5S-6

CC-Optimizer: キャッシュを考慮した問合せ最適化器

辻 良繁* 川島 英之† 今井 倫太‡

慶應義塾大学 理工学部§

1 はじめに

キャッシュアクセスコストとメモリアクセスコストの差は 10 倍以上あるため [2]，キャッシュミスを削減するという研究テーマはここ 10 年ほど扱われてきた [1] が，L1 命令キャッシュを考慮した研究はそれほど行なわれて来なかった．そのなかで，現実の RDBMS を用いた実践的な研究を行なったものに，PostgreSQL を用いた Zhou らのバッファリングオペレータの研究 [2] がある．

RDBMS がクエリ処理に必要とするオペレータの総サイズは，L1 命令キャッシュより大きい状況があり，この時に L1 命令キャッシュミスが発生し，RDBMS の性能が下げられてしまう．この問題を解決する為に Zhou らはバッファリングオペレータを提案した．これは，小フットプリントで実装可能でオペレータをバッファリング可能なオペレータである．これによりオペレータの実行順序を変更することで，L1 命令キャッシュミスを減らすことができる．文献 [2] によれば，PostgreSQL を用いた実験においてバッファリングオペレータは命令キャッシュミス回数を 80%程度削減し，クエリ性能を 15%ほど改善したとある．そのためバッファリングオペレータは有用な技術であると考えられる．

しかし，同技術を実用的な観点から見た時には，問題がある．その問題とは，同技法がクエリ処理の流れに組み込まれていないことである．バッファリングオペレータは条件によっては性能向上に働くが，条件によっては性能劣化に働いてしまう．そのため，バッファリングオペレータ使用を判断する機構が必要になる．上記課題を解決するために，我々はまず Zhou らの技術の再実装と実験的解析を行い，次に最適化器アルゴリズムの設計および実装を行う．

すなわち，本論文の貢献は，バッファリングオペレータの追試評価と最適化器の開発にある．

2 再実装と実験的解析

2.1 再実装

命令キャッシュの追い出しを防ぐため，Zhou らの論文を参考にバッファリングオペレータの再実装を行った．実装は PostgreSQL-7.3.16 上で行った．

バッファリングオペレータは既存のオペレータへ変更を加えないよう，独立した新しいオペレータを追加する形で実装した．追加処理は，最適化器によりプラン木の適切な位置にバッファリングを指示するノードを挿入し，

*Yoshishige Tsuji
†Hideyuki Kawashima
‡Michita Imai
§Faculty of Science and Technology, Keio University

```
select count(*), avg(p.point),avg(p.id),avg(n.id/2)
from points p, names n where p.id < 5000000 and
p.id = n.id and p.point >= 0
```

図 1 結合演算

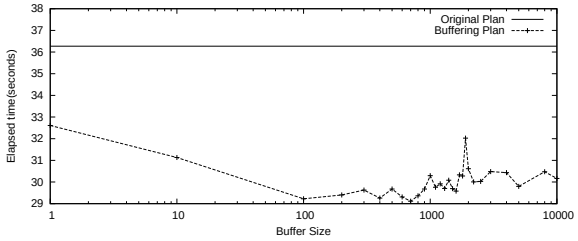

図 2 結合演算におけるバッファサイズを変化させた場合の実行時間比較

エグゼキュータにより実行される形式を採った．このため，エグゼキュータに対して，バッファリングオペレータと既存のオペレータを同様に扱えるよう最小の変更を行った．

2.2 実験的解析

2.2.1 バッファサイズの影響

ここでは，バッファリングオペレータがバッファリングできるオペレータの数をバッファサイズと呼ぶ．

バッファリングオペレータはバッファサイズを増やすほど，L1 命令キャッシュへ命令をロードする回数を減少させ，L1 命令キャッシュミス回数減少からクエリ実効時間の改善が期待できる．一方，バッファリングオペレータによって保持されるポインタの数が多くなれば，L2 データキャッシュミスの回数が増加する．そこで図 1 に示されるクエリについて，バッファサイズと実行時間の関係を求めた． 実験結果を図 2 に示す．図 2 よりバッファサイズ 700 において最大 19.7%の改善が見られた．図 1 のクエリは 3 箇所にバッファリングオペレータを挿入した．したがって，我々の実験条件において，複数のバッファリングオペレータが持つべきバッファサイズの合計は 2000 程度が適当であると考えられる．

2.2.2 濃度の影響

バッファリングオペレータは順序的に実行される 2 つのオペレータの命令サイズ合計が L1 命令キャッシュよりも大きいとき，実行順序を入れ替えることで，L1 命令キャッシュを減らしクエリ処理全体の実行時間を減少させる．一方，バッファリングオペレータの実装は従来の RDBMS に対し，その初期化，終了処理などの余計な処理を必要とする．したがって，子オペレータの実行

select count(*), avg(point), avg(id), avg(id/2) from points where id < (子オペレータの実行回数) and point >= 0

図 3　濃度の影響

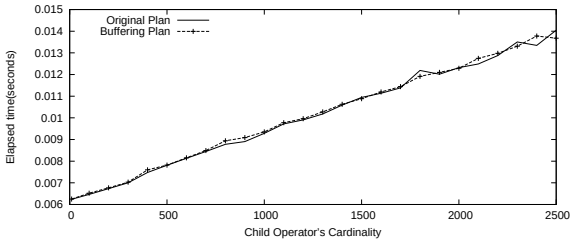

図 4　濃度を変化させた場合の実行時間の比較

回数の変化に応じて，この負荷が償却される閾値を実験によって求めた．実験には図 3 の SQL を使用した． 実験結果を図 4 に示す．実験結果は全体を通して，オリジナルプランとバッファリングプランの実行時間はほぼ同じであった．そこで本論文では，初めてバッファリングプランがオリジナルプランより高速となった 1500 を閾値として採用する．また，Zhou らの実装と異なり，バッファリングオペレータが初期化時に確保するバッファサイズを可変とした結果，子オペレータの実行回数が非常に少ない時でもバッファリングプランは大きな遅延を起こさず，その負荷を非常に小さくできた．

2.2.3　命令サイズ

バッファリングオペレータは順序的に実行される 2 つのオペレータの命令サイズが L1 命令サイズを超える場合に実行時間を改善するため，その場合にはバッファリングオペレータを挟み込むことで実行時間の改善を期待できるオペレータを推定することができる．

実験の結果，比較的命令サイズの大きい avg 関数，sum 関数を含む 3 種類以上の集約関数，結合，およびソート演算が使用される時，バッファリングプランは実行時間の改善を見せた．紙数の限界からここでは詳細を省く．

3　CC-Optimizer

バッファリングオペレータは優れた性能向上手法であるが，性能劣化を招く場合が存在する．本節ではバッファリングオペレータが有効である場合のみ選択を行う最適化器のアルゴリズムと実装を示す．

3.1　アルゴリズムの提案

バッファリングオペレータが有効である場合の判別は次の 5 条件により決定可能である：(1) 集約演算数，(2) 結合演算の有無，(3) ソート演算の有無，(4) バッファの L2 データキャッシュへの収容可能性，(5) 処理タプル数．

本研究では，CC-Optimizer はプラン木を辿り，下記 4 つの場合において，バッファリングオペレータを挿入をする．この条件は前節で得られた解析結果をまとめたものである．

1. 親ノードが 3 種類以上の集約関数を含み，子ノードがソート演算ではない主要モジュールである場合
2. 結合演算を含む場合で子ノードがソート演算ではない主要モジュールである場合
3. ソート演算であり，子ノードが主要モジュールである場合
4. 上記 3 条件のいずれかを満たし，かつ，想定される処理タプル数が 1500 以上である場合

上記条件を満たすバッファリングオペレータの挿入箇所が複数存在する場合，CC-Optimizer は，L2 データキャッシュへの収容可能性を考慮し，バッファサイズの合計が 2000 になるよう調整する．

3.2　最適化器の実装

PostgreSQL 内部において，プラン木は各オペレータと対応するノードのリスト構造で実装されている．CC-Optimizer の実装は完成されたプラン木をルートノードから再帰的に一つずつ辿り，バッファオペレータが有効と判断される場合にリスト操作によって，プラン木へバッファノードを追加する．CC-Optimizer は，PostgreSQL に既存の最適化器に変更を加えないよう，その最後に実行されるよう配置した．

4　結論

本論文では Zhou らの技法をも含むクエリ処理計画を選択可能な最適化器である，CC-Optimizer を実現した．CC-Optimizer の実現に際しては，最適化器がバッファリングオペレータを選択すべき状況を判断するアルゴリズムの新規提案と，同アルゴリズムの RDBMS への実装を行った．

実験用 RDBMS には Zhou らと同様に PostgreSQL を用い，Linux Kernel 2.6.15，CPU Intel Pentium 4(2.40GHz) なる条件において実験を行った結果，CC-Optimizer を用いた RDBMS は既存 RDBMS に対して性能向上を示した．性能向上率は，集約演算を含む問合せについては最大 3.2%，結合演算を含む問合せについては最大 19.7% だった．

今後の課題は，提案アルゴリズムの評価および，マシン環境に応じて動的にアルゴリズムを変更する手法を考案することにある．

参考文献

[1] A. Shadtal, C. Kant, and J.F. Naughton. Cache conscious algorithms for relational query processing. In *Proc. of VLDB Conference*, pp. 510–521, 1994.

[2] J. Zhou and K. A. Ross. Buffering database operations for enhanced instruction cache performance. In *Proc. of ACM SIGMOD Conference*, pp. 191–202, 2004.