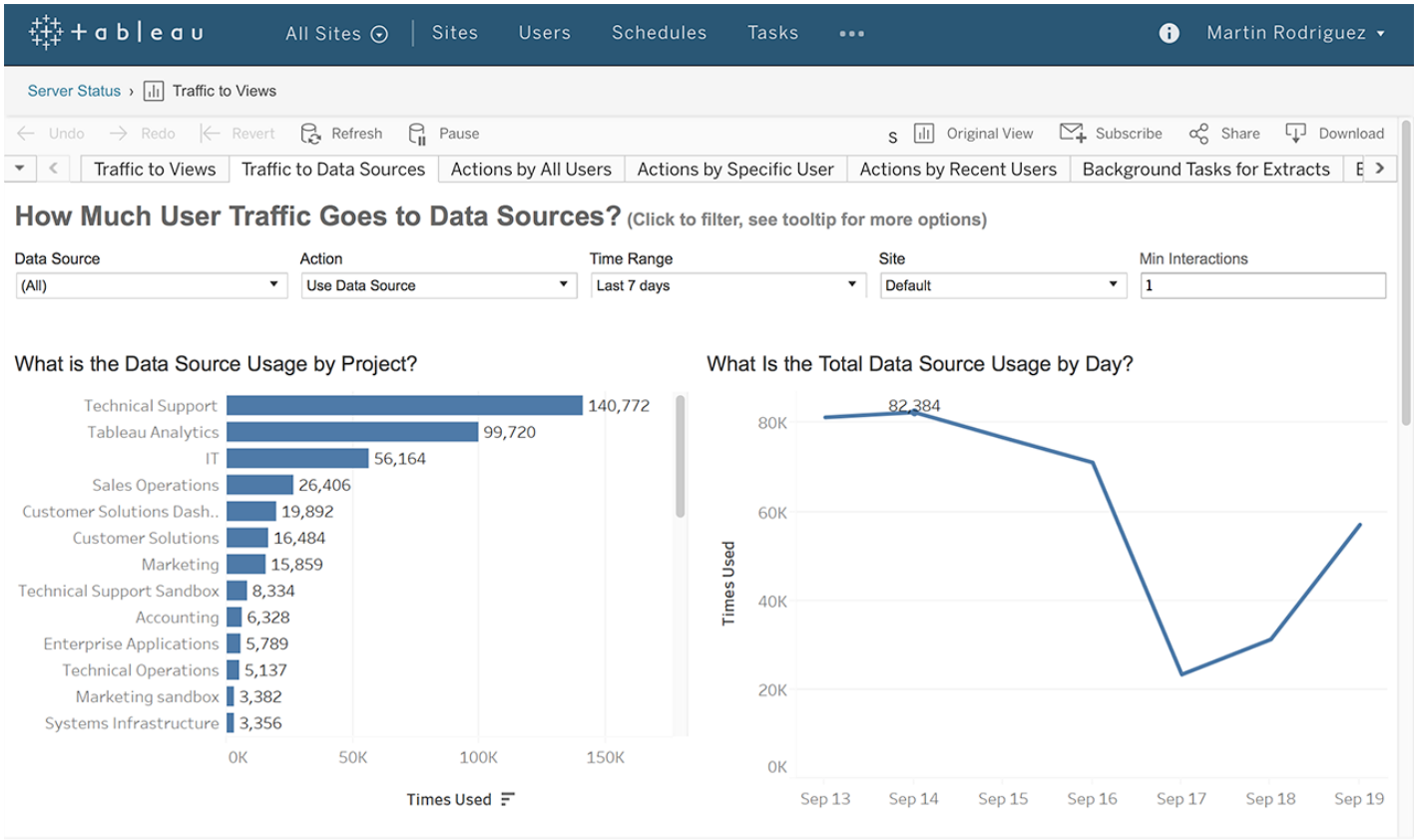


使用例1
TableauにてPostgreSQLを自由に参照し、表化、グラフ化。（更新は不可。）

使用例2
Rを用いて（もしくは直接）クエリ実行を行い、統計処理を実施し、表化、グラフ化。

Tableau

- データを直感的操作で可視化するBIツール
- PostgreSQLやその他のRDBMS、ファイルなど様々なデータソースに対応



NTT DATA

なぜ、PostgreSQLなのか

本件でデータベースに求められる要件

1. Tableau、R、psqlなど様々なフロントエンドからアクセスできるデータベースであること

2. オープンソースであること

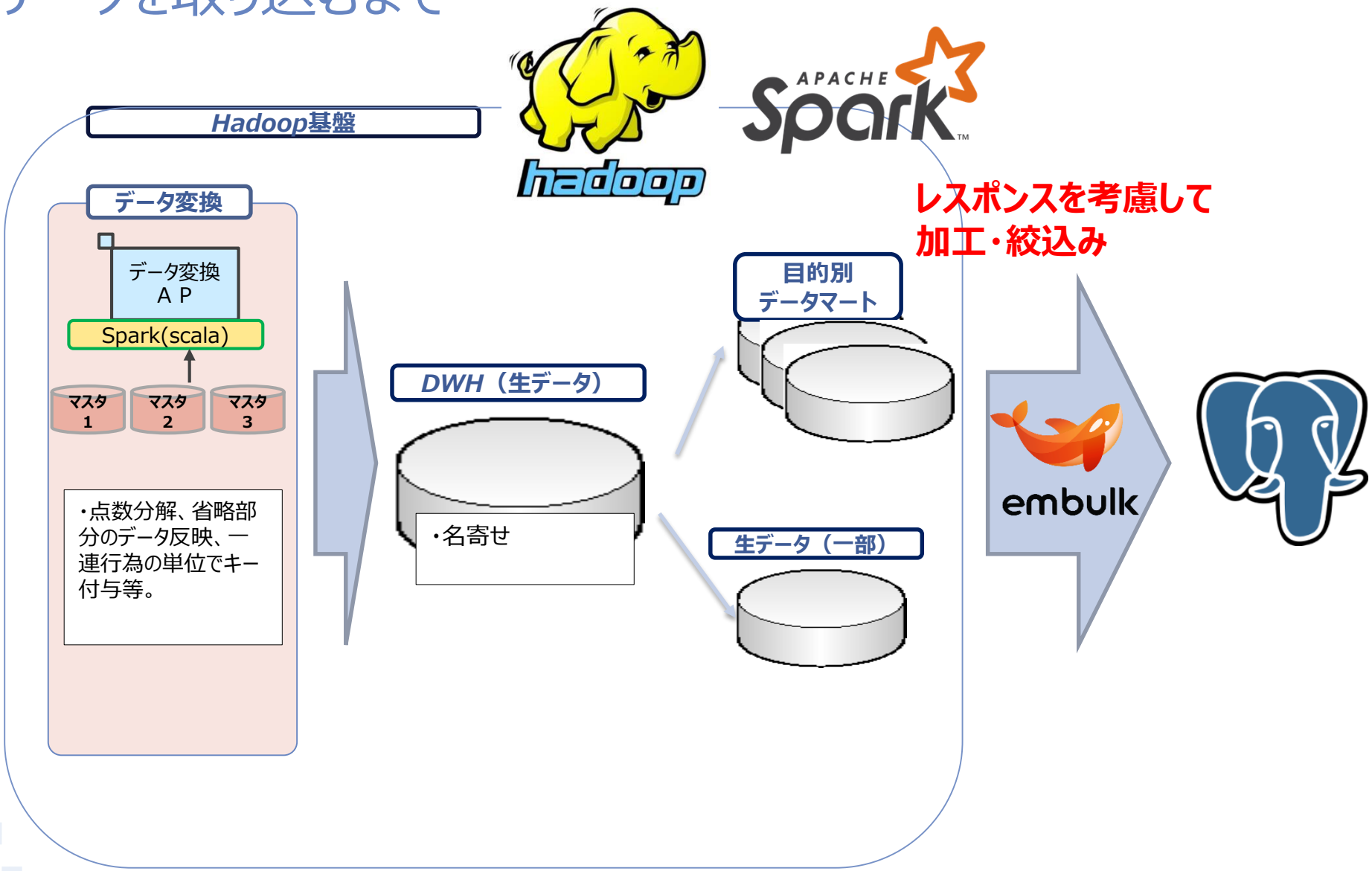
3. 大規模データセットに耐えられること

ミッション

- Tableauからのレスポンスを10秒で実現せよ

- データベースサイズは全部合わせて約10TB
- 一度のクエリで全体にアクセスすると、達成できない
  →目的別データマートを用意

データを取り込むまで


Hadoop基盤
hadoop
APACHE Spark
データ変換
データ変換
ＡＰ
Spark(scala)
マスタ 1
マスタ 2
マスタ 3
・点数分解、省略部分のデータ反映、一連行為の単位でキー付与等。
DWH（生データ）
・名寄せ
目的別
データマート
生データ（一部）
レスポンスを考慮して
加工・絞込み
embulk

DWHで高速化するための課題

- 小手先でうまくいかないのがDWH

- SQLの書き換えで高速化？
  - SQLを作るのはTableauなので、SQLチューニングは不可。
  - pg_hint_planでコメントを付け実行計画誘導もできない。

- インデックスで高速化？
  - 人の操作次第で、どんなクエリがくるか予想が難しい
  - インデックスをどこにはるかが難しい
  - とりあえず絞込みなしでGROUP BYがほとんど

→パラレルクエリの見せ所！
　そのために、まずはハードが強くないと闘えない！

ポイント(1) サイジング

目的は、「**パラレルクエリ**活用」

CPU
- パラレルクエリの**並列数×同時接続数**で使用されるコア数の確保

メモリ
[本件の前提]データサイズが大きいため、全てがメモリに載りきるのは難しい
- 256GB
  - 基本はIOで処理される前提で、IO重視

ストレージ
[本件の前提]ディスクアクセスが大量に走る
- SSD
  - IOが弱いと、パラレルクエリの恩恵が受けられない
- RAID6
  - 読み込み性能重視

ポイント(2) パラレルクエリ

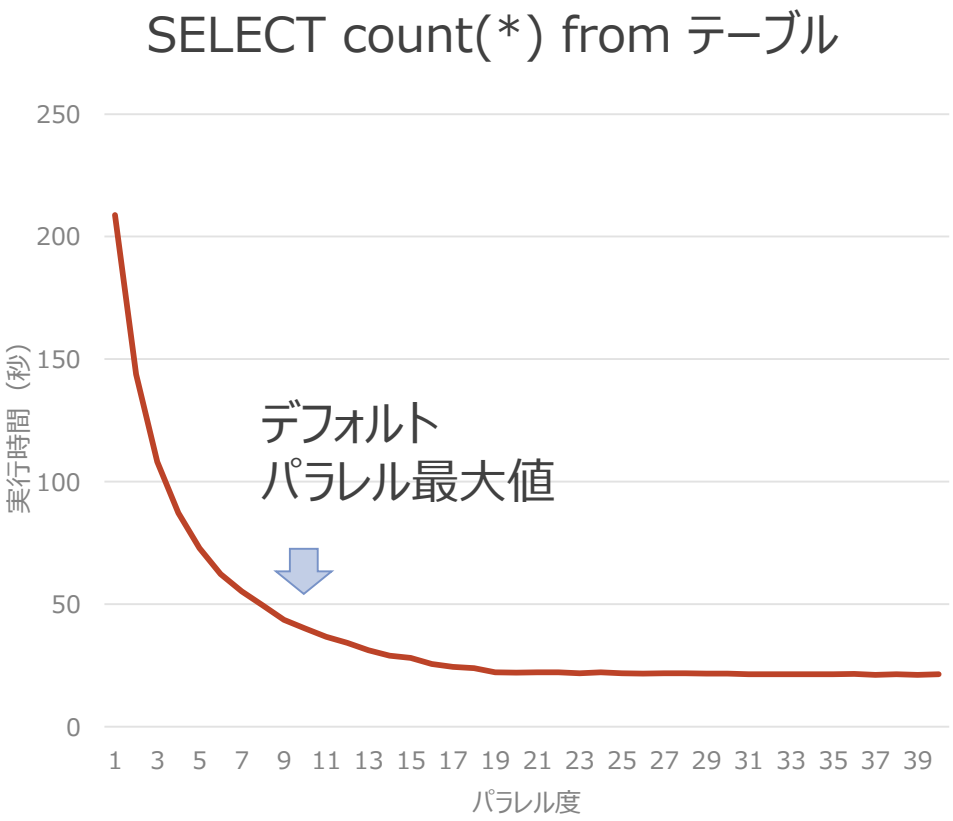
- PostgreSQL9.6で導入されたパラレルクエリ
  - 複数CPUを活用し複数プロセスが並列で処理をする
  - DWHとしての用途で性能面で効果が大きい

- 9.6では、Seq Scanなど一部のScan/Join方法のみ利用可
- 10ではIndex ScanやMerge Joinなど幅が広がった

- まだリリースされていなかった**10の採用**を前提に設計
- Beta版を使って検証
- スペアとして9.6の設計も行い共存させた

パラレル検証
並列数はどれぐらいが最適？

パラレルクエリ検証

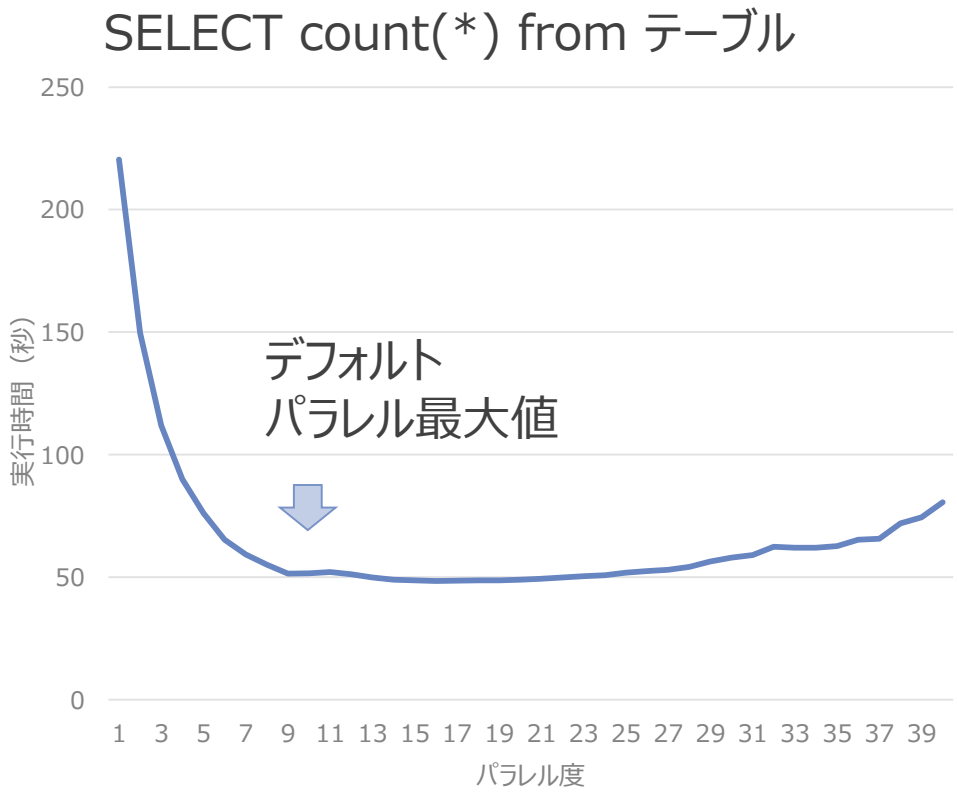


・各Parallel値毎に10回計測 /平均
・生データ分 160GB / 20億件

NTT DATA

パラレルクエリ検証

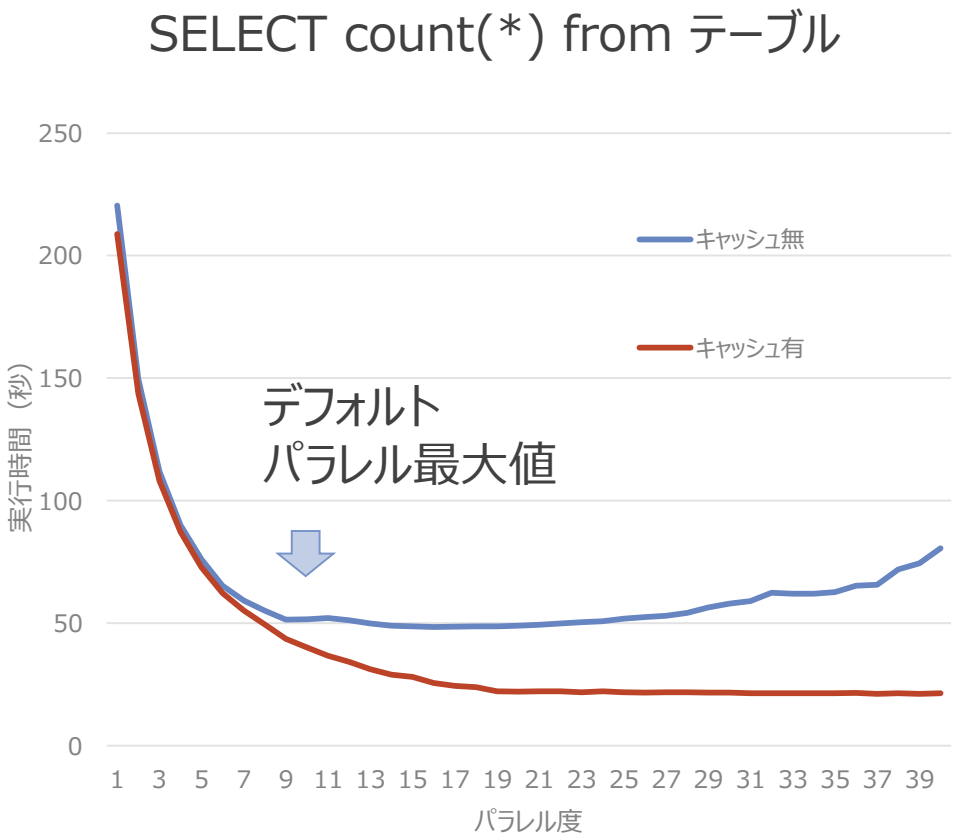
OSとPostgreSQLのキャッシュをクリアした場合



・各Parallel値毎に10回計測 /平均
・生データ分 160GB / 20億件

NTT DATA

パラレルクエリ検証



- utilはほぼ100%
- pg_stat_activityはIO/DataFileRead
- IO量は基礎性能測定に達していない

補足：PostgreSQL10のpg_stat_activity

- pg_stat_activityで以下のカラムが詳細化された
  - wait_event_type
  - wait_event

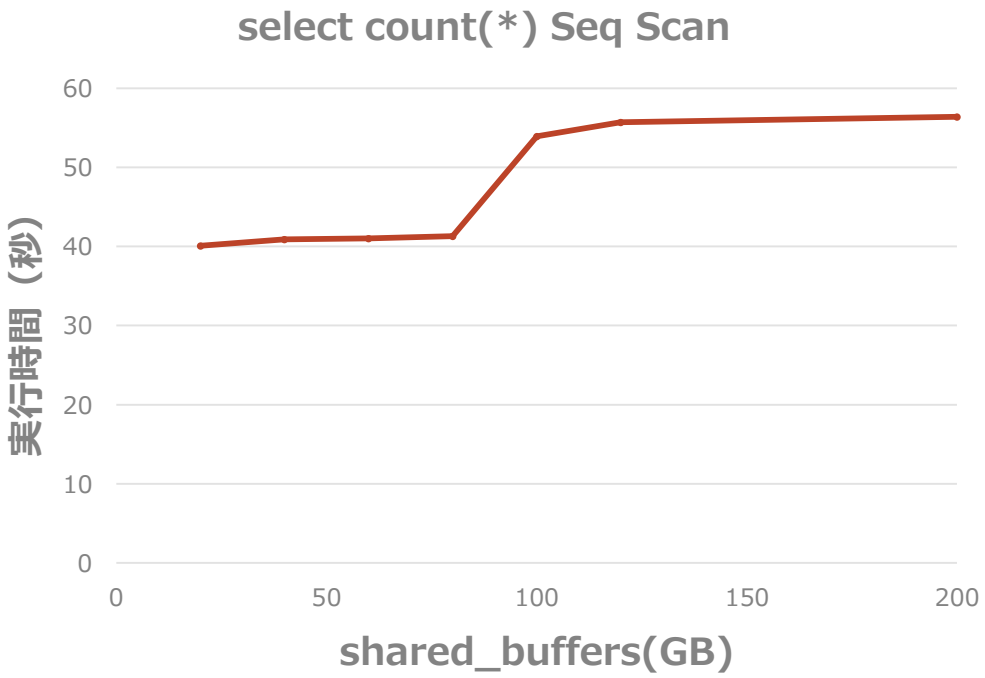
- wait_event_type
  - LWLock
  - Lock
  - BufferPin
  - Activity
  - Extension
  - Client
  - IPC
  - Timeout
  - IO

パラレル検証の結果

- キャッシュに乗っている場合、デフォルトの最大並列数よりも、並列数を上げる設定をすると性能向上
- キャッシュに乗らない場合、デフォルトの最大並列数のあたりで性能が伸びなくなった
- 並列度を増やしすぎるとオーバヘッドにより性能が悪化

ポイント(3) パラメータ

- shared_buffers検証



同じクエリを2回実行した結果
（キャッシュ有）

shared_buffers	OSキャッシュサイズ(想定)
20	236
40	216
60	196
80	176
100	156
120	136
140	116
160	96
180	76
200	56

サーバ搭載メモリ：　256GB
データ量：　160GB
データ件数：　20億件

なぜshared_buffersを大きくしてもだめ？

- PostgreSQLは、取得結果が大きく、共有バッファの大部分を占めてしまいそうな場合、shared_buffersを全て使わず、リングバッファ内にとどめる
- 逆に、shared_buffersを小さくして、OSキャッシュを大きくしたほうが、結果がキャッシュに載りやすくなる

ポイント(3) パラメータ

- SSDの場合、ランダムスキャンが強い
- random_page_costはseq_page_cost（1）に近づけるべし

- デフォルトの4で計算するとインデックススキャンのコストが過大に評価され、インデックススキャンのほうが速くても選ばれにくい

ポイント(4) インデックス

- pg_stat_statementで頻出するクエリなどを調査
- よく使われるカラムに対して、インデックスの付与を検討

- BRIN(Block Range Index)に注目
- PostgreSQL9.5から追加された

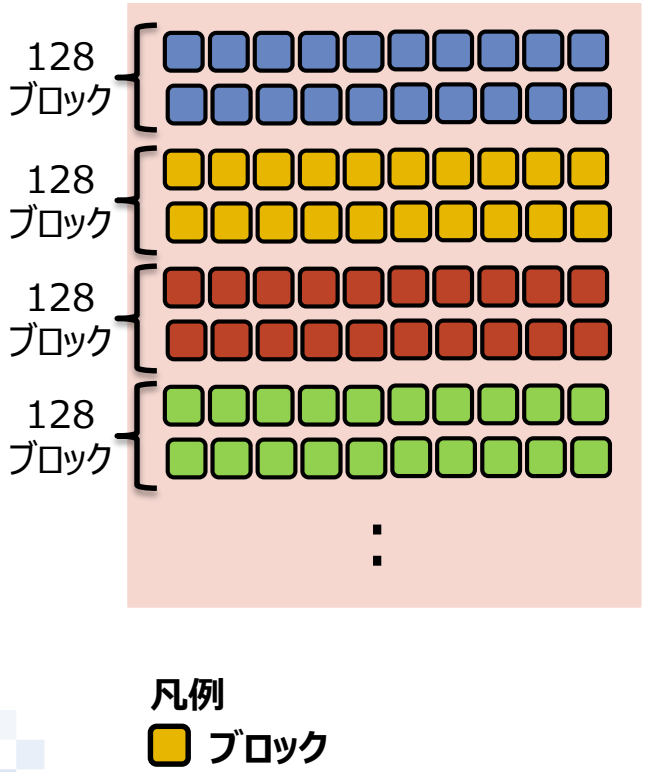
メリット
- 大規模テーブルに対して、範囲検索を行う場合高速
  - 特に、キー値の値が物理的に連続している場合（連番など）
- インデックスサイズが小さい
- インデックス作成時間が短い

→DWH用途に向いている

Block Range Index

インデックス作成

CREATE INDEX hoge_brin ON hoge USING **brin**(col);



近接している
ブロックの束に対して
列の**最小値/最大値**
をインデックスに記録

BRINインデックス

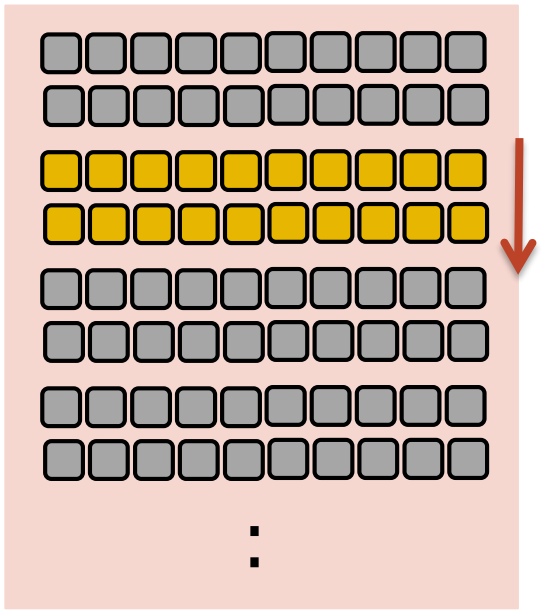
Block	Range(min/max)
1 - 128	1 ~ 1000
129 - 256	1001 ~ 2000
:	:

NTT DATA

Block Range Index

検索
SELECT * FROM hoge
WHERE col **BETWEEN 1500 AND 1700**;

テーブル



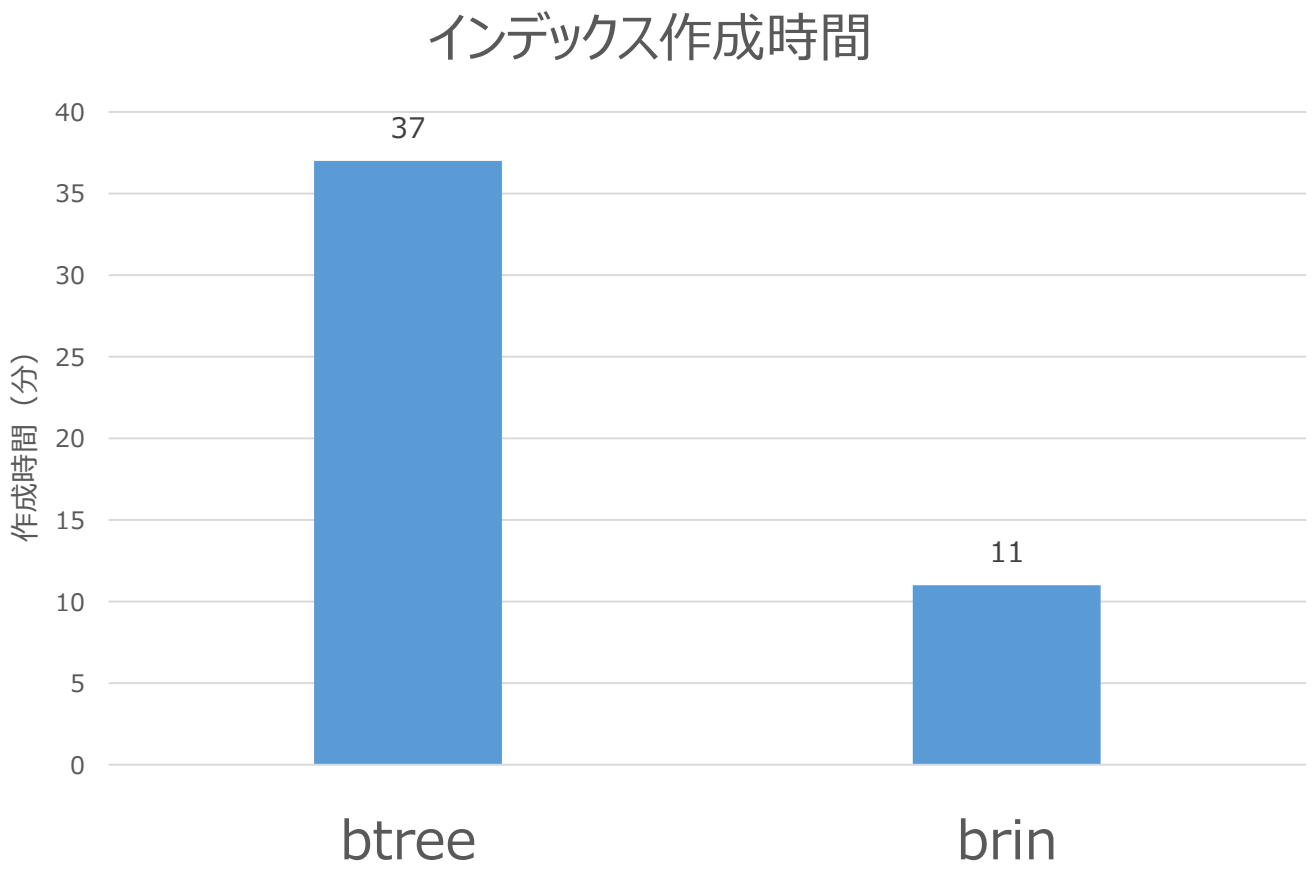
検索する範囲を
絞り高速に検索

BRINインデックス

Block	Range(min/max)
1 - 128	1 ~ 1000
129 - 256	1001 ~ 2000
:	:

凡例
ブロック

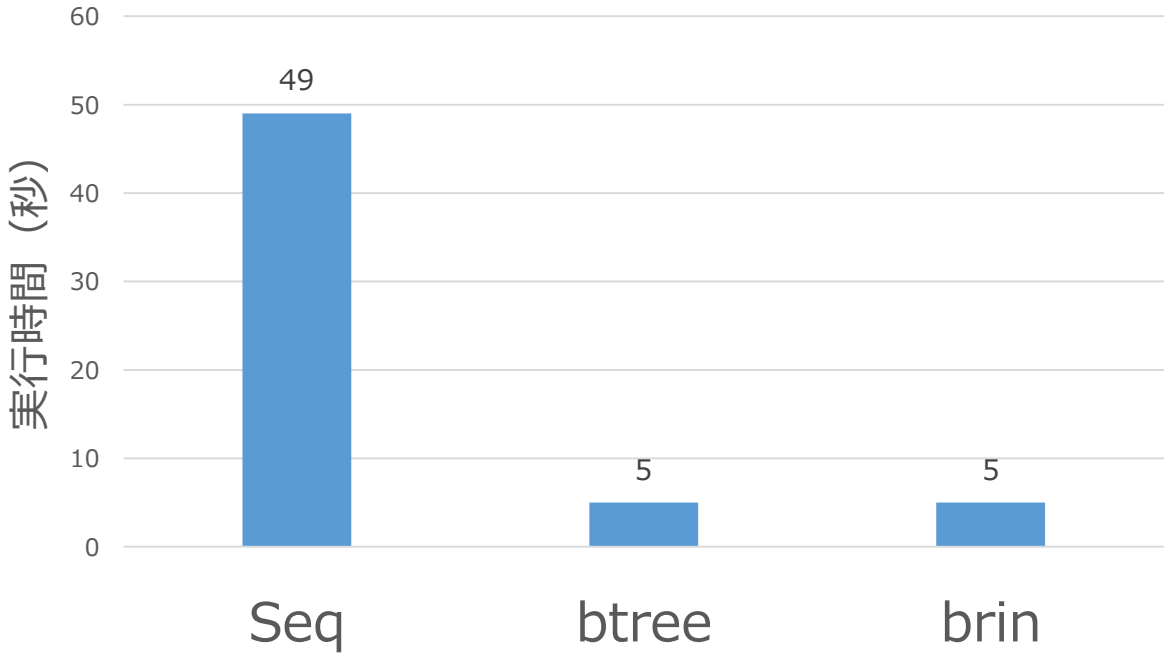
インデックス検証



インデックス検証

select * from table between xx < col1 AND col2 < yy;

※全体の30%程度を取得



- 処理時間は、データ分布（物理的に連続しているか）と取得件数の影響大
- PostgreSQLのプランナはseqscanを選んだ（random_page_cost = 1にもかかわらず）

対処法として1つのアイデア

- pg_hint_planの「テーブルでの指定」を使う
- コメントを使った方法と異なり、クエリを書き換えられなくても、「hint_plan.hints」テーブルに、**実行計画を制御したいクエリと<u>ヒント</u>**を登録しておくと、ヒントが効く

```
=# explain analyze select * from test where id = 1;
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------
Index Only Scan using a on test  (cost=0.29..8.30 rows=1 width=4) …
Index Cond: (id = 1)
   Heap Fetches: 1
 Planning time: 0.054 ms
 Execution time: 0.027 ms
(5 行)

=# set pg_hint_plan.enable_hint_table to on;
SET
=# explain analyze select * from test where id = 1;
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------
Seq Scan on test  (cost=0.00..170.00 rows=1 width=4) …
Filter: (id = 1)
   Rows Removed by Filter: 9999
 Planning time: 0.031 ms
 Execution time: 0.808 ms
(5 行)
```

DWHの落とし穴

パラレルクエリが効かない!?

- PostgreSQLにpsqlで接続し、直接クエリを実行するとパラレルになる
- がTableau経由でクエリを実行すると**パラレルにならない**
- クエリによっては、操作後、表示まで10分くらいかかってしまう。

→log_statement=allにしてサーバログを確認！

パラレルクエリが効かない!?(1)

2017-05-12 17:04:47 JST LOG: statement: BEGIN;**declare** “SQL_CUR0000000006524770” **cursor** for select c.relname, i.indkey, i.indisunique, i.indisclustered, a.amname, c.relhasrules, n.nspname, c.oid, d.relhasoids, i.indoption from pg_catalog.pg_index i, pg_catalog.pg_class c, …（略）

- TableauのPostgreSQL接続では、デフォルトでカーソルが定義される
  - クライアントに必要なメモリを抑えるため
- **PostgreSQLの仕様上、カーソルが使われると、パラレルにならない**

TableauのODBC接続で、DECLARE CURSORを使わないように設定

パラレルクエリが効かない!?(2)

- カーソルは使わなくなったが、まだ使えない
- ログを眺めていると。。

パラレルクエリが効かない!?(2)

- カーソルは使わなくなったが、まだ使えない
- ログを眺めていると。。

```
SET SESSION CHARACTERISTICS  AS ISOLATION LEVEL
SERIALIZABLE;
```

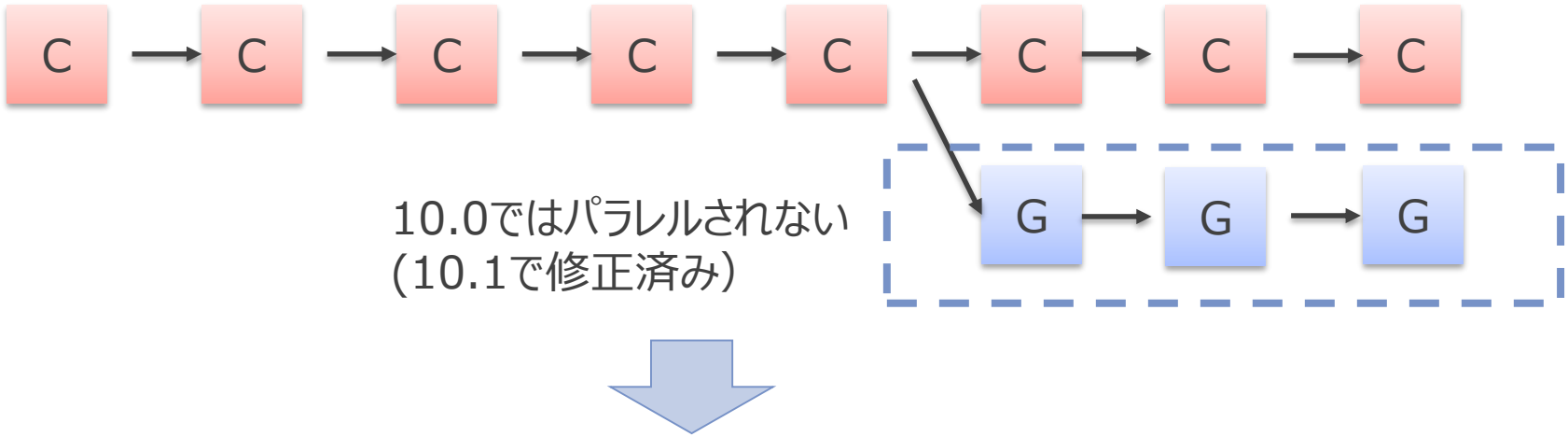
- tableauのデフォルトでシリアライザブルを設定。
- **PostgreSQLの仕様上、シリアライザブルの場合、パラレルにならない**

Read committedに変更。
本件では、読み取りのみであるので、挙動に差異はない。

- DWHではシリアライザブルがスタンダード？
  - Teradata, Netezza, redshiftもデフォルトシリアライザブル。

パラレルクエリが効かない!?(3)

- 10.0では、Prepared Statementを使っていると、パラレルされない場合がある。
  - Custom plan …バインド変数を**考慮した**実行計画
  - Generic plan …バインド変数を**考慮しない**実行計画



TableauのODBC接続パラメータでPreparedを使わないように設定。
Prepared Statement利用によるプランニング時間の短縮はできなくなる。

補足：generic planであるかどうか？の確認

- custom plan

```
Finalize Aggregate
  ->  Gather
        Workers Planned: 4
        ->  Partial Aggregate
              ->  Parallel Seq Scan on tenk1
                    Filter: (hundred > 1)
```

- generic plan

```
Aggregate
  ->  Seq Scan on tenk1
        Filter: (hundred > $1)
```

パラレルクエリが効かない!?(4)

- EXPLANではパラレルプランになったが、EXPLAIN ANALYZEではパラレルにならない

例）max_worker_processes/max_parallel_workersによる上限でワーカーが作成できない

```
Finalize Aggregate  (cost=75498.35..75498.36 rows=1 width=8) (actual time=2778.768..2778.768 rows=1
loops=1)
   ->  Gather  (cost=75497.93..75498.34 rows=4 width=8) (actual time=2778.761..2778.762 rows=1 loops=1)
         Workers Planned: 4
         Workers Launched: 0
         ->  Partial Aggregate  (cost=75497.93..75497.94 rows=1 width=8) (actual time=2778.595..2778.596
rows=1 loops=1)
               ->  Parallel Seq Scan on test  (cost=0.00..69247.94 rows=2499994 width=0) (actual
time=0.015..1598.895 rows=10000000 loops=1)
 Planning time: 0.100 ms
 Execution time: 2778.937 ms
(8 rows)
```

パラレル度が増えない!?

- パラレルプランになったが、パラレル度が増えない
- そもそも、パラレル度はどのように決まるのか？

(1)PostgreSQLがコストとプロセス数上限を考慮

- parallel_setup_cost
- parallel_tuple_cost
- max_parallel_workers_per_gather
- max_worker_processes

(2)PostgreSQLがテーブルサイズで上限を計算

※min_parallel_table_scan_size=8MBの場合

テーブルサイズ	8MB	24MB	72MB	216MB	648MB	1.8GB
パラレル度上限	1	2	3	4	5	6

(3)ユーザがテーブルごとのパラメータ設定により上限を調整

- parallel_workers
- ALTER TABLEでparallel workersを設定すれば、テーブルサイズから計算されるパラレル数上限を突破できる。

パラレルクエリが効かなかったら

- まずはパラメータを確認
  - max_parallel_workers_per_gather
  - dynamic_shared_memory_type
  - max_worker_processes
  - parallel_workers
- コストをさげてみる
  - parallel_setup_cost = 0
  - parallel_tuple_cost = 0
- パラレルにならない条件を確認
  - カーソルを使っていないか？
  - シリアライザブルになっていないか？
  - Preapared Statementのgeneric planになっていないか？(10.0のみ)
  - その他PostgreSQLのマニュアルを確認

まとめ

- **PostgreSQL10、新しい時代の幕開け**
- DWH用途としてのPostgreSQLには、課題も残っているが、十分闘える
- 特にパラレルクエリの貢献は大きい

- BIツールのバックエンドとして、PostgreSQLがスタンダードになるかもしれない
- BIツールのカスタマイズに頼らなくてもパラレルクエリが使えるように、今後、制約が少なくなることに期待

NTT DATA

Global IT Innovator