PGECons
PostgreSQL Enterprise Consortium

2020年度活動成果報告

PostgreSQLエンタープライズ・コンソーシアム
WG1（新技術検証WG）

パラレルクエリ検証
成果紹介

検証の目的

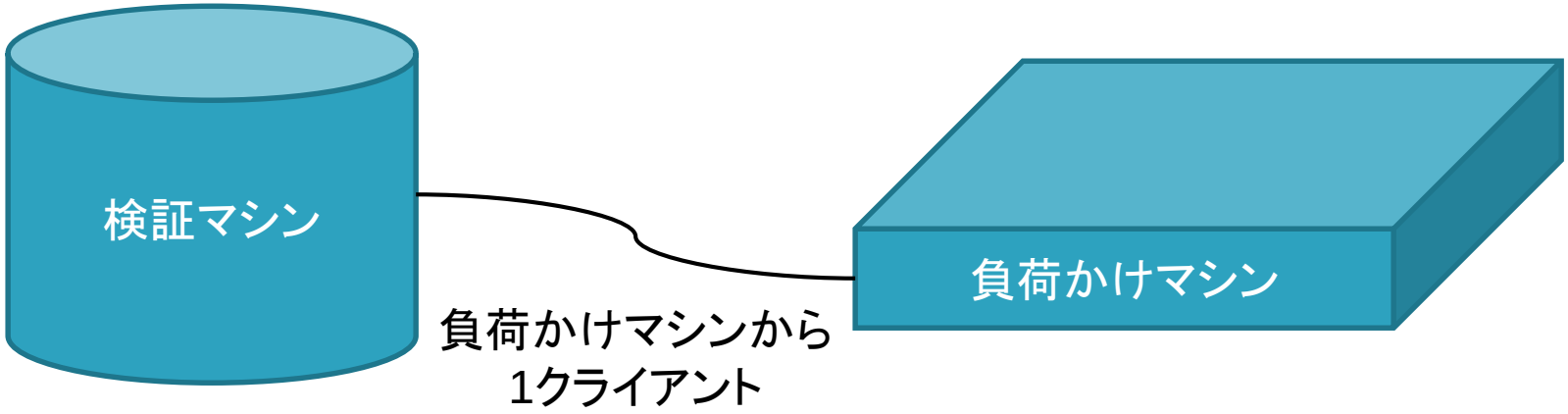
- PostgreSQLでのパラレルクエリとは
  - 1つのクエリを複数のプロセスで分担して並列処理すること
    - パラレルクエリで効率的に処理可能と判断した場合のみ採用
  - BIなどのOLAP用途で恩恵を受けやすい
  - PostgreSQL 9.6で採用され、10以降で継続的に強化されている
- PostgreSQLがOLAP用途で実用的に使用可能となったのかを検証するため、3種類の検証を試行
  - パラレルクエリ関連のパラメータのうち並列度を設定するパラメータを変更、かつインデックスを設定し、パラレルインデックススキャンにおける処理性能の改善状況と並列度の違いによる処理時間を比較
  - パラレルクエリ関連のパラメータのうち並列度を設定するパラメータを変更し、Windows環境における処理性能の改善状況を比較
  - テーブルのパーティショニングを行い、パーティショニングとパラレルクエリのそれぞれの処理時間、および併用時の処理時間の比較

検証方法

- 検証方法
  - すべての検証において、Star Schema Benchmark (SSB) に定義されている クエリ 4パターン/13本のクエリを使用する
  - 本検証のために作成したクライアントプログラムをクライアント上で実行し、DBサーバに対してクエリを実行する
  - クエリ実行前にpg_prewarmで全テーブルデータをメモリにロードする
  - Scale Factorは 100 とする
    - Scale Factor=1で概ね1GBのデータサイズが生成される
    - テーブルに格納すると、60GB程度の物理データサイズとなる
  - 上記以外の個別の検証条件については、各検証結果紹介の際に別途説明する

検証条件

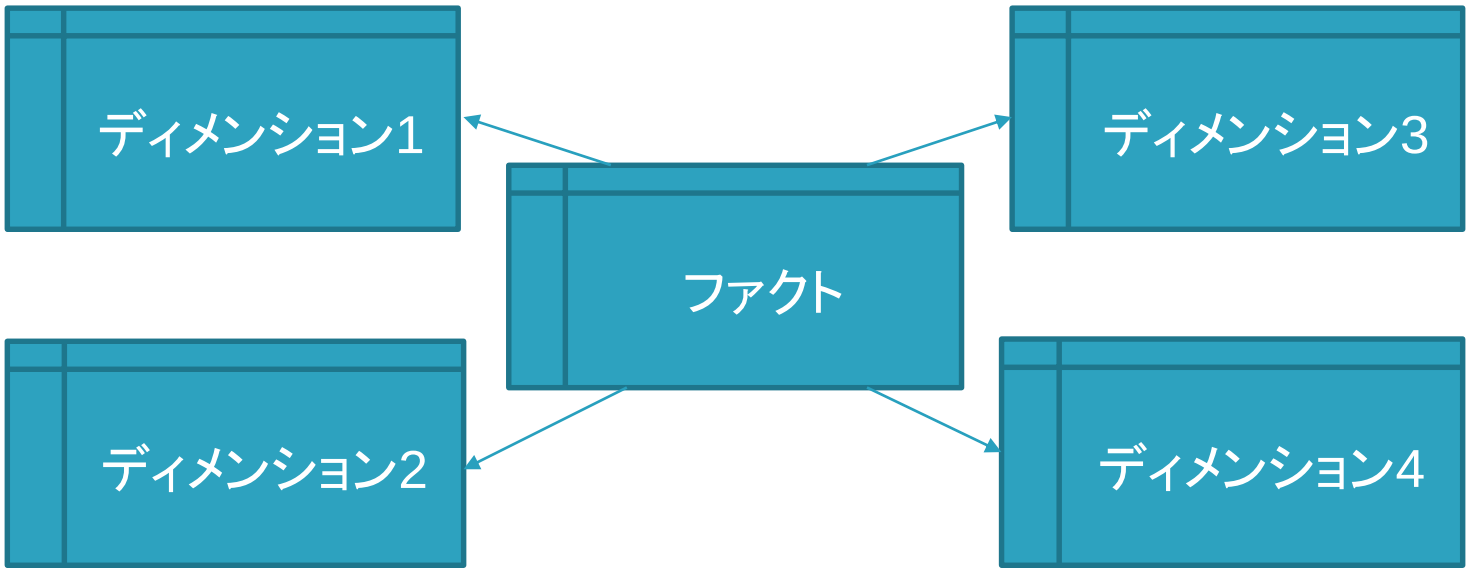
- 検証条件
  - 同一の負荷かけマシンから、1クライアントを起動して検証マシンに接続し、クエリを発行し応答時間を計測
  - 各クエリについて3回、もしくは5回計測を行い、中央値を採用
  - PostgreSQLのメジャーバージョンは13を採用
  - 使用したマシンの仕様については、各検証結果の紹介の際に別途説明する



スター・スキーマ・ベンチマーク とは？

スタースキーマベンチマーク（SSB）とは？

- スタースキーマベンチマーク（SSB）のベースはTPC-H
  - ファクトテーブル：分析対象となる実績値が格納
  - ディメンションテーブル：ビジネス上の軸になる詳細なマスターデータが格納



- テーブルサイズ
  - Scale Factor=100の時、全体で約60GB（テーブル物理サイズ）
    - 大部分がファクトテーブルに格納

スタースキーマベンチマーク（SSB）とは？

- SSBでのクエリは4パターン、計13本が定義されている。
  - q1（3本）
    ファクトテーブルとディメンションテーブル1つとの結合
    ファクトテーブルの選択率は0.008%（q1.3）～1.95%（q1.1）
  - q2（3本）
    ファクトテーブルとディメンションテーブル3つとの結合
    ファクトテーブルの選択率は0.02%（q2.3）～0.8%（q2.1）
    集約、ソートあり
  - q3（4本）
    ファクトテーブルとディメンションテーブル3つとの結合
    ファクトテーブルの選択率は0.0006%（q3.4）～3.4%（q3.1）
    集約、ソートあり
  - q4（3本）
    ファクトテーブルとディメンションテーブル4つとの結合
    ファクトテーブルの選択率は0.0009%（q4.3）～1.6%（q4.1）
    集約、ソートあり

パラレルクエリ／パラレルインデックススキャン性能検証

検証概要

- 目的
  - PostgreSQL のパラレルインデックススキャン性能について確認する
  - インデックスを利用したアクセス（実行計画）となるクエリについて、Worker 起動数を変化させることにより下記を確認することを目的とした
    1. Worker 数に応じて性能向上すること
    2. Worker 起動数の変動要素

検証方法

- 測定前提
  - 測定に際し、SQL の実行前に pg_prewarm を利用してディスクからデータを一度読み込み、PostgreSQL のバッファ上に常駐させた状態とした
  - また、VACUUM ANALYZE により不要領域を回収して Visibility Map を all-visible に設定するとともに、統計情報を更新した状態とした

- Worker 数の調整
  - max_worker_processes を想定起動 Worker 数以上の「16」で固定し、max_parallel_workers, max_parallel_workers_per_gather を変化させて Worker 起動数を変動させ、クエリ実行性能を確認
  - max_parallel_workers, max_parallel_workers_per_gather を「0」に設定した場合パラレルクエリ未使用となる

検証内容

- Star Schema Benchmark データに Index を作成
  - Scale Factor = 100GB で作成したデータセットに対し、ファクト表とディメンション表の結合列（外部キー）となっている列に B-Tree 索引を作成

- 検証①：Count クエリ
  - Worker 数の変化によるパラレルインデックススキャン性能向上を確認するため、単純なクエリとして Star Schema Benchmark のファクト表である lineorder の件数カウント時間を計測
    - クエリ：SELECT COUNT(*) FROM LINEORDER

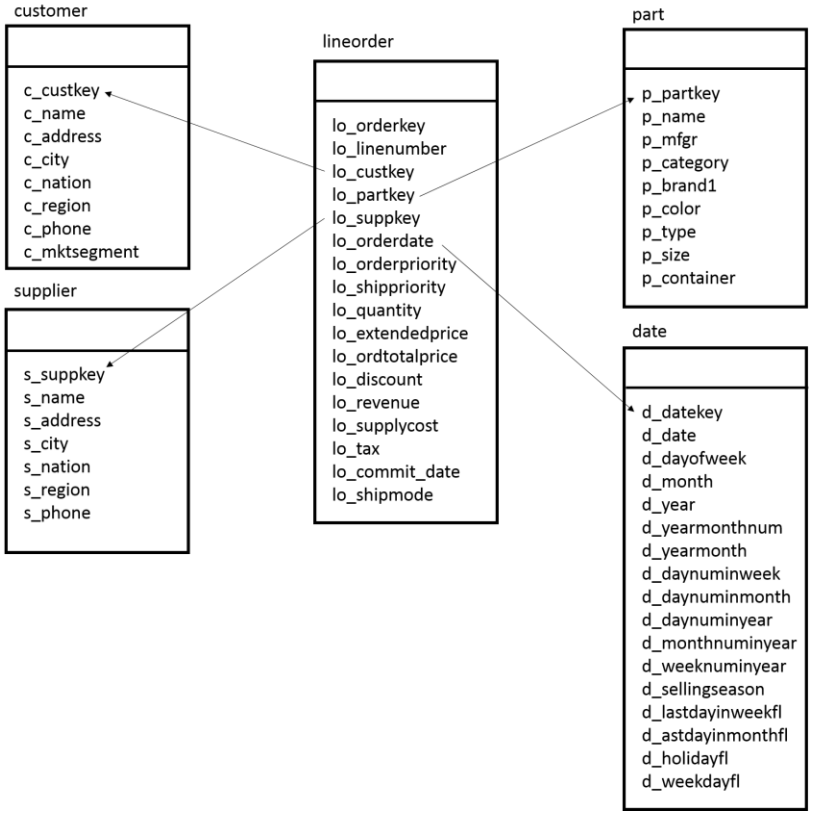
- 検証②：Star Schema Benchmark クエリ
  - パラレルインデックススキャンの応用として、実トランザクションでの性能向上度を確認
    - Star Schema Benchmark に定義されている13本のクエリを利用

参考：Star Schema Benchmark (SSB)

- 概要
  - Star Schema Benchmarkの論文によって公表されている、TPC-Hをもとに BIで用いられるものを模したファクトテーブル、ディメンションテーブル、クエリが設計されているベンチマークツール
  - 大規模なデータを取り扱うので、I/O周りはもちろんのこと、多数のジョイン操作や集約演算が行われることから、通常のOLTPよりもCPUの処理性能と、CPUをどの程度うまく使えているかがベンチマーク結果に影響

参考：Star Schema Benchmark (SSB)

- **データモデル**
  - スタースキーマ型のデータモデルを採用しており、
    1つのファクトテーブルと複数のディメンションテーブルで構成



- ファクトテーブル：
  分析対象となる実績値が
  格納される明細データで、
  データ件数は多い
  - lineorder表

- ディメンションテーブル：
  ビジネス上の分析の軸になる
  マスターデータであり、
  データ件数は少ない
  - customer表/supplier表/
    part表/date表

検証環境

ハードウェア

機器	CPU	コア数	メモリ (GiB)	ストレージサイズ (GiB)
NEC Express 5800	Xeon E5-2630 v4 2.20GHz	20 (10コア×2)	256	SSD 400GB SATA x 8 (RAID 5)

ソフトウェア

種類	ソフトウェア名およびバージョン
OS	Red Hat Enterprise Linux 8.3
Database	PostgreSQL 13.1

検証環境

- SSB検証データ(SF=100)

テーブル名/インデックス名	行数(理論値)	行数(実測値)	データサイズ(実測値)[Bytes]	データサイズ(実測値)[GB]
lineorder	SF × 6000000	600,044,480	63,557,664,768	59.19
idx_lo_pkey			13,477,847,040	12.55
idx_lo_custkey			4,266,074,112	3.97
idx_lo_suppkey			4,239,040,512	3.95
idx_lo_partkey			4,238,647,296	3.95
idx_lo_orderdate			4,159,217,664	3.87
customer	SF × 30000	2,999,825	374,030,336	0.35
idx_c_custkey			67,403,776	0.063
supplier	SF × 10000	1,000,000	114,376,704	0.11
idx_s_suppkey			22,487,040	0.021
part	200000 × (1+log2(SF)) (小数点切り捨て)	1,400,000	164,265,984	0.153
idx_p_partkey			31,473,664	0.029
date	2556	2,556	311,296	0.00029
idx_d_datekey			73,728	0.000069

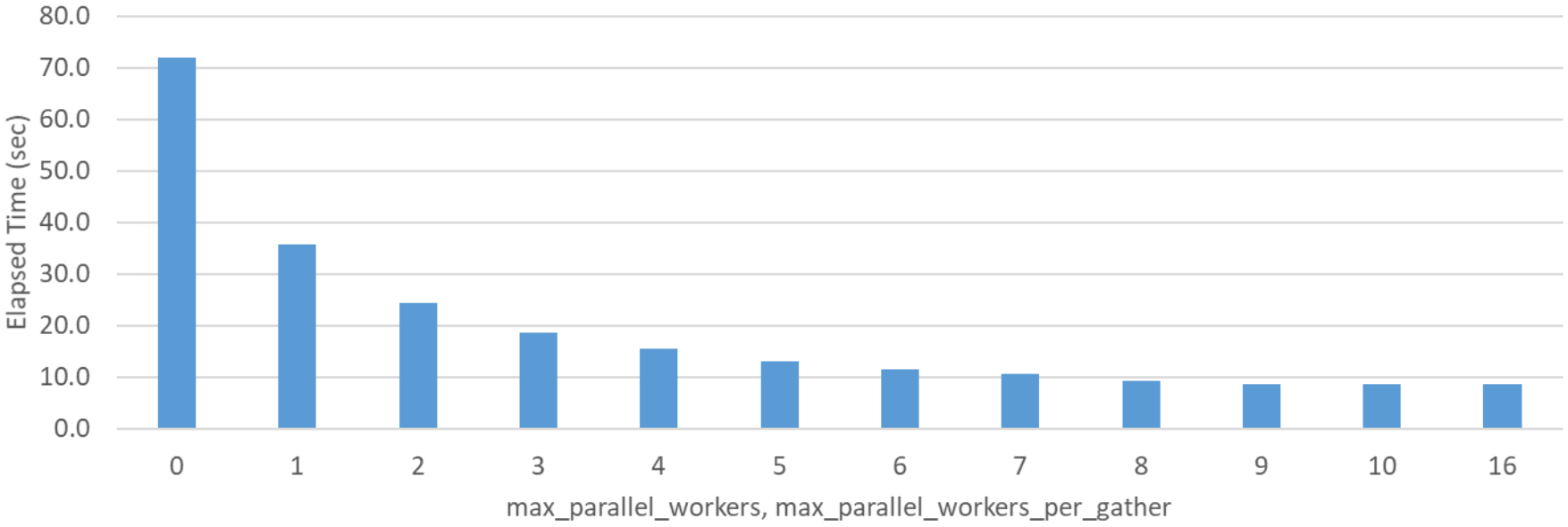
検証環境

- PostgreSQLの設定値
  - 性能測定にあたりデフォルトから変更したパラメータ

項目名	デフォルト値	設定値	備考
max_connections	100	20	
shared_buffers	128MB	100GB	
effective_cache_size	4GB	100GB	
work_mem	4MB	104857kB	
maintenance_work_mem	64MB	2GB	
wal_buffers	-1	16MB	
max_wal_size	1GB	16GB	
min_wal_size	80MB	4GB	
effective_io_concurrency	1	200	
checkpoint_completion_target	0.5	0.9	
random_page_cost	4.0	1.1	
default_statistics_target	100	500	
max_worker_processes	8	16	
max_parallel_workers	8	16	測定により値を変更して実施
max_parallel_workers_per_gather	2	16	測定により値を変更して実施
max_parallel_maintenance_workers	2	4	

検証結果 – 検証①：Count クエリ

- 性能測定結果
  - Star Schema Benchmark の lineorder 表のカウント結果で、Worker 数の増加により性能向上する様子を確認



max_parallel_workers max_parallel_workers_per_gather	0	1	2	3	4	5	6	7	8	9	10	16
Elapsed Time（sec）	71.9	35.7	24.4	18.6	15.5	13	11.5	10.6	9.3	8.7	8.6	8.6
Worker 起動数	0	1	2	3	4	5	6	7	8	9	9	9
性能向上比率（パラレルクエリ未使用との比率）	1	2	2.9	3.9	4.6	5.5	6.2	6.8	7.7	8.3	8.4	8.4

検証結果 – 検証①：Count クエリ

- 起動 Worker 数
  - Worker はインデックスサイズによって要求される起動数が決定されていることを確認
  - lineorder の件数カウントに利用される idx_lo_orderdate はサイズが 3,966.5MB であり、想定通り9プロセスが起動されていた

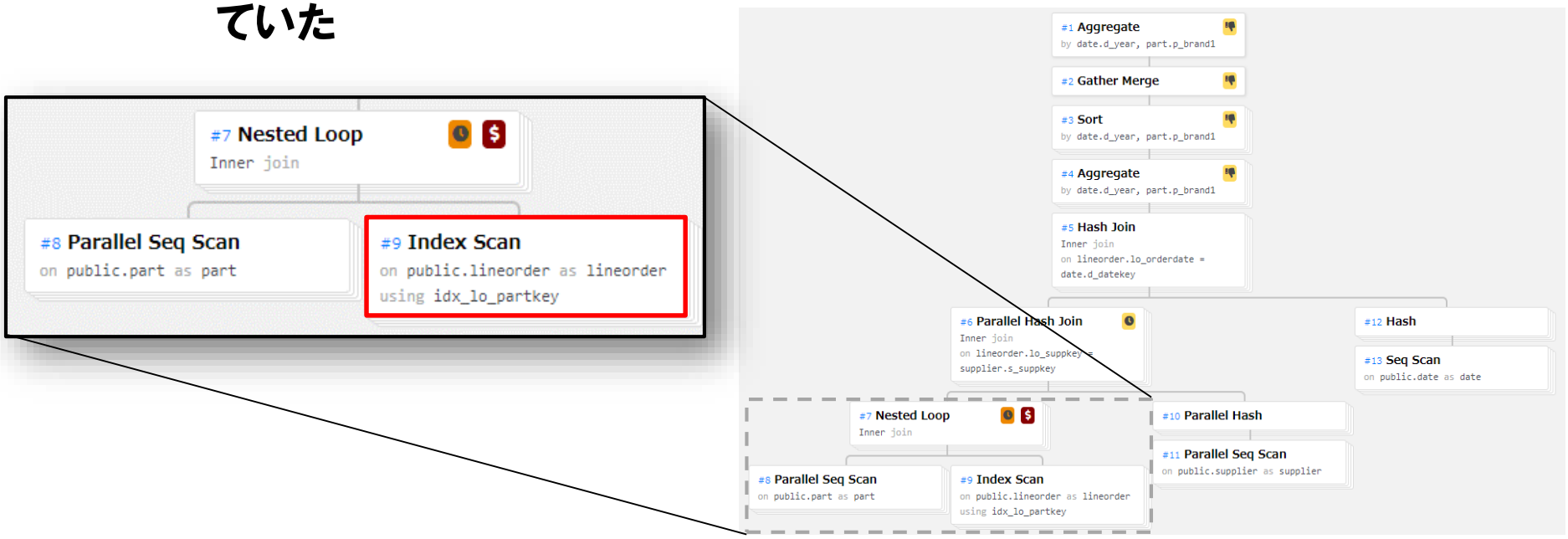
■ クエリ実行計画
(max_parallel_workers=10, max_parallel_workers_per_gather=10)

```
QUERY PLAN
------------------------------------------------------------------------------
 Finalize Aggregate  (cost=4393067.24..4393067.25 rows=1 width=8)
   ->  Gather  (cost=4393066.31..4393067.22 rows=9 width=8)
         Workers Planned: 9
         ->  Partial Aggregate  (cost=4392066.31..4392066.32 rows=1 width=8)
               ->  Parallel Index Only Scan using idx_lo_orderdate on lineorder
                     (cost=0.57..4225389.04 rows=66670905 width=0)
```

インデックスサイズ(MB)	起動Worker数	検索プロセス数
0.5	1	2
1.5	2	3
4.5	3	4
13.5	4	5
40.5	5	6
121.5	6	7
364.5	7	8
1,093.50	8	9
3,280.50	9	10
9,841.50	10	11

検証結果 - 検証②：Star Schema Benchmark クエリ

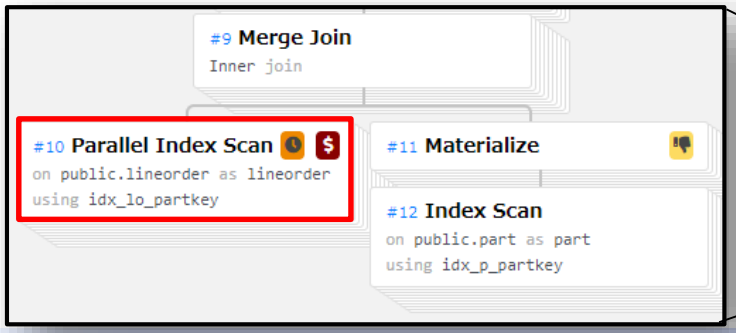
- インデックスありの状態の実行計画
  - 各クエリはインデックスを利用した実行計画とはなったが、Nested Loop Join の内部表として利用され、パラレルインデックススキャンとはならなかった
  - パラレルクエリとしては動くため、Worker 数は Nested Loop Join の外部表のとして選択されたテーブルサイズに依存して上限が決定されていた

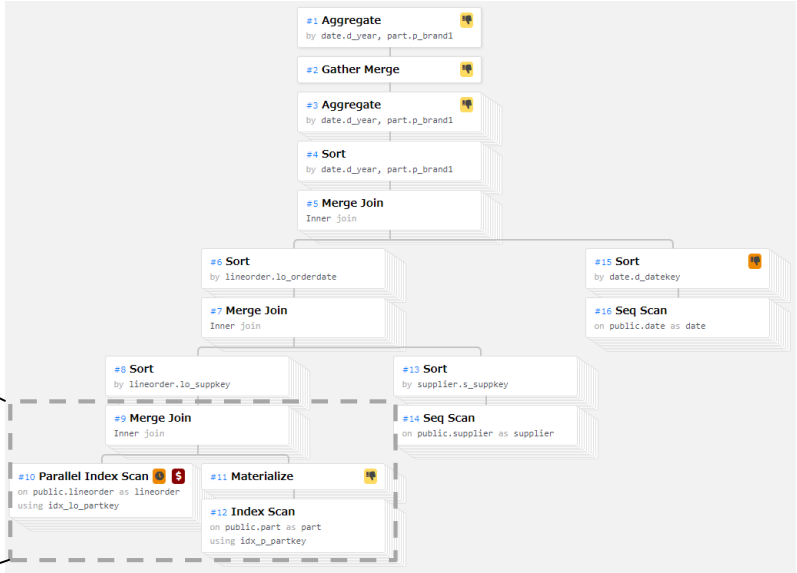


q2.1 の実行計画

検証結果 - 検証②：Star Schema Benchmark クエリ

- Merge Join の強制によるパラレルインデックススキャンの利用
  - パラレルインデックススキャンが選択された場合の挙動を確認するため、Nested Loop Join / Hash Join が選択されない状態に設定
  - Merge Join を強制した場合にはパラレルインデックススキャンが選択されることを確認
  - ただし、Merge Join を強制しない通常の実行計画と比較すると処理実行速度は遅い結果となった
    - DB パラメータ追加設定
      - enable_nestloop = off
      - enable_hashjoin = off

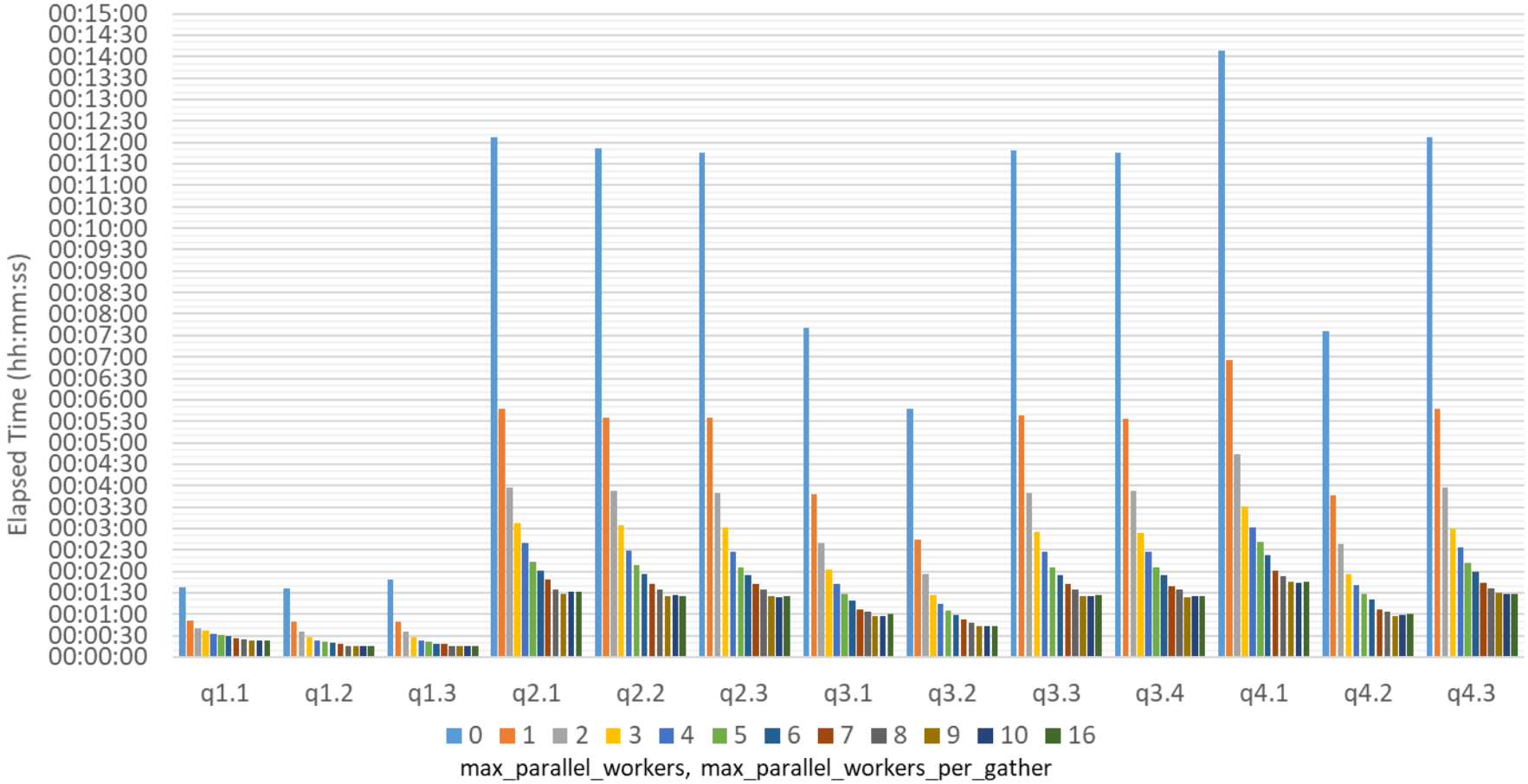




q2.1 の実行計画

検証結果 – 検証②：Star Schema Benchmark クエリ

- 性能測定結果 (Merge Join 強制パターン)
  - 各クエリとも Worker 数の増加により性能向上する様子を確認

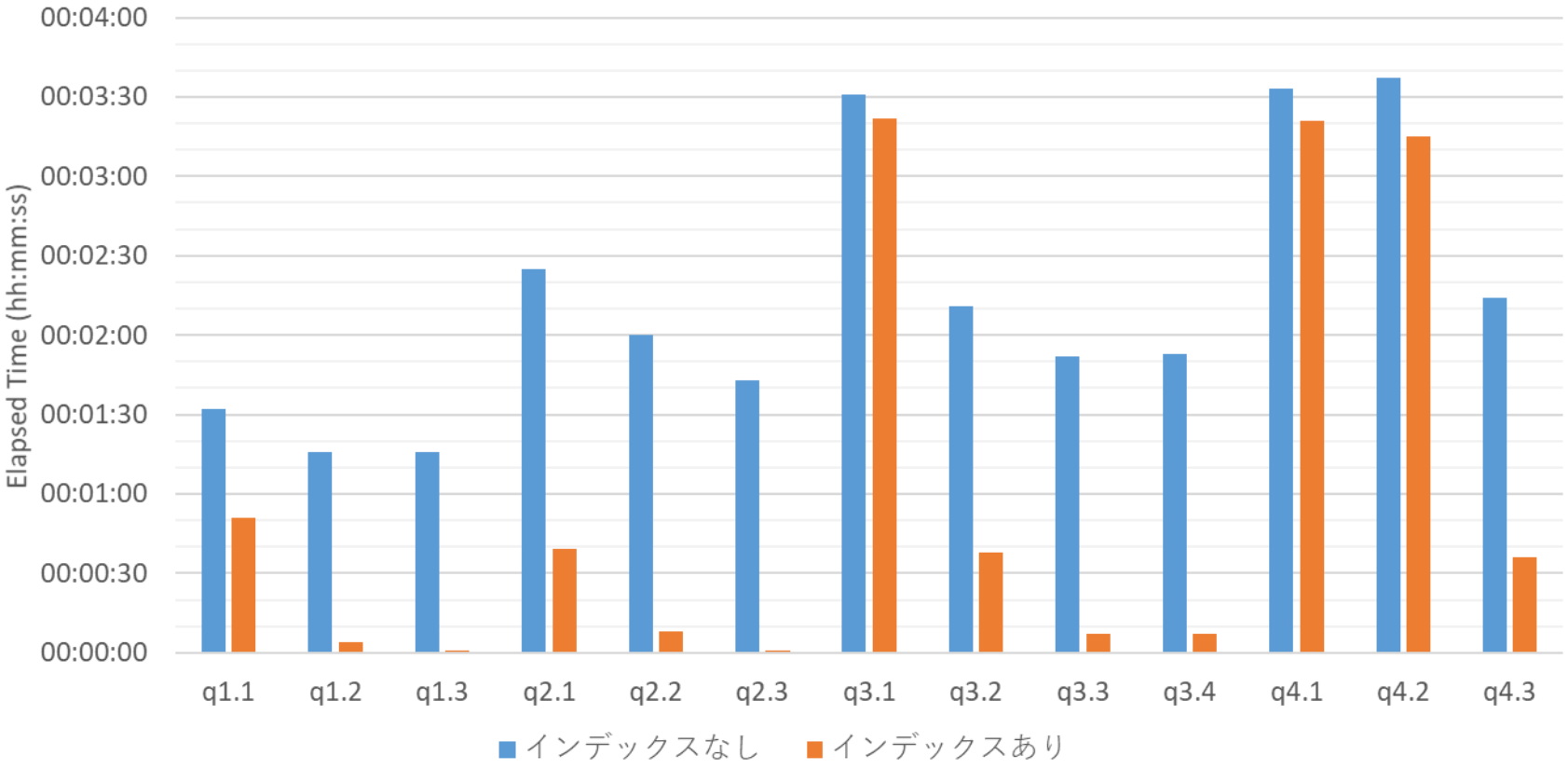


補足

- インデックス有無による性能差
  - パラレルクエリ未使用の場合（Worker 数が 0 の場合）は、各クエリともインデックスありの場合の方が処理実行速度は速く、インデックスの有効性を確認
  - ただし、パラレルクエリを使用した場合、クエリによってはインデックスありの場合の方が処理速度が遅いケースが見られた
  - これは、選択された実行計画で最初にアクセスするテーブルやインデックスのサイズに依存して Worker 数が決定されるため、インデックスなしの場合よりもサイズの小さなテーブルやインデックスからアクセスする実行計画となった際には Worker 数の上限が低くなり、処理速度の向上も頭打ちとなってしまうためと考えられる

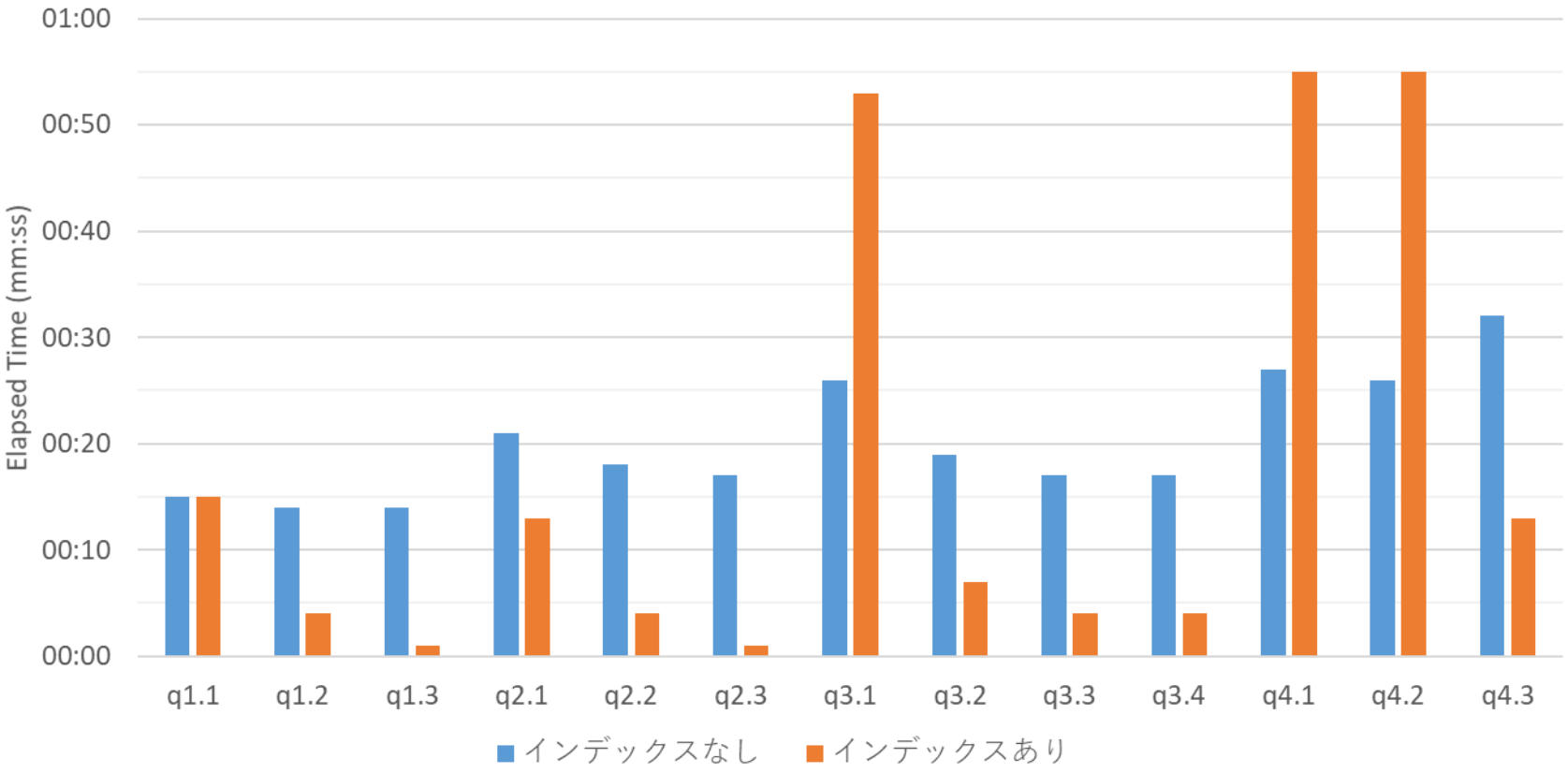
補足 – インデックス有無による性能差

- 性能測定結果：パラレルクエリ未使用の場合
  - いずれのクエリも「インデックスあり」のケースの方が高速
  - ただし、性能向上率はバラバラ



補足 – インデックス有無による性能差

- 性能測定結果：パラレルクエリの場合
  - 「インデックスあり」の方が遅いケースあり（q3.1、q4.1、q4.2）
    - 「なし」では起動 Worker 9 に対し「あり」では起動 Worker 3 となっていた



まとめ

- パラレルインデックススキャンの効果
  - 実行計画としてパラレルインデックススキャンが選択されるクエリについて、Worker 数に応じて性能向上する点、インデックスのサイズにより起動する Worker 数が決定される点を確認
  - ただし、パラレルクエリが有効な場合、インデックスを利用しない方が高速となるケースもあったため、Worker 起動数とともにインデックス要否の判断がチューニングのポイントとなる

注意点

- Parallel Index Only Scan 利用時の注意点

VACUUM の必要性

- Index Only Scan は、参照するページに変更されたデータがあると利用できない
- クエリ実行前に VACUUM ANALYZE により Visibility Map (VM:可視性マップ) が all-visible となっている必要あり

- 本検証で確認した Parallel Index Only Scan も、Visibility Map が all-visible の状態ではなく不可視ページがある場合にはテーブルへの参照が発生し性能劣化する可能性がある点に注意が必要

注意点

- 「パラレルインデックススキャン」と「インデックススキャンのパラレル実行」の違い
  - 本検証では「パラレルインデックススキャン」の性能を目的としたが、これは、1回のインデックス走査を Worker で分担する動作となる
  - 一方、Nested Loop Join の内部表としてインデックスが利用される場合、「インデックススキャンのパラレル実行」が行われるが、これは、何度も発生するインデックス走査を Worker で分担する動作となる
  - いずれもパラレルクエリであり、性能向上が見込まれるため、必ずしもパラレルインデックススキャン (Parallel Index Scan または Parallel Index Only Scan) が選択されなければインデックスアクセスが高速化されないわけではない

注意点

- パラレルインデックススキャン
  - Count クエリ(インデックスオンリースキャン)のケースでの実行計画抜粋

```
->  Parallel Index Only Scan using idx_lo_orderdate on public.lineorder
      (cost=0.57..4225389.04 rows=66670905width=0) (actual time=0.053..26609.216 rows=60003814 loops=10)
    Output: lo_orderdate
    Heap Fetches: 0
    Worker 0:  actual time=0.046..26816.741 rows=61381559 loops=1
    Worker 1:  actual time=0.046..26698.880 rows=59206752 loops=1
    Worker 2:  actual time=0.050..26714.891 rows=60687709 loops=1
    Worker 3:  actual time=0.041..26774.976 rows=59185760 loops=1
    Worker 4:  actual time=0.066..26203.609 rows=59597155 loops=1
    Worker 5:  actual time=0.045..26471.542 rows=59663385 loops=1
    Worker 6:  actual time=0.044..26637.349 rows=60829885 loops=1
    Worker 7:  actual time=0.046..26254.571 rows=60829602 loops=1
    Worker 8:  actual time=0.044..26772.008 rows=57950139 loops=1
```

  - idx_lo_orderdate へのアクセスで 60003814行のデータをカウントしているが、Worker + 1 の10プロセスで分担して各1回ずつアクセスとなっている

注意点

- インデックススキャンのパラレル実行
  - Star Schema Benchmark クエリで Nested Loop Join が選択されるケースでの実行計画抜粋

```
->  Nested Loop  (cost=0.44..331528.46 rows=774285 width=21) (actual time=46.857..2666072.754 rows=598740 loops=4)
      Output: lineorder.lo_revenue, lineorder.lo_orderdate, lineorder.lo_suppkey, part.p_brand1
      Worker 0:  actual time=89.215..2666065.374 rows=600791 loops=1
      Worker 1:  actual time=36.373..2666055.700 rows=610096 loops=1
      Worker 2:  actual time=26.764..2666099.598 rows=572545 loops=1
      ->  Parallel Seq Scan on public.part  (cost=0.00..14684.81 rows=10326 width=13) (actual time=19.352..430.546 rows=7970 loops=4)
            Output: part.p_partkey, part.p_name, part.p_mfgr, part.p_category, part.p_brand1, part.p_color, part.p_type, part.p_size, part.p_container
            Filter: ((part.p_category)::text = 'MFGR#12'::text)
            Rows Removed by Filter: 192030
            Worker 0:  actual time=19.202..410.377 rows=8180 loops=1
            Worker 1:  actual time=19.367..551.952 rows=8054 loops=1
            Worker 2:  actual time=19.326..424.537 rows=7544 loops=1
      ->  Index Scan using idx_lo_partkey on public.lineorder  (cost=0.44..29.68 rows=100 width=16) (actual time=5.296..334.383 rows=75 loops=31882)
            Output: lineorder.lo_orderkey, lineorder.lo_linenumber, lineorder.lo_custkey, lineorder.lo_partkey, lineorder.lo_suppkey, ...
            Index Cond: (lineorder.lo_partkey = part.p_partkey)
            Worker 0:  actual time=5.128..325.821 rows=73 loops=8180
            Worker 1:  actual time=5.180..330.898 rows=76 loops=8054
            Worker 2:  actual time=5.572..353.293 rows=76 loops=7544
```

    - idx_lo_partkey へのアクセスが 31882回実行されており、Worker + 1 の4プロセスで分担して各 8000回前後ずつアクセスしている
    - また、Parallel Seq Scan の各 Worker の rows の数値と、Index Scan の各 Worker の loops の値が一致
      - SeqScan のヒット1行に対して、IndexScanが1回呼ばれている

パラレルクエリ／Windows性能検証

検証概要

- 目的
  - PGEConsでは、過去にパラレルクエリに関してWindows版PostgreSQLを用いた検証を行っていなかった
  - 複雑な集計・分析が必要なOLAPで使用されるクエリを模したベンチマークテストツールであるStar Schema Benchmark (SSB) で定義されているクエリを題材として使用し、
    下記を確認することを目的とした

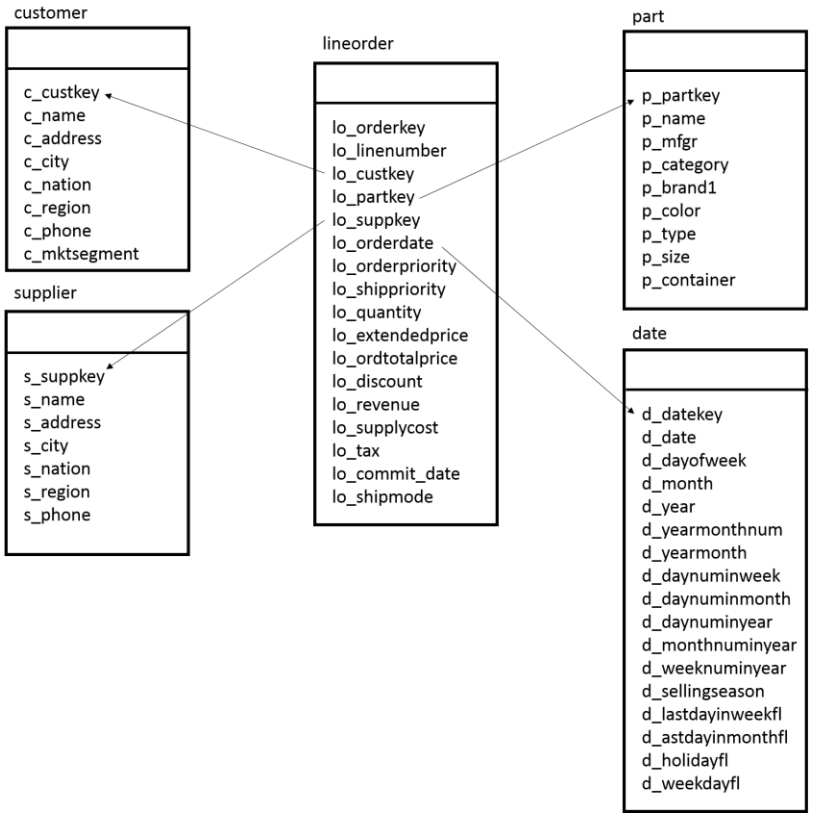
1. Linux版と比較したときのWindows版PostgreSQLのパラレルクエリの性能差の有無を確認

参考：Star Schema Benchmark (SSB)

- 概要
  - Star Schema Benchmarkの論文によって公表されている、TPC-Hをもとに BIで用いられるものを模したファクトテーブル、ディメンションテーブル、クエリが設計されているベンチマークツール
  - 大規模なデータを取り扱うので、I/O周りはもちろんのこと、多数のジョイン操作や集約演算が行われることから、通常のOLTPよりもCPUの処理性能と、CPUをどの程度うまく使えているかがベンチマーク結果に影響

参考：Star Schema Benchmark (SSB)

- **データモデル**
  - スタースキーマ型のデータモデルを採用しており、
    1つのファクトテーブルと複数のディメンションテーブルで構成



- ファクトテーブル：
  分析対象となる実績値が
  格納される明細データで、
  データ件数は多い

- ディメンションテーブル：
  ビジネス上の分析の軸になる
  マスターデータであり、
  データ件数は少ない

検証方法

- 検証方法
  - SSBに定義されているクエリ4パターン/13本のクエリを使用する
  - 本検証のために作成したクライアントプログラムをクライアント上で実行し、DBサーバに対してクエリを実行する
  - クエリ実行前にpg_prewarmで全テーブルデータをメモリにロードする
  - Scale Factorは 100 とする
    - Scale Factor=1で概ね1GBのデータサイズが生成される
    - テーブルに格納すると、60GB程度の物理データサイズとなる
  - 性能評価は処理時間は3回測定した結果の中央値とする
  - 実行計画の比較は EXPIAIN (ANALYZE, COSTS, VERBOSE, FORMAT JSON) で取得したもので実施した

検証環境

ハードウェア

機器	インスタンスタイプ	vCPU	メモリ(GiB)	ストレージサイズ(GiB)	備考
Azure Virtual Machine(※1)	E16s_v4	16	128	200	Premium SSD

※1 クライアントとDBサーバで合計2台を検証では使用した。インスタンスタイプは同一。

ソフトウェア

種類	ソフトウェア名およびバージョン
OS	CentOS-based 8.2, Windows Server 2019
Database	PostgreSQL 13.1(※2)

※2 Windows版はEnterprise DB社提供のインストーラーを使用
https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

検証環境

- SSB検証データ(SF=100)

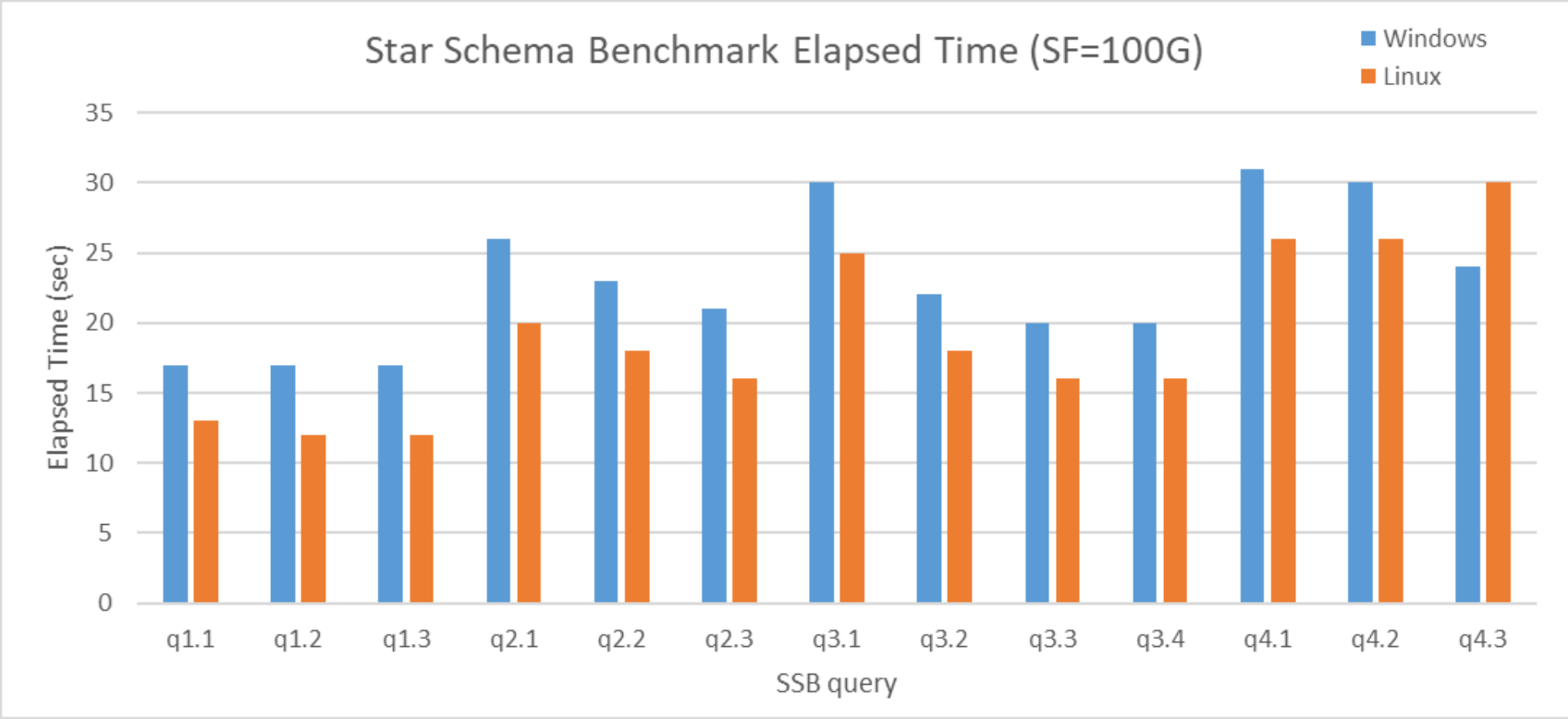
テーブル名	行数 (理論値)	行数 (実測値)	データサイズ (実測値) [Bytes]	データサイズ (実測値) [GB]
customer	SF × 30000	2,999,825	374,030,336	0.348
date	2556	2,556	311,296	0.00029
lineorder	SF × 6000000	600,044,480	63,557,664,768	59.193
part	200000 × (1+log2 (SF)) (小数点切り捨て)	1,400,000	164,265,984	0.153
supplier	SF × 10000	1,000,000	114,376,704	0.107

検証環境

- PostgreSQLの設定値
  - デフォルトから変更した主な項目は以下の通り

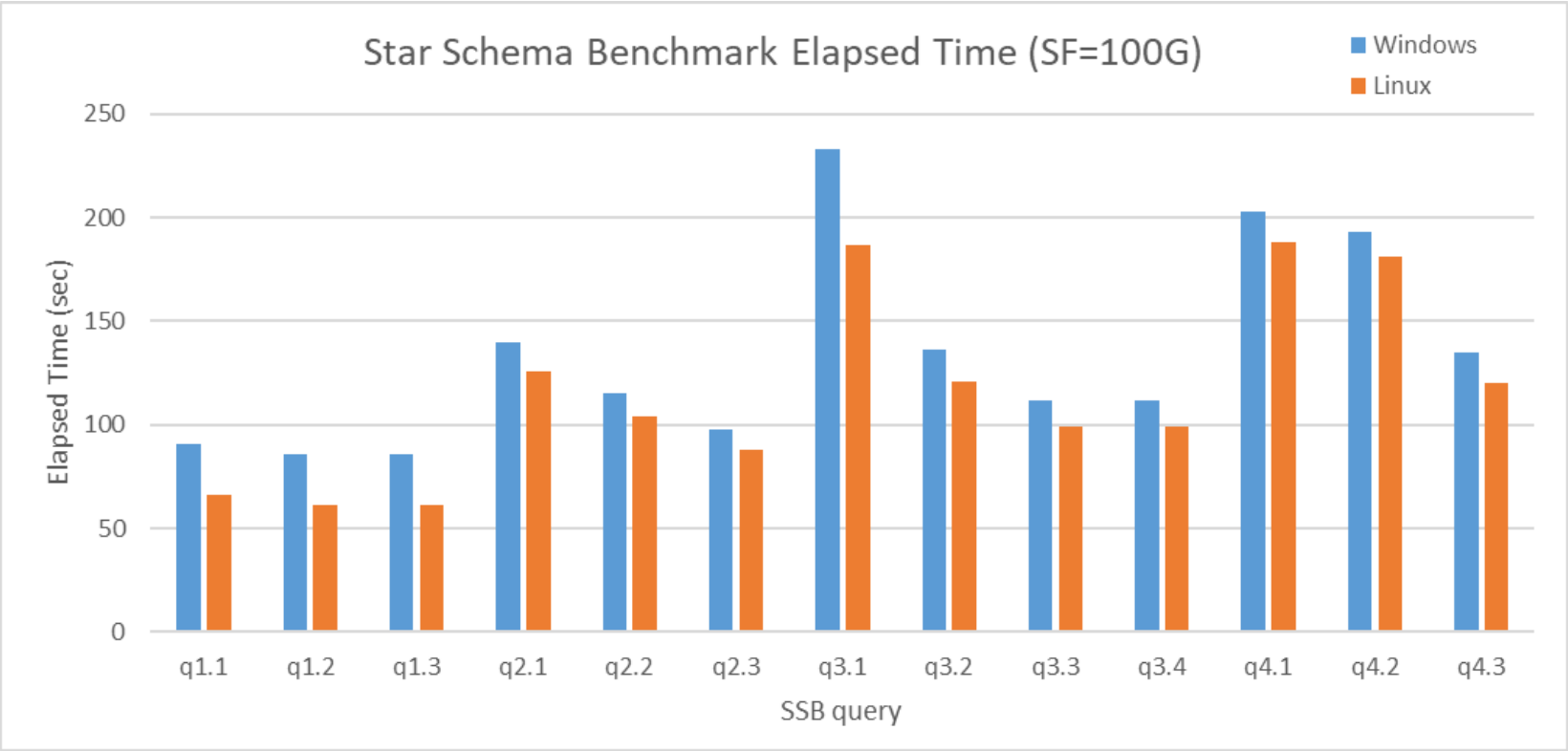
項目名	デフォルト値	設定値	備考
shared_buffers	128MB	96GB	ディスクIOを発生させない値を設定。
effective_cache_size	4GB	96GB	
work_mem	4MB	1GB	
max_worker_processes	8	16	
max_parallel_workers_per_gather	2	16	テーブルサイズからワーカー数は「9」となるので、それを上回る値を設定。
max_parallel_workers	8	16	
jit	off	off	

Star Schema Benchmarkの結果(パラレルクエリ有効)



単位:[秒]	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
Windows	17.5	17.4	16.8	25.6	22.6	21.3	29.8	22.1	20.0	20.3	30.9	29.8	24.4
Linux	12.8	12.1	12.0	20.4	17.9	15.9	25.2	17.7	16.3	16.2	25.8	25.5	30.0

Star Schema Benchmarkの結果(パラレルクエリ無効)



単位:[秒]	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
Windows	90.7	85.8	85.9	139.5	114.8	97.8	233.2	135.7	112.2	112.5	202.8	192.8	134.8
Linux	66.3	61.1	61.2	125.8	103.7	87.8	187.0	121.1	98.8	99.3	188.0	181.3	119.8

Star Schema Benchmarkの結果考察

- Windows版とLinux版において性能差は見られなかった。
  - 全体的にLinux版が性能が良いという結果となったが、パラレルクエリが有効/無効でも傾向が変わらないことから、パラレルクエリ自体に性能差はないと考えられる。

- パラレル有効時のq4_3については、Windows版で起動したパラレルワーカーの要求数が少なかったことが原因であった。
  - Linux版ではワーカー数が「４」となっていたが、Windows版では「９」となっていた。（SeqScan対象のlineorderテーブルは60GBであるため、本来の要求数は「９」となる。）
  - 本事象はPGEConsの2018年度の成果物で確認したものと同じで実行計画による問題と考えられる。

パラレルクエリ／テーブル・パーティショニング性能検証

検証概要

- 目的
  - 大容量のデータに対して複雑な集計・分析を行うクエリは、
    パラレルクエリおよびテーブル・パーティショニングによる、
    性能改善の恩恵を受けやすいとされる

  - 複雑な集計・分析が必要なOLAPで使用されるクエリを模した
    ベンチマークテストツールであるStar Schema Benchmark (SSB) で
    定義されているクエリを題材として使用し、
    下記を確認することを目的とした

1. パラレルクエリ/テーブル・パーティショニングそれぞれの性能影響
2. パラレルクエリとテーブル・パーティショニングを組み合わせた場合の
   性能影響および留意点

検証内容

- 検証A：パラレルクエリの効果確認
  - パラレルクエリ使用時、未使用時のぞれぞれの状態で
    クエリ実行時の処理時間を測定

- 検証B：テーブル・パーティショニングの効果確認
  - レンジパーティショニングでそれぞれ月毎のパーティションを
    作成した状態でクエリ実行時の処理時間を測定
  - 参考：パラレルクエリとテーブル・パーティショニングの比較

- 検証C :パラレルクエリとテーブル・パーティショニングの
  組み合わせによる効果確認
  - 月毎のパーティションを作成した状態でパラレルクエリを有効化し、
    クエリ実行時の処理時間を測定

検証方法

- 検証方法
  - SSBに定義されているクエリ4パターン/13本のクエリを使用する
  - 本検証のために作成したクライアントプログラムをクライアント上で実行し、DBサーバに対してクエリを実行する
  - クエリ実行前にpg_prewarmで全テーブルデータをメモリにロードする
  - Scale Factorは 100 とする
    - Scale Factor=1で概ね1GBのデータサイズが生成される
    - テーブルに格納すると、60GB程度の物理データサイズとなる
  - 検証での処理時間は5回測定した結果の中央値とする

検証環境

■ 検証環境

□ ハードウェア

機器	インスタンスタイプ	vCPU	メモリ(GiB)	ストレージサイズ(GiB)	備考
Azure Virtual Machine	E16s_v4	16	128	200	Premium SSD

□ ソフトウェア

種類	ソフトウェア名およびバージョン
OS	CentOS-based 8.2
Database	PostgreSQL 13.1

検証環境

■ 検証環境

□ SSB検証データ (SF=100)

テーブル名	行数 (理論値)	行数 (実測値)	データサイズ (実測値) [Bytes]	データサイズ (実測値) [GiB]
customer	SF × 30000	2,999,825	374,030,336	0.348
date	2556	2,556	311,296	0.00029
lineorder	SF × 6000000	600,044,480	63,557,664,768	59.193
part	200000 × (1+log2 (SF)) (小数点切り捨て)	1,400,000	164,265,984	0.153
supplier	SF × 10000	1,000,000	114,376,704	0.107

□ 測定値は処理時間 (5回測定、中央値)

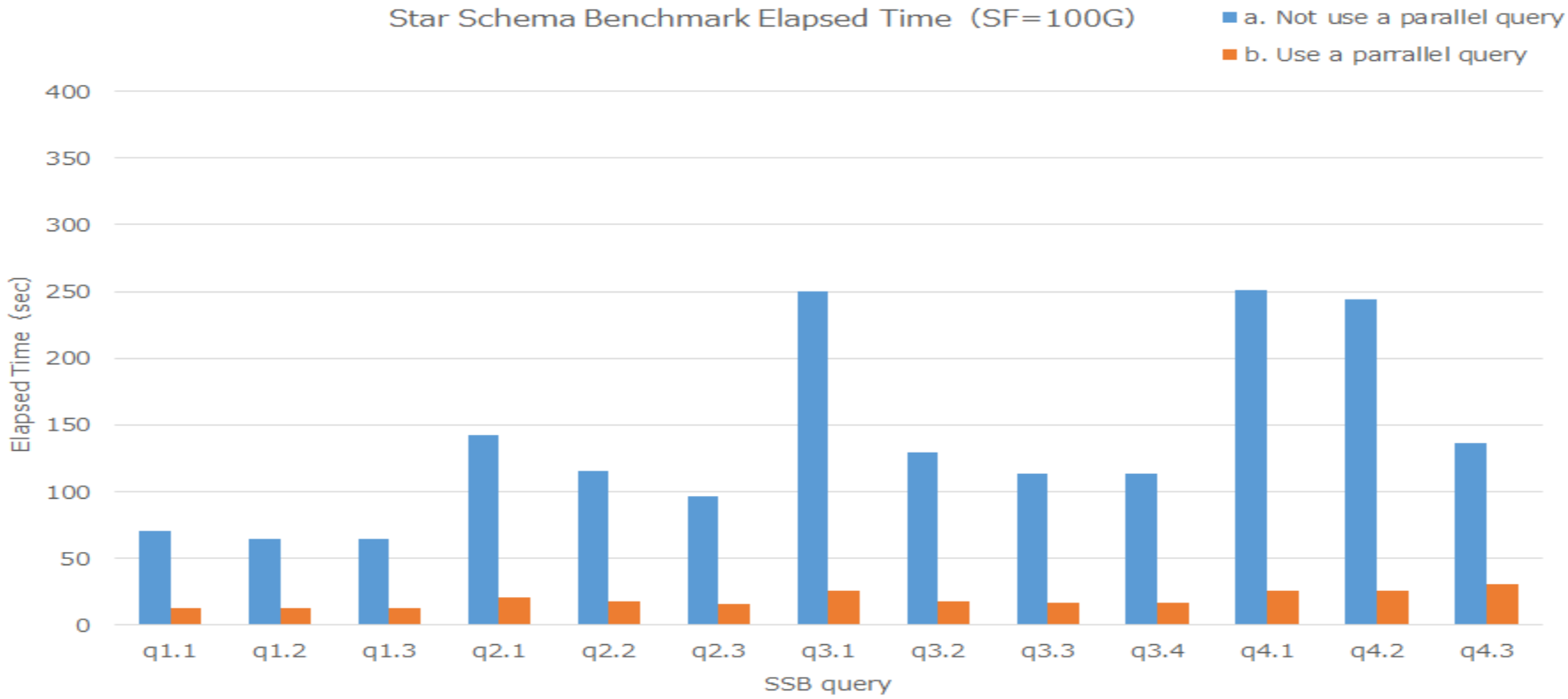
検証内容

- **検証A：パラレルクエリの効果確認**
  - パラレルクエリ使用時、未使用時のぞれぞれの状態で
    クエリ実行時の処理時間を測定

- **検証B：テーブル・パーティショニングの効果確認**
  - レンジパーティショニングでそれぞれ月毎のパーティションを
    作成した状態でクエリ実行時の処理時間を測定
  - 参考：パラレルクエリとテーブル・パーティショニングの比較

- **検証C :パラレルクエリとテーブル・パーティショニングの
  組み合わせによる効果確認**
  - 月毎のパーティションを作成した状態でパラレルクエリを有効化し、
    クエリ実行時の処理時間を測定

検証A：パラレルクエリの効果確認



単位:[秒]	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
パラレルクエリ未使用	70.153	64.541	64.586	142.623	115.156	95.849	249.763	129.444	113.373	113.252	250.72	243.713	136.262
パラレルクエリ使用	12.812	12.118	11.979	20.355	17.924	15.898	25.239	17.747	16.335	16.243	25.762	25.532	30.004
性能向上比	5.48	5.33	5.39	7.01	6.42	6.03	9.90	7.29	6.94	6.97	9.73	9.55	4.54

検証A：パラレルクエリの効果確認　考察

- パラレルクエリの利用により、
  4.54～9.73倍の性能改善を確認できた
- パラレルクエリで起動されるワーカー数は、
  テーブルサイズにより起動する数が決定される
- 利用したlineorderテーブルは60,613MBのため、ワーカ数は9

	ブロック数	サイズ(MB)	起動ワーカ数
×3	1,024	8	1
×3	3,072	24	2
×3	9,216	72	3
×3	27,648	216	4
×3	:		:
×3	6,718,464	52,488	9
×3	20,155,392	157,464	10
	…	…	…

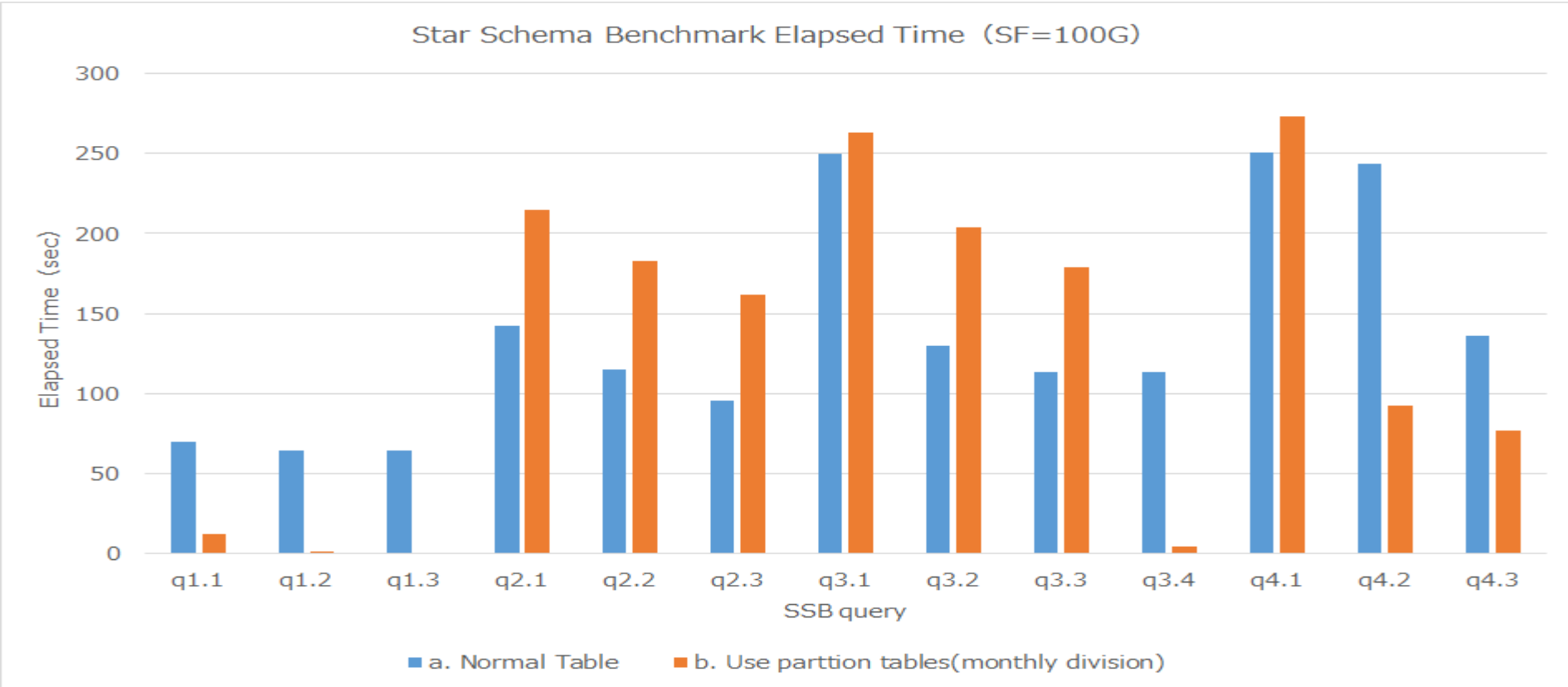
検証内容

- 検証A：パラレルクエリの効果確認
  - パラレルクエリ使用時、未使用時のぞれぞれの状態で
    クエリ実行時の処理時間を測定

- **検証B：テーブル・パーティショニングの効果確認**
  - **レンジパーティショニングでそれぞれ月毎のパーティションを作成した状態でクエリ実行時の処理時間を測定**
  - 参考：パラレルクエリとテーブル・パーティショニングの比較

- 検証C :パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認
  - 月毎のパーティションを作成した状態でパラレルクエリを有効化し、
    クエリ実行時の処理時間を測定

検証B：テーブル・パーティショニングの効果確認



単位:[秒]	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
パーティションなし	70.153	64.541	64.586	142.623	115.156	95.849	249.763	129.444	113.373	113.252	250.72	243.713	136.262
パーティションあり	12.338	0.945	0.819	214.432	183.1	162.104	263.195	203.9	179.16	4.163	273.148	92.129	76.521
性能向上比	5.69	68.30	78.86	0.67	0.63	0.59	0.95	0.63	0.63	27.20	0.92	2.65	1.78

検証B：テーブル・パーティショニングの効果確認　考察

- 各クエリでのパーティションキーとした日付での、絞り込み条件と性能向上比は以下の通り

クエリ	日付での絞り込み条件（必要なデータの割合）	性能向上比
q1.1	1年分のデータ（14.3%）	5.69
q1.2	1ヶ月分のデータ（1.2%）	68.30
q1.3	1週間分のデータ（0.27%）	78.86
q2.1	（無し）（100%）	0.67
q2.2	（無し）（100%）	0.63
q2.3	（無し）（100%）	0.59
q3.1	6年分（85.7%）	0.95
q3.2	6年分（85.7%）	0.63
q3.3	6年分（85.7%）	0.63
q3.4	1ヶ月分のデータ（1.2%）	27.20
q4.1	（無し）（100%）	0.92
q4.2	2年分（28.6%）	2.65
q4.3	2年分（28.6%）	1.78

検証B：テーブル・パーティショニングの効果確認　考察

- **パーティショニング・プルーニング（※1）による性能改善**
  - 1週間分のデータが必要なクエリ（q1.3）：　78.86倍
  - 1ケ月分のデータが必要なクエリ（q1.2,q3.4）：68.30倍、27.20倍
  - 2年分のデータが必要なクエリ（q4.2,q4.3）：2.65倍、1.78倍

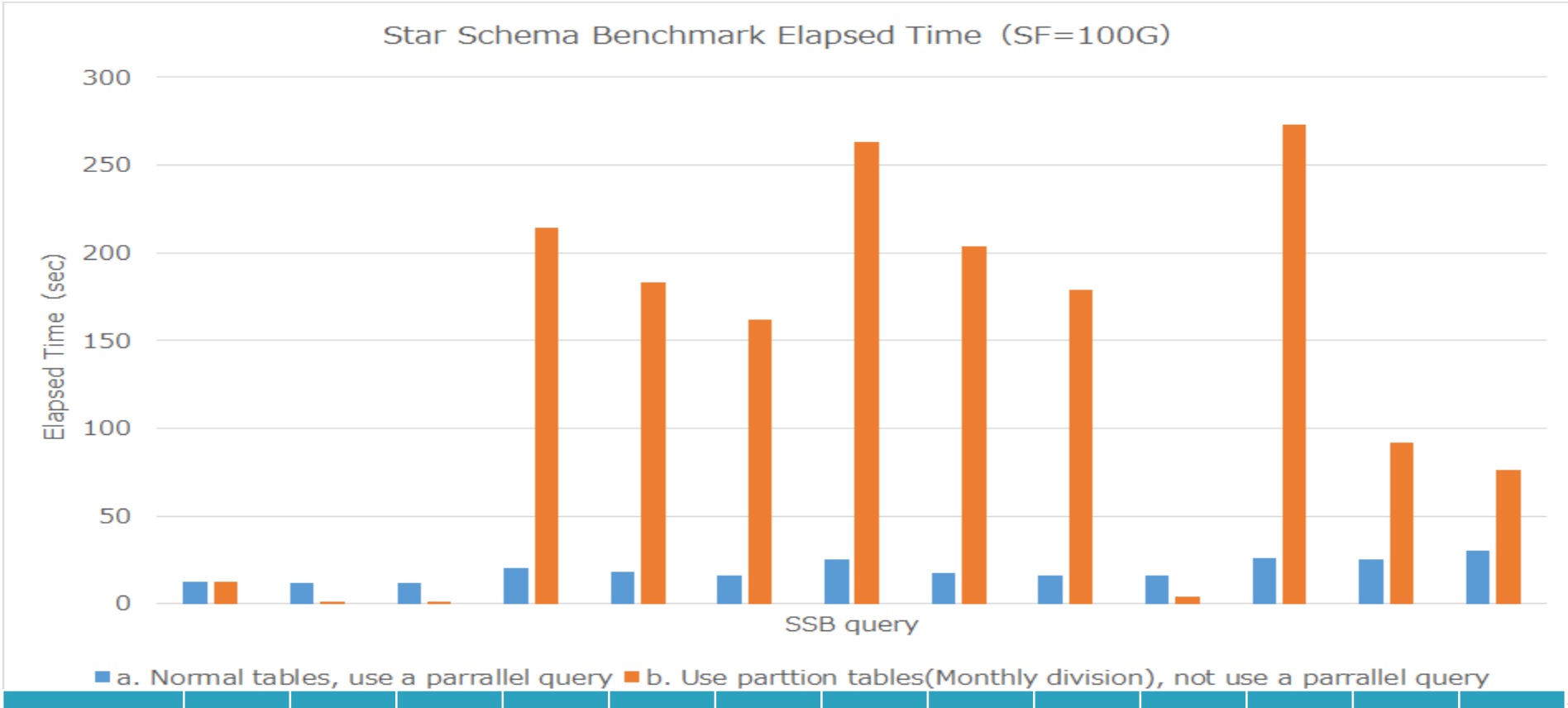
  ※1　検索条件に合致しないパーティションに対するスキャンを省略する機能。
  検索に不要なパーティションへのスキャンを省略することで処理時間を短縮

- **パーティショニングによる性能劣化**
  - 日付での絞り込み条件なしのクエリや、
    2年分のデータが必要なクエリでは性能劣化が確認された
  - テーブルを構成する各パーティションへのスキャン後に、
    各パーティションから取得したレコードをAppend（UNION）する処理が
    実行され、Appendにかかる時間がパーティション利用時のオーバヘッド
  - 取得するレコード数が多いほどAppendに時間がかかるため、
    大量のレコードを取得するクエリでは注意が必要

検証内容

- 検証A：パラレルクエリの効果確認
  - パラレルクエリ使用時、未使用時のぞれぞれの状態で
    クエリ実行時の処理時間を測定
- 検証B：テーブル・パーティショニングの効果確認
  - レンジパーティショニングでそれぞれ月毎のパーティションを
    作成した状態でクエリ実行時の処理時間を測定
  - **参考：パラレルクエリとテーブル・パーティショニングの比較**
- 検証C :パラレルクエリとテーブル・パーティショニングの
  組み合わせによる効果確認
  - 月毎のパーティションを作成した状態でパラレルクエリを有効化し、
    クエリ実行時の処理時間を測定

参考：パラレルクエリとテーブル・パーティショニングの比較



単位:[秒]	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
パラレルクエリ	12.812	12.118	11.979	20.355	17.924	15.898	25.239	17.747	16.335	16.243	25.762	25.532	30.004
テーブル・パーティション	12.338	0.945	0.819	214.432	183.1	162.104	263.195	203.9	179.16	4.163	273.148	92.129	76.521
性能向上比	1.04	12.82	14.63	0.09	0.10	0.10	0.10	0.09	0.09	3.90	0.09	0.28	0.39

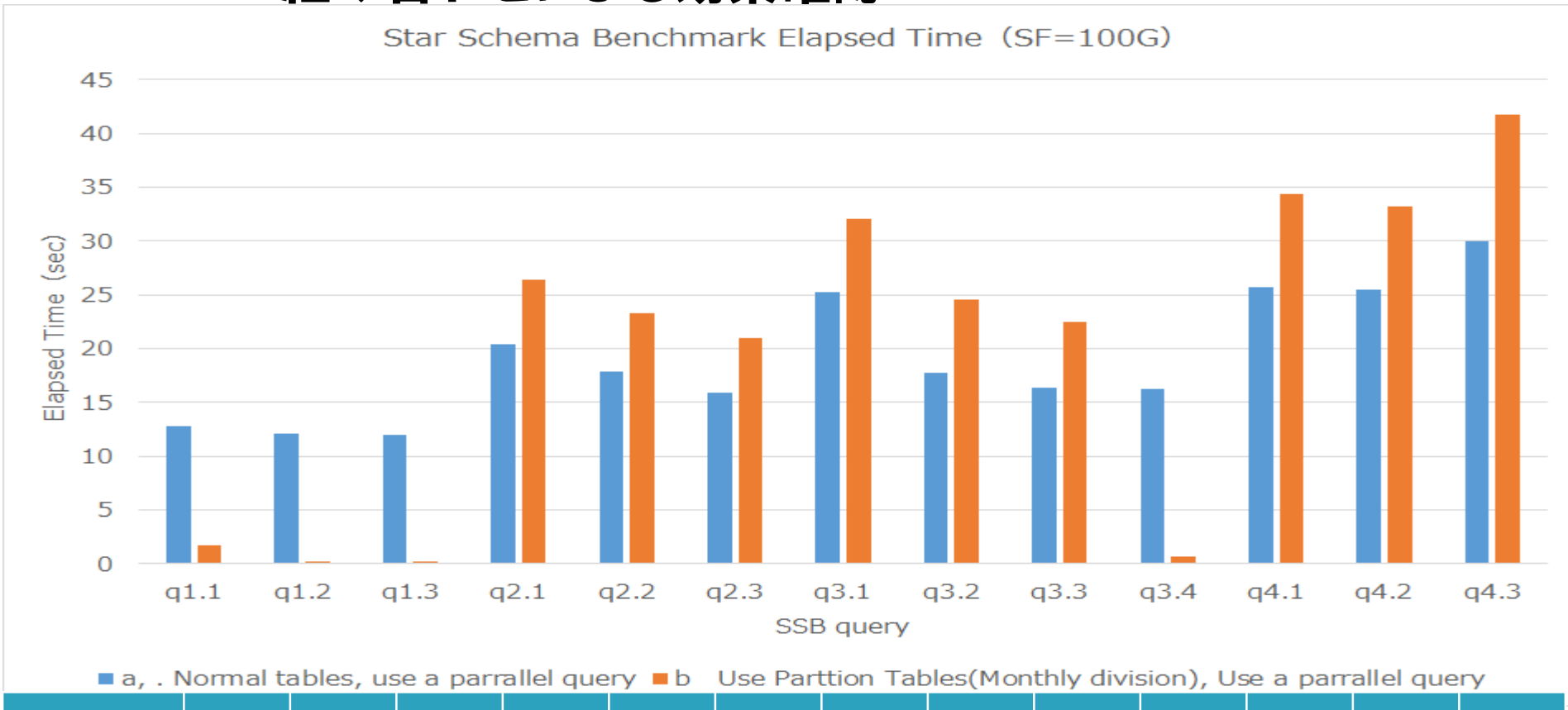
検証内容

- 検証A：パラレルクエリの効果確認
  - パラレルクエリ使用時、未使用時のぞれぞれの状態で
    クエリ実行時の処理時間を測定

- 検証B：テーブル・パーティショニングの効果確認
  - レンジパーティショニングでそれぞれ月毎のパーティションを
    作成した状態でクエリ実行時の処理時間を測定
  - 参考：パラレルクエリとテーブル・パーティショニングの比較

- **検証C :パラレルクエリとテーブル・パーティショニングの
  組み合わせによる効果確認**
  - **月毎のパーティションを作成した状態でパラレルクエリを有効化し、
    クエリ実行時の処理時間を測定**

検証C：パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認



単位:[秒]	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
パラレルクエリ	12.812	12.118	11.979	20.355	17.924	15.898	25.239	17.747	16.335	16.243	25.762	25.532	30.004
パラレルクエリ+パーティション	1.721	0.232	0.211	26.431	23.242	21.005	32.035	24.557	22.451	0.642	34.359	33.25	41.703
性能向上比	7.44	52.23	56.77	0.77	0.77	0.76	0.79	0.72	0.73	25.30	0.75	0.77	0,72

検証C：パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認　考察

- **パラレルクエリとテーブル・パーティショニングの組み合わせによる性能改善**
  - 1週間分のデータが必要なクエリ(q1.3)： 56.77倍
  - 1ケ月分のデータが必要なクエリ(q1.2,q3.4)：52.23倍、25.30倍
  - 組み合わせにより更なる性能改善が確認できた

- **パラレルクエリとテーブル・パーティショニングの組み合わせによる性能劣化**
  - 2年分のデータが必要なクエリ(q4.2,q4.3)： 0.77倍、0.72倍
  - 6年分のデータが必要なクエリ(q3.1,q3.2,q3.3)
    ：0.79倍、0.72倍、0.73倍
  - 全てのデータが必要なクエリ(q2.1,q2.2,q2.3)
    ：0.77倍、0.77倍、0.76倍
  - パラレルクエリのみの場合と比べ、本検証では
    およそ**20%～30%の性能劣化**

検証C：パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認　考察

- 性能劣化要因調査1
  - 各パーティションテーブルのワーカー数を調査したところ、従来のパラレルクエリでは、ワーカー数が9となっていたのに対し、テーブル・パーティショニングを利用した場合は、7となっていた
  - ソースコードからテーブル・パーティショニングでのワーカー起動数はパーティション数を基に決定していることを確認

```
(ソースコードより抜粋)
if (enable_parallel_append)
{
    parallel_workers = Max(parallel_workers,
                           fls(list_length(live_childrels)));
    parallel_workers = Min(parallel_workers,
                           max_parallel_workers_per_gather);
}
Assert(parallel_workers > 0);
```

  - テーブル・パーティショニング利用時のパラレルクエリクエリ起動数は、「少数点以下切り上げ (log2 (パーティション数)) 」で算出可能

検証C：パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認　考察

- 性能劣化要因調査1（追加検証）
  - ワーカー起動数が処理時間に影響しているのではないかと考え、pg_hint_planを用いてワーカー起動数を強制的に9に変更して性能が改善するかを確認

	パラレルクエリ（ワーカー数:9）	パラレルクエリとテーブル・パーティショニングの組み合わせ（ワーカー数:7）	パラレルクエリとテーブル・パーティショニングの組み合わせ（ワーカー数:9）
q2.1	20.355 s	26.431 s	25.564 s

  - ワーカー起動数を9に変更して追加検証を行ったが、大幅な性能改善は見られなかった

検証C：パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認　考察

- 性能劣化要因調査2
  - 改めて実行計画を確認したところ、各パーティションへのSeq Scanに利用されるワーカー数が異なっている

```
->  Parallel Append  (cost=0.00..0.00 rows=66678639 width=16) (actual time=0.018..15260.222 rows=60003814 loops=10)
      Worker 0:  actual time=0.020..14775.131 rows=72975180 loops=1
      Worker 1:  actual time=0.016..15985.096 rows=41250157 loops=1
      Worker 2:  actual time=0.012..14788.680 rows=72555040 loops=1
      Worker 3:  actual time=0.030..16000.167 rows=41689661 loops=1
      Worker 4:  actual time=0.008..14768.869 rows=72786323 loops=1
      Worker 5:  actual time=0.024..14692.645 rows=72109601 loops=1
      Worker 6:  actual time=0.028..16062.315 rows=41518687 loops=1
      Worker 7:  actual time=0.022..14784.564 rows=71853808 loops=1
      Worker 8:  actual time=0.011..14714.544 rows=72372799 loops=1
      ->  Parallel Seq Scan on public.lineorder_199201 lineorder_1  (cost=0.00..0.00 rows=859244 width=16) ...[省略]
            Output: lineorder_1.lo_revenue, lineorder_1.lo_orderdate, lineorder_1.lo_partkey, lineorder_1.lo_suppkey
            Worker 4:  actual time=0.006..1119.096 rows=7733051 loops=1
      :
      ->  Parallel Seq Scan on public.lineorder_199711 lineorder_71  (cost=0.00..0.00 rows=831702 width=16) ...[省略]
            Output: lineorder_71.lo_revenue, lineorder_71.lo_orderdate, lineorder_71.lo_partkey, lineorder_71.lo_suppkey
            Worker 1:  actual time=0.020..7.313 rows=21991 loops=1
            Worker 3:  actual time=0.022..1145.522 rows=3761844 loops=1
            Worker 4:  actual time=0.017..6.210 rows=38656 loops=1
            Worker 5:  actual time=0.024..532.416 rows=3568783 loops=1
            Worker 7:  actual time=0.012..14.826 rows=94593 loops=1
       :
       ->  Parallel Seq Scan on public.lineorder_199804 lineorder_76  (cost=0.00..0.00 rows=830465 width=16) ...[省略]
            Output: lineorder_76.lo_revenue, lineorder_76.lo_orderdate, lineorder_76.lo_partkey, lineorder_76.lo_suppkey
```

ワーカー数=1

ワーカー数=5

検証C：パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認　考察

- 性能劣化要因調査2
  - 実行計画から、各パーティションに割り当てられているワーカー数を調査し、それぞれのSeq Scanの処理時間の平均を算出したところ、ワーカーが割り当てられていないパーティションでの遅延を確認

ワーカー数	該当数のワーカーでSeq Scanが実行されたパーティション数	各パーティションのSeq Scanにかかった平均時間（ms）
9	1	176.2
8	0	(-)
7	0	(-)
6	0	(-)
5	0	341.2
4	0	(-)
3	1	578.6
2	5	549
1	66	1368.2
0	5	2314.2
0 （データが少ないパーティション）	1	163.4
0 （データが存在しないパーティション）	4	0

検証C：パラレルクエリとテーブル・パーティショニングの組み合わせによる効果確認　考察

- 性能劣化要因調査2
  - Seq Scanに割り当てられたワーカー数が増加することで、テーブル・パーティションにおけるSeq Scanの時間は短縮されているが、ワーカーが割り当てられなかった（ワーカー数:0）Seq Scanは並列に実行されることがなく、順次実行となるためワーカー数0の処理が全体のボトルネットとなると考えられる

ワーカが割り当てられなかった（ワーカー数:0）Seq Scanにかかる時間
= 各パーティションSeq Scanにかかった平均時間（ワーカー数:0）＊ パーティション数
= 2314.2ms ＊ 5 + 163.4ms ＊ 1
= 11734.4ms

  - ワーカーが割り当てられなかった（ワーカー数:0）パーティションへのSeq Scanが順次実行され、性能のボトルネックになる事象への対処方法は現在調査中

まとめ

- 複雑な分析を行うOLAPで使用されるクエリを題材に以下を確認できた
  - パラレルクエリ利用時の性能影響
    - 複雑な分析を行うOLAPを模したクエリではパラレルクエリによる性能改善の効果が大きい
  - テーブル・パーティショニング利用時の性能影響
    - パーティションキーを検索条件として指定し、キーを用いた絞り込みによりアクセスするパーティションの大部分を省略できるクエリでは性能改善効果が大きい
    - 上記以外のクエリでは、各パーティションから取得したレコードの集約処理によるオーバヘッドの影響が大きく、性能が劣化
  - パラレルクエリとテーブル・パーティショニングの組み合わせ利用時の性能影響
    - OLAPのワークロードにおいては、パラレルクエリと比較するとテーブル・パーティショニングで性能改善できるクエリは限られている
    - 各パーティションに対してパラレルスキャンが実行されるが、ワーカーが割り当てられないパーティションへのスキャンがボトルネックとなり性能劣化が発生（本検証では20%～30%）

まとめ

- 複雑な分析を行うOLAPにおける各機能の利用方針は以下とする
  - パラレルクエリ
    - 性能改善の効果が大きく、OLAPにおける利用を推奨
  - テーブル・パーティショニング
    - OLAPのような検索条件が不定なクエリで利用する場合、
      テーブル・パーティショニングによる性能へのオーバヘッドが発生するため、
      運用性の向上（データの一括削除等）とクエリの処理性能のトレードオフとなる
  - パラレルクエリとテーブル・パーティショニングの組み合わせ
    - パーティショニングのケースと同じ

PGECons

PostgreSQL Enterprise Consortium