実行計画関連

実行計画を説明する前にクエリ統計情報を説明する。

クエリ統計情報

●pg_stat_statements モジュール

①システムカタログでも情報コレクターでもない contrib モジュール（追加・拡張モジュール）。

※contrib モジュールとは、限られたユーザ向けや実験的なものということで PostgreSQL 本体に取り込まれていないモジュールのこと。contrib は contribution の略で、ユーザから寄贈されたことに由来するもの。

②設定ファイルにおいて、shared_preload_libraries = pg_stat_statements 、及び compute_query_id = on（又は auto）とする。

③スーパーユーザ又は定義済みロールである pg_read_all_stats（全ての pg_stas_＊を参照可能）は、クラスタ内の全てのクエリの統計情報（SQL 文、実行回数、処理時間等）を pg_stat_statements ビューで参照可能

④定義済みロールの設定例は GRANT pg_read_all_stats TO user1;

●log_min_duration_statement パラメータ（以下のパラを含め設定及び変更はスーパーユーザのみ。以下 SU）

①250ms をセットした場合、完了未完を問わず、250ms 以上の処理時間を要した全ての SQL 文がログとして記録

②そのため、0 をセットすると全ての SQL 文がログとして記録されてしまう。

③また、デフォルトである-1 は、何も記録しない。

④例えば log_min_duration_sample = 250ms、log_statement_sample_rate = 0.5 とすると、250ms 以上の処理時間を要した全ての SQL 文の 50%がログとして記録されるため、ログファイルの肥大化を防止する。

クエリ統計情報の説明の中でログが出てきたので、関連するログについて説明する。

ログ関連

●Silver 範囲の log_destnation、logging_collector、log_directory、log_filename 等の他に以下のログがある。

●log_duration（SU、デフォは off）：全ての完了した SQL の経過時間のログ（SQL 文はログしない）

●log_statement（SU、デフォは none）

①どの SQL をログするのかを制御。ddl は DDL、mod は DDL+DML そして all がある。

②all にしても単純な構文エラーはログしない。つまり Execute 段階以前のエラーはログ対象外なので、当該ログが必要ならば log_min_error_statement（デフォは ERROR）を ERROR 以下に設定する。

③監査ログに相当するログであるが、内容が不十分な上に、出力が膨大な量となるため、DB の性能劣化を引き起こす可能性がある。

●log_connections（SU、デフォは off）：クライアントによるサーバへの接続に係る成功終了と認証ログ

●log_disconnections（SU、デフォは off）：セッション終了とセッション経過時間のログ

●log_truncate_on_rotation （デフォは off）：off で同じ名前の log 名に対して追記。on は上書き

ここから本題

PREPARE コマンドと EXPLAIN コマンド

●PREPARED 文というが、PREPARED コマンドではなく PREPARE コマンドである。以下の AS の後は SELECT, INSERT, UPDATE, DELETE, VALUES 等が来る。ponta は任意の名前で OK であるが、セッション内で一意。

（例）PREPARE ponta (int, int) AS SELECT ＊ FROM … $1…$2 … ;

●EXPLAIN ANALYZE EXCUTE ponta (1000, 500) ;は、プリペアド文を EXCUTE で実行して実行計画を再利用し、EXPLAIN で実行計画を表示し、ANALYZE で実処理時間を表示する。

●PREPARE コマンドにより、実行計画を必ず再利用されるためには、plan_cache_mode パラメータ（デフォルトは auto）を force_generic_plan にする。その都度、実行計画を作成するためには、force_custom_plan とする。

スロークエリの実行計画の表示

●EXPLAIN コマンドを実行しない限り、実行計画は表示されない。そこで contrib モジュールの auto_explain モジュールを導入する。

●全セッションにおいて当該モジュールの使用は、設定ファイルで shared_preload_libraries = auto_explain とし、特定のセッションでの使用は、psql 等で接続後、LOAD 'auto_explain';とする。

●当該モジュールにより、EXPLAIN コマンドを実行することなく、**スロークエリ**の実行計画を auto_explain.log_min_duraton（使用方法及び権限は log_min_duration_statement と同じ）で表示できるが、これだけでは情報がスカスカなので、EXPLAIN コマンドと同様に、auto_explain.log_analyze（実処理時間）を設定（true）したうえで、auto_explain.log_timing（各ノードの実処理時間）や auto_explain.log_buffers（共有バッファ及びディスクから読み込んだページ数）を設定（true）する。

●上記の補足であるが、EXPLAIN ANALYZE コマンドは、実処理時間と実出力行数が表示されるが、これは、デフォで TIMING パラメータも true になっているためである。これによってクエリは時間計測のためにオーバーヘッドがかかることにより遅くなる。実出力行数だけが必要であれば TIMING パラを false にすることも考慮すべきである。同様に auto_explain.log_timing も auto_explain.log_analyze が有効の時に有効になるが、スロークエリがさらに遅くなる結果を引き起こす。

実行計画の制御

●SET 文で変更可能な enable_seqscan や enable_hashjoin 等の enable を接頭語にもつパラメータは、デフォルトで on であり、実行計画に利用可能なノード要素であることを意味する。逆に off にすることで、利用不可とすることができるが、インデックスがないのに、enable_seqscan を off にすると、コストを 100 億にして、シーケンシャルスキャンを実行することになる。

●contrib モジュールの利用

①実行計画制御ツールである pg_hint_plan モジュールを使い、EXPLAIN コマンドの後にヒント句として付加することで、ノード要素を強制的に変更及び実行することができる。全セッション及び特定セッションでの使用方法は auto_explain モジュールに同じ。

②実行計画制御ツールである pg_dbms_stats モジュールを使い、理想的な統計情報を固定化することができる。特にテーブルの更新が激しくて ANALYZE が間に合わない場合に有効である。全セッション及び特定セッションでの使用方法は auto_explain モジュールに同じ。

●拡張統計情報は contrib モジュールではなく、CREATE STATISTICS 文で定義できる。通常の統計情報はカラム毎であるが、WHERE 句等に指定した 2 つのカラムに関数的依存性の可能性がある場合、CREATE STATISTICS 文でその 2 つのカラムを指定することで実行計画が最適化される。例示は 2 つのカラムとしたが、組み合わせごとに定義する必要があり、拡張統計情報は、pg_statistic ではなく、pg_statistic_ext_data システムカタログ（対応するビューは pg_stats_ext）に保持される。

●実行計画の作成に当たり、多くのパターンがある場合、全てを試すのではなく、遺伝的問い合わせ最適化（GEQO：Generic Query Optimization）を実行している。これは、geqo（デフォルトで有効）が有効であること、FROM 項目の閾値（最小値：デフォ 12）を設定する geqo_threshold が効いているためである。よって、pg_hint_plan で制御したい場合は、geqo_threshold（全てを試す範囲）を大きくして GEQO が機能しないようにする等の措置をとる。