4L-03

実体化ビュー選択の最適化によるクエリ処理の高速化

乗松奨真 †
大阪大学情報科学研究科 †

涌田悠佑 ‡
大阪大学情報科学研究科 ‡

佐々木勇和 §
大阪大学情報科学研究科 §

鬼塚真 ¶
大阪大学情報科学研究科 ¶

1 はじめに

データベース内で行われる分析処理には多くの共通した処理が含まれている．この共通した処理の結果を実体化ビューとして保存しておくことでそのクエリ処理結果をベーステーブルにアクセスせず実体化ビューを利用して再利用できる様になる．これにより，ワークロード全体での処理速度が向上することが知られている [1].

利用する実体化ビューによってクエリ処理がどの程度高速になるかが決定するため，実体化ビューとして保存する処理を適切に決めることが問題点となる．考慮すべき項目には．実体化ビューを利用することによって得られるクエリ利得，実体化ビューの更新処理の速度が挙げられる．この二つの項目は，実体化ビューを利用することにより得られる利得が高い場合その実体化ビューに含まれる処理は計算に時間がかかり，その実体化ビューに関連する更新処理は再度計算し直す必要があり，長時間となる．逆に利得が小さい時，更新処理は短時間である．よってこの二つの項目はトレードオフの関係にある．また，ストレージ容量の大きいサブクエリを実体化した時，このサブクエリに関する更新が行われると，そのストレージ容量の大きいサブクエリ自体の更新を行う．そのため，ストレージ容量が大きいサブクエリ程，更新処理を遅くする．よって，更新処理の速度は実体化ビューのストレージ容量と同等に扱える．本稿ではストレージ容量の上限を制約条件として，実体化ビューを利用することによって得られるクエリの利得を最大化する最適化問題を解く．しかし，解の候補はワークロードに含まれる全てのクエリ処理となるため，解の候補が膨大すぎて整数計画問題で処理することができない．本稿では，クエリ処理を高速化するために実体化すべき重いクエリ処理や出現頻度の高い処理を含んだ別の処理やそこに含まれるもう一方の処理を解候補とすることで解候補を削減し，整数計画問題を解く．

2 問題定義

2.1 節で各種変数，事前知識について説明し，2.2 節で整数計画問題での定式化を説明する．

2.1 事前知識

· **サブクエリ**:ワークロードから得られるクエリ群はそれぞれ実行プランへと変換され，それは木構造で表すことができる．この木構造に含まれる全ての処理 (部分木) がサブクエリ s_i となる．· **サブクエリの重複**: 図 1 に示す様に，各クエリの木構造の間で一致した木構造が存在する．これがサブクエリの重複であり，このサブクエリを実体化することで，同じ処理の繰り返しを防ぐことができる．従ってこのようなサブクエリを実体化ビューとして保存し，処理結果を直接取り出すことで、同じ処理を繰り返すことなく，高速なクエリ処理が可能となる．

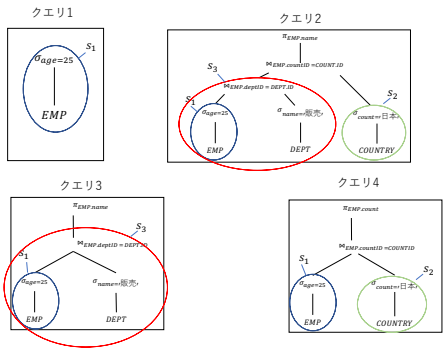

図 1 サブクエリの重複

· **サブクエリの利得**:BIGSUBs[1] ではクエリ q_i 内のサブクエリ s_j の利得 u_{ij} は s_j を実体化せずに q_i を処理したときのコスト $cost(i)$ と実体化して処理したときのコスト $cost(i|s_j)$ の差で利得を評価している．

2.2 整数計画問題を用いたスキーマ最適化

この項では，整数計画問題を用いたスキーマ設計の具体例を示す．y_{ij} は q_i に含まれる s_j を実体化する時に 1 をとるバイナリ変数，z_j を s_j が実体化される時に 1 をとるバイナリ変数とすると

$$maximize \sum_i \sum_j u_{ij} \cdot y_{ij} \tag{1}$$

(1) 式を以下の制約下で解く

· 制約 1:ストレージ制約 (b_j は s_j のストレージ，B_{max} は使用できるストレージ容量)

$$\sum_i b_j \cdot z_j \leq B_{max} \tag{2}$$

· 制約 2:一つのクエリに対して重複する二つのサブクエリは同時に実体化しない (過剰評価制約)

上記の整数計画問題は，巨大なワークロードに対しては解の候補 (s_j) が膨大すぎて実行するために長時間かかるため，解くための工夫が必要である．

3 提案手法

本稿ではサブクエリの出現頻度とそのコストを考慮し，多数の解候補を削減して，効率的に整数計画問題を実行し，クエリの処理時間を低減させる．

3.1 提案手法の定式化

2.1 節でも述べたように一つのクエリからは多数の実行プランが考えられる．そこに含まれる全ての部分木がサブクエリとなるためサブクエリの総数は膨大なものとな

Making Query-Processing Faster by Optimizing Selection of Materialized View

† Shoma Norimatsu, Osaka University
‡ Yusuke Wakuta, Osaka University
§ Sasaki Yuya, Osaka University
¶ Makkoto Onizuka, Osaka University

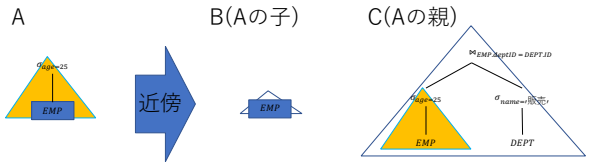

図 2　サブクエリ A の近傍，子:B、親:C

る．提案手法では，予め解候補を利得の高いものとその近傍に限定する．よって解候補に含まれるサブクエリのみの包含関係を考えれば良くなる．提案手法では，サブクエリ同士の包含関係は目的関数 (3) に組み込む．つまり，重複して実体化に選ばれてしまったサブクエリは子の部分クエリのコストを引くことで調整する．この調整は 2.2 節で述べた制約 (2) を達成するためのものである．定式化は以下となる．ただし後述するが，O_i は s_i の出現頻度，s_i は s_i のコスト，A_i は解候補となった s_i の集合，a_i は s_i が実体化されるときに 1 となるバイナリ変数．[*1]

$$maximize \sum_i (O_i \cdot c_i \cdot a_i - \sum_{A_j \subset A_i} O_i \cdot c_j \cdot a_j) \qquad (3)$$

制約条件はストレージ制約のみとなる．

$$\sum_i b_i \cdot a_i \leq B_{max} \qquad (4)$$

上記の整数計画問題を解候補を探索しながら繰り返し解くことでクエリ処理速度の低減に最も貢献できる実体化ビュー候補を得ることができる．

3.2　有用サブクエリの導出

前節で述べた利得の定義では，1 つのクエリに対し，多数のサブクエリがあるようなクエリでは，利得の計算だけに膨大な時間を要してしまう [2]．本項での s_i の利得について定義する．利得はサブクエリの出現頻度と前項で述べたコストを考慮する．サブクエリの出現頻度とは，あるサブクエリが異なるクエリ内に現れる回数である．この回数を O_i とし，O_i にそのサブクエリのコストを乗じたものを利得 e_i とする．e_i が高いサブクエリを有用サブクエリと呼ぶ．

3.3　有用サブクエリの近傍探索

e_i がある閾値 k より高いサブクエリ群を初期解候補 A_i とする．後述する理由により，次に A_i の近傍を探索し，有用な解候補を追加する．ここでサブクエリの近傍とは，そのサブクエリが含まれていた実行プランにおけるサブクエリの親または子を根とする部分木のことである．(図 2)

ある有用サブクエリ a の子サブクエリ b は a より得られる利得は小さいが，出現頻度は a より多い．逆に，親サブクエリ c は a より得られる利得は大きく，出現頻度は a より少なくなる．従って，サブクエリ b，c は利得が高くなる．このような理由から，有用サブクエリの近傍を有用なサブクエリだと推測して解候補に追加する．

[*1] 本稿では s_i のコスト c_i は s_i に対し，PostgreSQL の EXPLAIN ANALYZE コマンド https://www.postgresql.jp/document/11/html/using-explain.html を実行した際のコスト c_i を導入する．

3.4　整数計画問題の繰り返し

得られた解候補に対し，整数計画問題を実行する．得られた解の合計利得が一つ前の整数計画問題で得られた節約コストより大きいならば前項の近傍探索を現在の解候補に対して行い，再び整数計画問題を実行する．これにより，新たな合計利得を得る．あるサブクエリの近傍にはより有用な解の集合が見つけられない場合でも，もう一つ先の近傍に有用な解の集合が存在していることが考えられる．よって，$n-1$ 回目の処理で得られた合計利得が n 回目の処理で得られた合計利得以下で，$n+1$ の処理で得られた合計利得も $n-1$ での利得以下の時，$n-1$ 回目の解をアルゴリズム全体での解として反復を終了する．

この方法により，巨大な解候補空間に整数計画問題を実行するのではなく，有用な解候補のみに整数計画問題を実行して，整数計画問題の実行時間を大きく低減することができる．

4　実験計画

以下の 2 つの項目を評価する．

· **高速性**:2.2 節の整数計画問題を解ける程度のワークロードに対し，提案手法と整数計画問題の応答速度の差を評価する．大規模なワークロードに対しても許容できる応答速度であることを確認する．

· **正確性**:2.2 節の整数計画問題 (1) を解ける程度のワークロードに対し提案手法と整数計画問題 (1) で比較して最終的に得たクエリの性能差が小さいことを確認する．

5　関連研究

データベース上のクエリ処理を高速化するために適切な実体化ビューを選択する研究はいくつか行われている．[1][2][?]．BIGSUBs[1] では二部グラフのラベリング問題を用いることで，整数計画問題を細分化し，確率的なアルゴリズムを導入している．このアルゴリズムではそれぞれの反復で二つのステップがある．一つ目のステップではあるビューが実体化されるべきかどうかを確率的に判断する．この確率は (1) そのビューのがどれほどディスクの容量を占めるか，(2) そのビューによる利得の二つにより計算される．2 つ目のステップでは小さな整数計画問題を解く．これらの反復は閾値に到達するまで行われる．この処理により実体化ビューを選択する．Wide-deep[2] 実体化するサブクエリのコストも課題とし，コスト推定モデルを実装し，ビュー選択問題を整数計画問題とし，強化学習を用いることにより，実体化ビューを選択する．

6　おわりに

提案手法はワークロード内のサブクエリの出現頻度とそのコストを考慮して解候補を削減し，整数計画問題を近傍探索を行いながら繰り返し解くという手法で処理時間の低減を目指す．

7　謝辞

この成果は、国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の助成事業の結果得られたものです。

参考文献

[1] Alekh Jindal, Konstantions Karanasos, Siram Rao, and Hiren Patel. Selecting subexpressions to materialize at datacenter scale. PVLDB, 11(7):800-812, 2018.

[2] Haitao Yuan,Guoliang Li,Ling Feng,Ji Sun,and Yue Han. Automatic View Generation with Deep Learning and Reinforcement Learning.IEEE,1501-1512,2020.