4Q-6

動的処理バケット選択方式における Zipf-like 分布データに対する結合演算性能評価

中山雅哉　　喜連川優　　高木幹雄

(東京大学生産技術研究所)

1. はじめに

関係データベースシステムにおいて、結合演算が他の演算に比べて処理負荷が重いことは知られており、特に大規模なデータベースを扱う場合にはネストループ方式やソートマージ方式では高速に演算処理を施すことができないことから、ハッシュ操作を用いた各種の結合演算処理方式が提案されてきた[2,1]。

このうち、我々の提案する『動的処理バケット選択方式』では、従来の GRACE ハッシュ方式[2]にハイブリッドハッシュ方式[1]で採られている分割フェーズと結合フェーズのオーバーラップ処理機構を融合することで、データ分布が不均一な場合でもほとんど処理性能に影響を及ぼさないことが文献[3]等に示されている。

これまでは、解析結果との比較が容易な三角分布を不均一分布例として取り上げてきたが、これは実在するデータベースに即した分布であるとは言い難い。これに対して本稿では、[4]の標準ベンチマークで用いられる Zipf-like 分布による結合演算処理性能の評価結果について報告し、分割バケット数の決定方式についてまとめている。

2. 本稿で用いる用語の定義

結合演算処理を施す2つの対象リレーションを R 及び S とし、結果リレーションを RES と表す。また、リレーション R の保持するタプル数を$\{R\}$で、ページ数を$|R|$で表現し、$\{R\} \leq \{S\}$ となるように R と S を定める。

動的処理バケット選択方式では、GRACE ハッシュ結合方式や、ハイブリッドハッシュ結合方式と同様に、「スプリット関数」を用いて各リレーションを部分空間(バケット)に分割し、各バケット毎に「ハッシュ関数」を用いた結合演算処理を行う2フェーズの処理アルゴリズムをとり、前者を分割フェーズ、後者を結合フェーズと呼ぶ(図1)。

リレーション R を分割処理した際の各バケットはR_iで表現し、各バケットのタプル数やページ数は、先に述べたように$\{R_i\}$、$|R_i|$で表す。また、各バケットR_i(及びS_i)から生成される結果リレーションは、RES_iと表記する。

結合演算処理には、各バケットのステージング用に$|M|$ページとリレーションの入力・出力用に各1ページずつの合計$|M|+2$ページの主記憶が用意されていると仮定し、分割バケット数Hは、$2 \sim |M|$の範囲の値をとることができる。リレーション分割時に生ずる主記憶サイズを越えるバケット(あふれバケット)は、主記憶サイズ以下になるまで再帰的に分割処理を施して結合演算処理を実行する方法をとっている。

[0]Performance Evaluation of the Dynamic Hybrid GRACE Hash Join Method for the Zipf-like distribution data
Masaya NAKAYAMA, Masaru KITSUREGAWA, Mikio TAKAGI
Institute of Industrial Science, University of Tokyo

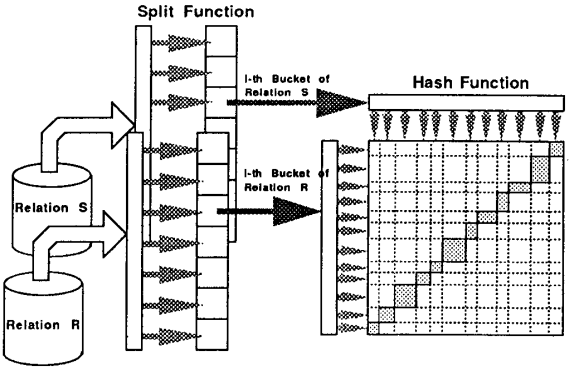

図1: 動的処理バケット選択方式の処理アルゴリズム

ハイブリッドハッシュ方式では、分割フェーズにオーバーラップして結合演算処理を施すバケット(R_1バケット)を用意することで入出力コストの削減を図っているが、このバケットは結合演算処理に先だって静的に決定されている。これに対して我々の方式では、各バケットのデータ分布を基にして動的にオーバーラップ処理バケットを選択する方法をとっており、種々の不均一分布データに対して柔軟に対応することが可能となる[3]。

結合属性値に重複がない場合は、結合演算処理に要するコストの大半はディスクとの入出力コストで占められることになる。動的処理バケット選択方式における入出力コストは、その処理アルゴリズムから次式で表現されることになる。

$$COST(R,S) = |R| + |S| + \sum_{k=1}^{t} |RES_k| + \sum_{k=t+1}^{H_s} (|R_k| + |S_k|) + \sum_{k=t+1}^{H_s} COST(R_k, S_k) \quad (1)$$

ここでバケット番号kは、スプリット関数値に基づくバケット番号iとは異なり、バケットのサイズに基づいて論理的に割り当てられたもので、t番目のバケットまでが、分割フェーズにオーバーラップして結合演算処理を施すバケットに相当する。

3. Zipf-like 分布に従うデータ分布特性

Zipf 分布は、ある対象の出現確率がその出現ランクに反比例する分布のことで、自然言語で書かれたテキストにおける各単語の出現確率がこの分布に従うことが G. K. Zipf 氏により示された[4]。Zipf-like 分布は、これを一般化して各対象の出現確率f_i'がランクiのz乗に反比例する分布として定義される。

$$i^z \times f_i' = \frac{1}{constant^{(z)}} \quad (1 \leq i \leq n)$$

$$constant^{(z)} = \sum_{j=1}^{n} \frac{1}{j^z}$$

z は衰退率と呼ばれ、$z = 0$ の時は一様分布を、$z = 1$ の時はZipf分布を表している。この他にもZipf氏は、個人の収入の分布が$z = 0.5$ の分布に相当することを示している。

本稿では、各バケットのデータ分布がZipf-like分布に従う場合の性能評価を行い、実際のデータベースにおける不均一データ分布に対する動的処理バケット選択方式の有効性についての評価を行っている。

4. Zipf-like分布に対する結合演算処理性能

本章では、Zipf-like分布をとる場合の結合演算処理性能についてまとめ、分割バケット数H_sの決定方法について考察する。性能評価に用いた各パラメータの値は以下の通りである。

結合演算処理に用いる主記憶サイズは、$|M| = 1000$ページで固定とし、各ページには32タプル分のデータが格納できるとする。また、簡単のため$\{R\} = \{S\} = \{RES\}$となる結合演算を仮定し、対象リレーションが100k, 200k,...,1Mタプルとなる場合について性能評価を行う。また、衰退率zについては、実際に分布に意味がある3種類の値 (0.0, 0.5, 1.0) を用いている。

この環境では、入出力コストが結合演算処理性能を左右することになる為、(1)式に基づくシミュレーションにより結合演算処理に必要な入出力回数を測定することで、結合演算処理性能として評価を行っている。

$z = 0.5$ において、分割バケット数H_sを2 ～ 999 まで変化させた時の動的処理バケット選択方式の処理性能は、図2の様になる。ここで、図中の白抜きの点は、同じ分布をとる場合のハイブリッドハッシュ方式に対する処理性能を示している。

この図から、分割バケット数が小さい場合には、あふれバケットの再帰分割処理に伴うオーバーヘッドで処理性能が著しく低下するのに対して、分割バケット数を大きい場合は、ほぼ一定の処理性能となることがわかる。ハイブリッドハッシュ方式は、均一なデータ分布を仮定して分割バケット数を決定している為、この様に不均一なデータ分布に対する処理性能は低いものとなる。

図3には、z値を変化させた時の各方式の処理性能を示している。ハイブリッドハッシュ方式では、均一なデータ分布以外では処理性能が低下して、$z = 0.5$ で約1.2倍、$z = 1.0$ で

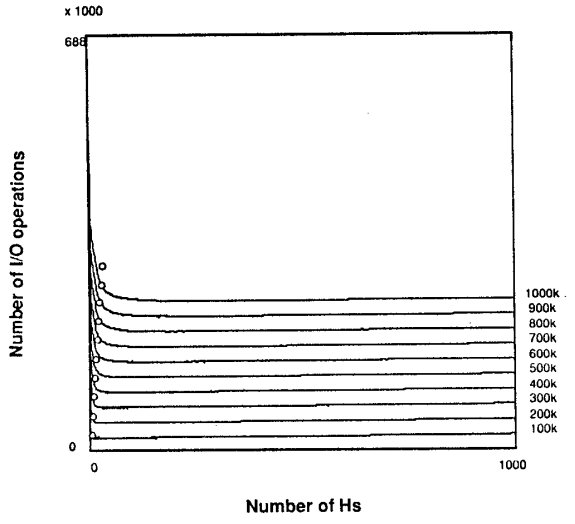

図2: 分割バケット数の変化に伴う処理性能 ($z = 0.5$)

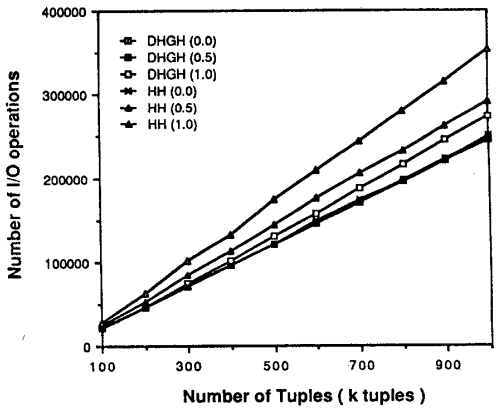

図3: z値変化に伴う各方式の処理性能

約1.5倍のコストを必要とする。これに対して動的処理バケット選択方式では、$z = 1.0$ の場合に大規模なリレーションを扱う場合の処理コストが約1.1倍必要になるものの、それ以外では、ほぼ最小コストで演算処理を実行できる。

このように、対象リレーションのサイズによって分割バケット数を決定する方法では、データ分布が不均一になる場合の性能低下が大きくなるため、実際に分割バケット数を定める際には、予め多くのバケットに分割することが好ましいことが分る。

5. まとめ

本稿では、分割後の各バケットがZipf-like分布をとる場合の動的処理バケット選択方式における結合演算処理性能を分割バケット数H_sや、衰退率zを変化させて測定し、不均一なデータ分布に対する分割バケット数の決定方法について考察を行った。

この結果、衰退率z を一定にした場合の処理性能は、大規模なリレーションを演算対象とする限り、分割バケット数H_s を多くしても処理性能はほぼ一定となることが明らかとなった。また、z を変化させても我々の方法では処理性能の低下を低く抑えることが可能となっている。これに対して、ハイブリッドハッシュ方式で用いる分割バケット数では、不均一なデータ分布に対して、あふれバケットが生成される為、バケットの再分割処理に伴うオーバーヘッドから処理効率が大きく低下する。

このように、分割バケット数を予め多くとることで、未然にあふれバケットの生成を防ぐ方法が一般のデータ分布をとる場合にも有効であることが示されている。

参考文献

[1] D. J. DeWitt and et. al. "Implementation Techniques for Main Memory Database Systems". In *ACM SIGMOD '84*, pp. 1-8, 1984.

[2] M. Kitsuregawa, H. Tanaka, and T. Moto-oka. "Application of Hash to Data Base Machine and Its Architecture". *New Generation Computing*, Vol. 1, No. 1, pp. 66-74, 1983.

[3] M. Nakayama, M. Kitsuregawa, and M. Takagi. "Hash-Partitioned Join Method Using Dynamic Destaging Strategy". In *Proc. of the 14th Int. Conf. on VLDB*, pp. 468-478, 1988.

[4] C. Turbyfill. *"Comparative Benchmarking of Relational Database Systems"*. PhD thesis, Cornell University, September 1987.