

WG2活動報告書  
パラレル処理移行編

# 目次

目次	2
1. 改訂履歴	4
2. ライセンス	5
2. はじめに	6
2.1. 本資料の目的	6
2.2. 本資料で記載する範囲	6
2.3. 本資料で扱う用語の定義	6
2.4. 本資料で扱うDBMSおよびツール	6
3. PostgreSQLの平行処理	7
3.1. OracleとPostgreSQLの平行処理について	7
3.2. 平行処理の機能差	7
3.3. 平行処理の指定方法	8
3.4. 平行処理の優先度	9
4. 平行処理の適用	12
4.1. 平行指定方法の指針	12
4.2. 運用上の考慮点	12
5. 平行処理まとめ	14
5.1. 平行処理指定の適用方針案	14
6. 著者	15



# 1. 改訂履歴

版	改訂日	変更内容
1.0	2020/03/17	新規作成

## 2. 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。

- Eclipseは、Eclipse Foundation,Inc.の米国、およびその他の国における商標もしくは登録商標です。
- 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。
- Intel、インテルおよびXeonは、米国およびその他の国における Intel Corporation の商標です。
- Java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。
- 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。
- Windows は米国 Microsoft Corporation の米国およびその他の国における登録商標です。
- TPC, TPC Benchmark,TPC-C, TPC-E, tpmC, TPC-H, 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。

## 2. はじめに

### 2.1. 本資料の目的

本資料は、以下の参考資料として利用されることを想定しています。

- Oracle Enterprise Editionでパラレル処理を活用し性能要件を満たしている場合にPostgreSQLへデータベースを移行する場合
- パラレル処理を有効活用し性能向上を行う場合

### 2.2. 本資料で記載する範囲

本資料では、パラレル処理におけるOracleとの機能差、PostgreSQLでの指定方法と考慮点を中心に記載します。パラレル指定時の性能比較については本資料では取り扱っておりません。性能面に関しては、「2017年度WG1活動報告書」、「2018年度WG1活動報告書」等を参照してください。

### 2.3. 本資料で扱う用語の定義

資料で記述する用語について以下に定義します。

表 2.1 用語定義

No.	用語	意味
1	DBMS	データベース管理システムを指します。
2	Oracle	データベース管理システムの Oracle Database を指します。

### 2.4. 本資料で扱うDBMSおよびツール

本書のパラレル処理の優先度、考慮点は以下のDBMSを前提にした調査結果を記載します。

表 2.2 本書で扱うDBMS

DBMS名称	バージョン
PostgreSQL	11.4

## 3. PostgreSQLの паралел処理

### 3.1. OracleとPostgreSQLの паралел処理について

PostgreSQLではバージョン9.6より паралелクエリがサポートされました。本書では паралелクエリに限定せず、 паралел化できる処理として扱います。そのため、表題等も“ паралел処理”としました。Oracleから移行を前提にした場合、Oracle Enterprise Editionでは паралел処理がサポートされており、 паралел処理を前提で性能を確保している場合もあります。PostgreSQLに移行した場合、 паралел化せずに性能要件を満たしていれば問題ありませんが、満たしていない場合はどのように対応すればよいか。また、 паралел処理がサポートされないOracle Standard Editionからの移行では、PostgreSQLに移行することで паралел処理の恩恵を受ける部分もあり、移行後のシステムの性能改善にも繋がります。本書ではOracleとPostgreSQLの паралел処理の機能差を示し、PostgreSQLでの運用における指定方法について説明します。

### 3.2. паралел処理の機能差

PostgreSQLではバージョン9.6で паралелクエリがサポートされる以前より一部機能については паралел化が可能な処理もありました。ここではPostgreSQLのバージョン毎に単純にOracleとの機能差を示します。

- паралел処理の機能差  
PostgreSQLとOracleの паралел処理の機能差を以下表に示します。PostgreSQLの機能には一部周辺ツールも含まれます。脚注も合わせて確認して下さい。

機能分類	比較機能	Oracle12c		PostgreSQL				
		SE	EE	9.5	9.6	10	11	12
クエリ	Seq Scan	—	○	—	○	○	○	○
	Hash Join			—	○	○	○	○
	Nested Loop Join			—	○	○	○	○
	集約 (Aggregation)			—	○	○	○	○
	ヒント句			—	○*1	○*1	○*1	○*1
	IndexScan(B-tree)			—	—	○	○	○
	Merge Join			—	—	○	○	○
	Bitmap Heap Scan			—	—	○	○	○
	サブクエリ			—	—	○	○	○
	ソート			—	—	○	○	○
	Parallel Hash Join*2			—	—	—	○	○
	паралелアペンド			—	—	—	○	○
	SELECT INTO			—	—	—	○	○
	分離レベルがSERIALIZABLE			—	—	—	—	○
	外部表 (PostgreSQLはFILE_FDW)			—	—	—	—	—
DML	INSERT,UPDATE,DELETE	—	○	—	—	—	—	—
DDL	CREATE TABLE ~ AS SELECT	—	○	—	—	—	○	○
	CREATE MATERIALIZED VIEW			—	—	—	○	○
	CREATE INDEX(B-tree)			—	—	—	○	○
	ALTER INDEXまたはREINDEX			—	—	—	○	○
その他	統計取得 (DBMS_STATS,ANALYZE)	—	○	—	—	—	—	—
	バックアップ (rman,PITR)			○*3	○*3	○*3	○*3	○*3
	論理バックアップ(DataPump,pg_dump)			○*4	○*4	○*4	○*4	○*4
	データロード			○*5	○*5	○*5	○*5	○*5

○：機能あり、—:機能なしまたは対象外

\*1:pg\_hint\_plan 1.2以降を使用

\*2:ハッシュ作成も паралелで実行される。バージョン10までのハッシュ結合は結合する2つのテーブルの1つのみ паралелスキャンとなる

\*3:Barman 2.2以降を使用 (ただし паралел化には複数のBarmanサーバが必要)

\*4:pg\_dumpはディレクトリ形式で複数のテーブルを処理する場合、pg\_restoreはディレクトリ形式およびカスタム形式が対象  
 \*5:pg\_bulkload 3.1以降を使用

### 3.3. パラレル処理の指定方法

#### 1. パラレル処理の指定箇所

以下はOracle,PostgreSQLでパラレル処理の指定箇所を表にしたものです。3.2.パラレル処理の機能差の表の”その他”は除きます（それぞれのユーティリティでパラレル度を指定します）。Oracleに関してはEnterprise Editionのみとなります。移行を考慮した場合、移行元のパラレル処理の具体的な指定を踏襲するのではなく、“どのような指針で計画されているか”を分析し、新たに指針として計画することを推奨します。多くの場合、数年以上前のハードウェアで稼動するデータベースを最新のハードウェアで稼動させるだけで性能は向上します。以前はパラレル指定が必要であったケースでも不要となる場合もあるからです。

RDBMS	パラレル処理の指定箇所	説明
Oracle (EEのみ)	初期化パラメータ	自動、手動含め様々なワークロードに対応
	リソースマネージャ	
	表・索引	パラレル実行の有無、パラレル度の指定
	セッション単位	
	ヒント句	
PostgreSQL	postgresql.conf (パラメータ)	パラレル度の制限、実行計画の調整要素を指定
	表	parallel_workersで制限（テーブルサイズ依存なし）
	ロール	set句でpostgresql.confのパラメータを上書き
	セッション	set文でpostgresql.confのパラメータを上書き
	ヒント句(pg_hint_plan 1.2.0~)	ヒント句でパラレル度を指定

#### 2. パラメータの指定

PostgreSQLのパラレル処理関連で指定できるパラメータについて、運用上考慮すべきものについて以下表に示します。デフォルト値はバージョンにより異なる場合があるので該当するバージョンのマニュアルを参照してください。

表 3.1 パラメータの指定値

パラメータ	指定可能バージョン	説明
max_worker_processes	全て	・インスタンス内のバックグラウンドプロセスの最大値。サーバ内でデータベースインスタンスに割り当てるCPUコア数を指定する。このパラメータはパラレル動作させるためのパラメータではなくパラレルクエリで使うCPUコアを制限するものと考えてよい。 ・ロール、セッション指定でも上書きすることはできない。
max_parallel_workers	9.6以降	・インスタンス内のパラレルクエリ操作のワーカー数の最大値。 max_worker_processes以下を指定する。 ・ロール、セッション指定で上書きできてしまうため、インスタンスでのパラレルクエリで使うCPUコアの制限値とすることはできない。
max_parallel_workers_per_gather	9.6以降	GatherまたはGather Mergeノードに対して起動できるワーカー数の最大値。SQL内での1つの処理で動作するパラレル度と考えた方が分かりやすい。複数ユーザが接続し複数のSQLが同時に動作する場合などmax_parallel_workersの限界に達しパラレル実行できないケースもある。
max_parallel_maintenance_workers	11以降	create index,reindexなどメンテナンス処理で使用するワーカー数の最大値。
min_parallel_index_scan_size	10以降	パラレルスキャンが考慮されるために、スキャンされなければならないインデックスデータの最小量。インデックスサイズではないことに注意。
min_parallel_table_scan_size	10以降	パラレルスキャンを考慮する最小のテーブルデータのサイズ。SEQ SCANの場合はテーブルサイズと同じです。
min_parallel_relation_size	9.6	min_parallel_table_scan_sizeと役割は同じ。

#### 3. 表、ロール、セッション、pg\_dump、pg\_restoreでのパラレル度指定方法例

表、ロール、セッションでのパラレル度の指定方法について例文で示します。ロール、セッションではすべてのパラメータを上書きできるわけではありません。



- 表での指定方法  
pgbench\_accounts表の平行度を3にする。

```
alter table pgbench_accounts set ( parallel_workers = 3 );
```

- ロールでの指定例  
postgresロールのmax\_parallel\_workers\_per\_gatherを4に上書きする。

```
alter role postgres set max_parallel_workers_per_gather = 4;
```

- セッションでの指定例  
当セッションのmax\_parallel\_workers\_per\_gatherを4に上書きする。

```
set max_parallel_workers_per_gather = 4;
```

- ヒント句での指定例  
pgbench\_accounts表に対しての平行度を4にする。

```
/*+ Parallel(pgbench_accounts 4) */  
select count(*) from pgbench_accounts;
```

- pg\_dumpでの指定例  
test01データベースを/home/postgres/backup/test01\_dディレクトリに-jオプションを使用して4並列（4つのテーブルを同時）でダンプする。

```
pg_dump -Fd -v -j 4 -f /home/postgres/backup/test01_d test01
```

- pg\_restoreでの指定例  
/home/postgres/backup/test01\_dのバックアップをtest01データベースに-jオプションを使用して4並列（4つのテーブルを同時）でリストアする。

```
pg_restore -c -v -j 4 -d test01 /home/postgres/backup/test01_d
```

### 3.4. 平行処理の優先度

運用では”postgresql.confのパラメータで指定し、かつ特定ロールでパラメータを上書き”のように複数の箇所で平行度が指定されるケースがあります。その場合、指定した平行度は以下の4つのルールで優先度が決まりで実行計画が作成されます。実行に関してはmax\_worker\_processes等の制限を受け、実行計画通りに平行化されない場合もあります。

#### ①基本的な指定優先度

上位の優先度が下位の優先度の平行度を拡張または制限できます。

ヒント句 > セッション指定 > ロール指定 > **postgresql.conf**

#### ②テーブルの平行度を指定した場合

テーブルの平行度を指定した場合は他の指定に対し、より制限される（低い）平行度が適用されます。

例1 セッション指定の平行度よりテーブルの平行指定が低いケース

テーブルの平行度：3、セッション指定の平行度：4のときテーブルの平行度で計画される。

```
test01=# alter table pgbench_accounts set ( parallel_workers = 3 );
ALTER TABLE
test01=# set max_parallel_workers_per_gather = 4;
SET
test01=# explain select count(*) from pgbench_accounts;
                QUERY PLAN
-----
Finalize Aggregate (cost=409514.48..409514.49 rows=1 width=8)
-> Gather (cost=409514.16..409514.47 rows=3 width=8)
    Workers Planned: 3
-> Partial Aggregate (cost=408514.16..408514.17 rows=1 width=8)
    -> Parallel Seq Scan on pgbench_accounts (cost=0.00..392385.13 rows=6451613 width=0)
(5 行)
```

例2 テーブルの平行指定よセッション指定の平行度が低いケース

テーブルの平行度：6、セッション指定の平行度：4のときセッション指定の平行度で計画される。

```
test01=# alter table pgbench_accounts set ( parallel_workers = 6 );
ALTER TABLE
test01=# set max_parallel_workers_per_gather = 4;
SET
test01=# explain select count(*) from pgbench_accounts;
                QUERY PLAN
-----
Finalize Aggregate (cost=391369.42..391369.43 rows=1 width=8)
-> Gather (cost=391369.00..391369.41 rows=4 width=8)
    Workers Planned: 4
-> Partial Aggregate (cost=390369.00..390369.01 rows=1 width=8)
    -> Parallel Seq Scan on pgbench_accounts (cost=0.00..377869.00 rows=5000000 width=0)
(5 行)
```

③テーブルの平行度を指定しない場合

テーブルの平行度を指定しない場合はテーブルのサイズにより平行度が制限が決定され、他の指定に対し、より制限される（低い）平行度が適用されます。

以下の図は「[PGECons 勉強会#2 平行クエリ](#)」からの抜粋になります。

## □ 平行度はテーブルのサイズに依存

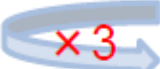
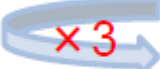
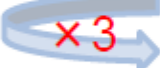
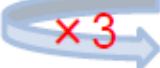
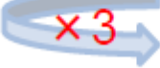

	ブロック数	サイズ(MB)	起動Worker数	検索プロセス数
	1,024	8	1	2
	3,072	24	2	3
	9,216	72	3	4
	27,648	216	4	5
	82,944	648	5	6
	248,832	1,944	6	7
	...	...	...	...

図 3.1 テーブルサイズと平行度

④平行処理の方が早いと想定される実行計画のとき

パラレルでない実行計画よりパラレル実行計画のコスト推定値が低い場合にパラレル実行が計画されます。

## 4. パラレル処理の適用

### 4.1. パラレル指定方法の指針

データベースが搭載されるサーバは複数のCPUコアを搭載し運用される場合がほとんどです。そして、運用されるシステム内でも性能要求が“高い処理”と“低い処理”があります。性能要求が“高い処理”であっても小さなデータ量であれば1つのCPUコアで十分に性能要件を満たすこともできます。パラレル処理は性能要求が“高い処理”でかつデータ量が多い場合に要件を達成させるための手法のひとつとなります。パラレル指定方法の指針は、限りあるCPUリソースの中で、必要なときに必要な処理には必要なCPUを割り当てられるようにすることです。以下にいくつかのCPU割当の基準を示します。実際の運用設計にあたっては、これらを複合的に組合わせて適切なワークロードとなるようにします。

- パラレル処理の全体のCPUを制限する

データベースサーバにデータベース以外のサービスが搭載される場合、または複数のデータベースインスタンスを起動させる必要がある場合など対象のデータベースサービスだけでサーバCPUを使い切らないように制限する必要があります。具体的にはpostgresql.confでパラメータのmax\_worker\_processesを指定します。ただし、このパラメータの変更はインスタンスの再起動が必要となるため、運用スケジュールによるワークロード変更が必要な場合かつ運用上、再起動をしたくない場合はmax\_parallel\_workersで指定します（このパラメータはロールやセッションで上書き可能なため、運用ルールとして上書きしないようにするなど考慮が必要）。

- ロールのSQLで使用するCPUを制限する

- 複数のロール（ユーザ）を使用する場合、ロールによっては高い性能要求が必要ないものもあるかもしれません。そのようなロールでのSQL実行にはCPUの割当を制限します。
- オンライン処理で同時実行トランザクションが多く、パラレル化でCPUが枯渇するようであれば、その接続ロールではパラレル処理が動作しないようにします。

- セッション指定でCPUを確保する

特定のバッチ処理などpostgresql.confの指定より十分なCPUを使用して性能を確保したい場合もあります。この場合はセッションで十分なCPUを割り当てられるようにします。ただし運用全体でCPUを枯渇させない考慮を合わせて行います。

- ヒント句でCPUを確保する

セッション指定と同じ意図ですが、よりピンポイントに指定する方法です。

- 一定サイズ以上のテーブル以外はパラレル処理が動作しないようにする

要求性能を満たせる場合はmin\_parallel\_table\_scan\_sizeやmin\_parallel\_index\_scan\_sizeを指定してパラレル処理が動作しないようにします。運用上、効果の低い処理にCPUを使用するより必要な処理にCPUを確保可能とするための処置です。

### 4.2. 運用上の考慮点

PostgreSQLはパラレルクエリ以外にも3.2 パラレル処理の機能差の表の“その他”で示したように個別のユーティリティでもパラレル処理を実装しているものがあります。この場合、それぞれのユーティリティで指定したパラレル度以外でプロセスを使用する場合もあるので設計には注意が必要です。

- pg\_restoreでの例

pg\_restoreではアーカイブ形式がカスタム形式、ディレクトリ形式の場合にパラレル処理で復元できるとなっております。実際には以下の表のように、リストア中のそれぞれの処理でプロセスが起動します。

表 4.1 pg\_restoreで発生するプロセス

リストアでの処理	pg_restore*1	パラレルクエリの機能*2	自動バキュームの機能*3
コピー処理	複数の表を同時にコピー		
索引の作成	複数の索引を同時に処理	1つの索引を複数のプロセスで処理	
表のANALYZE			複数の表を同時にANALYZE

\*1:pg\_restoreの-jパラメータで指定するジョブ数

\*2:max\_parallel\_maintenance\_workers の指定値

\*3:autovacuum\_max\_workersの指定値

実際に以下に例を示します。

- 実行コマンド  
前提として、max\_worker\_processes=8,max\_parallel\_maintenance\_workers=4, autovacuum\_max\_workers=2とします。

```
$ pg_restore -c -j 4 -d test01 backup/test01.dmp
```

- 処理起動後の前半  
4並列でコピーが実行され、終わったものはANALYZEが開始されています。

```
test01=# select backend_type,query from pg_stat_activity where state = 'active';
 backend_type | query
-----
client backend | select backend_type,query from pg_stat_activity where state = 'active';
client backend | COPY public.pgbench_accounts (aid, bid, abalance, filler) FROM stdin;
client backend | COPY public.pgbench_accounts5 (aid, bid, abalance, filler, upd_date) FROM stdin;
client backend | COPY public.yubin (tihou_koukyou_dantai_cd, yubin_no, jyusyo_kana, jyusyo, flg1, flg2) FROM stdin;
client backend | COPY public.pgbench_accounts4 (aid, bid, abalance, filler, upd_date) FROM stdin;
autovacuum worker | autovacuum: ANALYZE public.pgbench_accounts3
(6行)
```

図 4.1 pg\_restore処理前半のプロセス

- 処理起動後の後半  
索引作成がパラレルで動作している。この後、索引の作成が完了すればpg\_restoreはANLYZEを待たずに完了する。（ただし、リストアップした全てテーブルに対しANLYZEは継続して続行される。並列度が高く、リソースが枯渇しなければその分早く完了する）

```
test01=# select backend_type,query from pg_stat_activity where state = 'active';
 backend_type | query
-----
client backend | select backend_type,query from pg_stat_activity where state = 'active';
client backend | CREATE UNIQUE INDEX pgbench_accounts4_idx1 ON public.pgbench_accounts4 USING btree (aid);
client backend | CREATE INDEX pgbench_accounts4_idx2 ON public.pgbench_accounts4 USING btree (bid);
parallel worker | CREATE INDEX pgbench_accounts4_idx2 ON public.pgbench_accounts4 USING btree (bid);
parallel worker | CREATE UNIQUE INDEX pgbench_accounts4_idx1 ON public.pgbench_accounts4 USING btree (aid);
parallel worker | CREATE INDEX pgbench_accounts4_idx2 ON public.pgbench_accounts4 USING btree (bid);
parallel worker | CREATE UNIQUE INDEX pgbench_accounts4_idx1 ON public.pgbench_accounts4 USING btree (aid);
parallel worker | CREATE UNIQUE INDEX pgbench_accounts4_idx1 ON public.pgbench_accounts4 USING btree (aid);
parallel worker | CREATE INDEX pgbench_accounts4_idx2 ON public.pgbench_accounts4 USING btree (bid);
autovacuum worker | autovacuum: ANALYZE public.pgbench_accounts
(10行)
```

図 4.2 pg\_restore処理後半のプロセス

## 5. パラレル処理まとめ

### 5.1. パラレル処理指定の適用方針案

まとめとしてPostgreSQLにおけるパラレル処理指定の指針と考え方案を表に示します。中規模であっても1つのテーブルで数百GBあるような場合はテーブルのパーティショニング化の考慮も必要でしょう。また、同じく中規模でも小さなテーブルばかりで性能要求が低いのであれば運用コスト低減のために”指針なし”もあり得ます。運用要件によって最適な指針は変わると考えますので参考程度としてして下さい。

表 5.1 パラレル処理指定の適用方針案

DB規模	前提条件	指針	考え方
小規模	<ul style="list-style-type: none"> <li>DBサイズは50GB未満</li> <li>DBに割り当てられるCPU(コア)が4以下</li> <li>接続ユーザ数は10未満</li> <li>性能要件は規定されない</li> </ul>	指針なし。postgresql.confのデフォルトのまま	<ul style="list-style-type: none"> <li>管理コストは減らす(管理しない)</li> </ul>
中規模	<ul style="list-style-type: none"> <li>DBサイズは50GB～1TB</li> <li>DBに割り当てられるCPU(コア)が32以下</li> <li>接続ユーザ数は50未満</li> <li>性能要件が規定される</li> </ul>	パラレル化方式を規定し、ワークロードに合わせて指定	<ul style="list-style-type: none"> <li>優先度の高い処理の性能を確保</li> </ul>
大規模	<ul style="list-style-type: none"> <li>DBサイズは1TB以上</li> <li>DBに割り当てられるCPU(コア)が潤沢</li> <li>接続ユーザ数は50以上</li> <li>性能要件が規定される</li> </ul>	中規模と同様+必要に応じてパーティショニング含め性能確保を検討	<ul style="list-style-type: none"> <li>優先度の高い処理の性能を確保</li> <li>パラレル化のみでは性能を確保できない可能性あり。運用が成立するための追加方式を検討</li> </ul>

## 6. 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WG2)	富士通エフ・アイ・ピー株式会社	ビジネス推進本部	多田 明弘