大規模オブジェクト集合を対象とするハッシュ結合演算方式

1G－1

安村 義孝
NEC C&C 研究所

1 はじめに

様々なデータ構造を扱えるオブジェクト指向データベース管理システム(OODBMS)では、エクステントを含めた任意個のオブジェクト集合を実行時に定義することができるので、それらの間の結合演算は関係データベース管理システム(RDBMS)よりも多様で複雑であると考えられる。結合演算はアルゴリズムや要素数により膨大な処理時間がかかってしまうことがあるため、その実装には細心の注意が必要となる。本稿では、大規模オブジェクト集合を対象とする結合演算の高速化技法の一方式としてハッシュ結合演算方式について述べる。GRACEハッシュ結合演算方式に基づき、演算中のメモリ利用効率やディスク入出力を考慮して、中間集合管理にオブジェクト識別子(OID)を利用し、バケットサイズを動的に調整する機能を持つことを特徴とする。本方式による結合演算はOODBMS PERCIO上に実装済みであり、その性能評価についても述べる。

2 OODBにおける結合演算

結合演算はRDBMSの問合せにおいて最も負荷のかかる処理であり、高速化のために様々な研究がなされている。一方、OODBMSではオブジェクト間に関連を付け、OIDによるナビゲーションで関連を順に辿ることができ、それらの関連を問合せ文中に記述することも可能である。したがって、関連を付けておけばRDBMSで頻繁に使用される結合演算を避けられるので、目的のデータを取得するためのDBアクセスを高速に行うことができるという特徴がある。しかし、あらゆる問合せ条件を想定して関連を付けるのは非現実的であるので、アプリケーションによってはOODBMSでも結合演算を行う必要がある[3]。

結合演算の例としてネットワーク管理システム(NMS)における管理情報ベース(MIB)を挙げる。OSI管理では管理オブジェクト(MO)の属性データ型にASN.1を利用した複雑なデータ構造を定義可能であるので、MOを管理するMIBの実現にOODBMSを適用することは有効である。NMSのモデル化の一例として、MIB内に管理するデータ構造は、ディレクトリ情報となるMOを一意に識別する識別名(DN)を相対識別名(RDN)によって管理する包含木情報クラスと、装置情報や論理情報となるMO自身である管理対象クラスに分離し、それらのインスタンス間に1対1の関連が付けられているとする。包含関係はRDNの親子間の集合関係として保持し、それらの関係をリンクで結ぶことによりDNを表現する。

ここで、あるDN(DN1)を基点として、自分を含むその上位のRDNが保持する下位のRDNと同じ名前を持つ、別のRDN (RDN2)の下位のRDNを検索することを考える。PERCIOのDMLであるPERCIO/C++での記述例は、

```
Od_List<相対識別名 *> DN1; 相対識別名 * RDN2;
Od_Iterator<相対識別名 *> itr = DN1.create_iterator();
相対識別名 * RDN1; itr.last(RDN1);
Od_Collection<相対識別名 *>* res = RDN2-> 下位集合.
    select([this: R2; &RDN1-> 上位 -> 下位集合: R1;
            where strcmp(R1-> 名前, R2-> 名前) == 0;]);
```

のようになる。このような問合せは任意のRDNに対して実行される。また、RDNの下位集合の要素は数千から数万個存在することがある。全てのメンバ変数にインデックスを付けることは物理的に不可能であるので、上記のような結合演算の高速化が必要となる。

3 ハッシュ結合演算方式

PERCIOでは宣言的な問合せが可能であり、複合条件による検索文や入れ子、関数呼出、結合演算などを記述することができる。これらの条件文は問合せ処理系で解析され、統計情報を利用した最適化を行い、検索アルゴリズムを選択する。ハッシュ結合演算も検索アルゴリズムの1つであり、その他の結合演算にネストループ結合演算、ソートマージ結合演算、インデックスを利用した結合演算が用意されている。

ハッシュを用いる結合演算には様々な方式が提案されているが、それらの中でもGRACEハッシュ結合演算は実装が容易であり性能も優れている。従来の研究ではデータ分布が一様であると仮定して評価を行っているものが多いが、実際にはそのような仮定は不可能であり、メモリ利用効率を十分に考慮しなければならない。そこで、極端に結合キー値が不均一な場合でも比較的安定した性能が得られる多分割方式とした。また、ディスク入出力回数を減らすために、ビットフィルタとバケットサイズの調整(bucket size tuning)[2]の機能を導入する。

GRACEハッシュ結合演算のアルゴリズムは分割フェーズと結合フェーズの2フェーズから構成される。ここでは、集合Rと集合S ($|R| < |S|$)との結合演算を考える。分割フェーズでは、まずRの全要素に対して分割ハッシュ関数を施し、バケットに分割して中間集合R'を生成する。各バケットが利用するステージングバッファ上のページは動的に確保する。中間集合に格納するデータは処理対象のOIDと結合キー値(可変長サイズも可)のみである。その際、分割ハッシュ関数により生成されたハッシュ値を利用してビットフィルタに登録し、該当するビットを立てておく。

次に、Sの全要素に対しても同様の処理を行い中間集合S'を生成する。ビットフィルタに関してはハッシュ値で照合して、該当ビットが立っていなければその要素を処理対象から外す。また、溢れバケットを未然に防ぐた

Hash-based Join Processing for Large Object Collections
Yoshitaka Yasumura
C&C Research Laboratories, NEC Corporation

めに分割するバケット数を多数(ステージングバッファのページ数分)にしているが、中間集合に格納する前にバケットをまとめ上げてフラグメントページを減らすようにしている。

結合フェーズでは、R' と S' の分割された各バケット毎に処理が進められる。R' の任意のバケット R'_i 内の要素に対して結合ハッシュ関数を施し、さらにバケットに分割してステージングバッファに展開しておく。分割フェーズで多分割されたバケットをまとめ上げているので、溢れバケットはほとんど生成されない。次に、そのバケットに対応する S' のバケット S'_i 内の各要素に結合ハッシュ関数を施し、同一のハッシュ値を持つバケットの要素と結合キー値を比較し、等しければ結合処理を行い中間結果集合 IM に OID を出力する。以上の処理を全バケットに対して行い、最終結果集合 RES のために、中間結果集合内の OID のソート処理により一括して B 木を生成する。

また、問合せ処理系では統計情報等に基づいて検索アルゴリズムを選択しているが、統計情報は必ずしも正確であるとは限らず、集合内の要素数を演算実行前に知ることができない。要素が少ない場合には 2 フェーズにすることによるディスク入出力が無駄になってしまう。そこで、分割フェーズで R の要素が全てステージングバッファ(メモリ)上に載り、溢れバケットが発生しなかった場合には、メモリ上の R の要素をそのまま使って、単純ハッシュ結合演算によって結合処理を行う。

4 性能評価

前節で述べたハッシュ結合演算の性能測定のために、ウィスコンシンベンチマークのスケーラブルな関係 [1] を用いた。要素数が 10 対 1 のエクステント集合に対する単純結合がある選択演算であり、結果集合の全要素にアクセスするという測定内容で、選択率が 0.1 になるように結合キーを選んで測定を行った。使用したマシンはメモリを 48M バイト持つ EWS4800/330 であり、キャッシュサイズは 8M バイト、演算領域(ハッシュのステージングやソートのためのキャッシュ内に確保される領域)は 1M バイトである。

測定結果を図 1 〜図 4 に示す。ここで、ディスク入出力回数と比較回数は演算実行中に発生するものと最終結果生成のためのソート処理のものを含むが、DB アクセス部分はカウントしていない。この結果より、unique1 (整数、ユニーク、ランダム順) 同士の結合では、分割フェーズで少量の集合 (5K 件〜 50K 件) の全要素がステージングバッファに載るので、ディスク入出力がほとんどないことがわかる。onePercent (整数、0-99、ランダム順) と tenPercent (整数、0-9、ランダム順) の結合では、ハッシュ分割時にバケットの片寄りができるため、unique1 同士の結合よりも性能が悪くなる。しかし、ページを動的に確保することにより、ディスク入出力回数は同じになるので、比較回数の増加が性能に直接影響している。

unique2 (整数、ユニーク、昇順) 同士の結合では、処理内容が同じである unique1 同士の結合よりも高速である。これは unique2 同士の結合による選択結果がエクステント集合の先頭の方に集中していることによる。したがって、DB 内でのクラスタリングが重要であるといえる。stringu1 (文字列、ユニーク、ランダム順) 同士の結合では、文字列のサイズが 52 バイトなので要素数が多

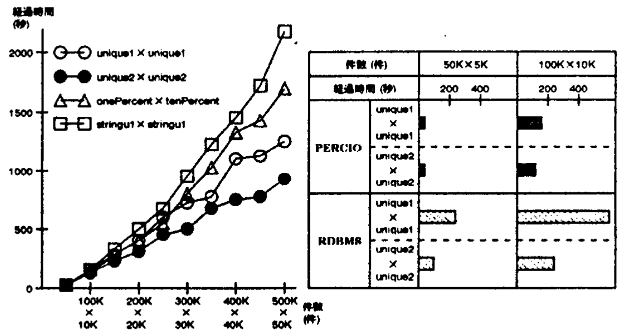

図 1 : 経過時間

図 2 : RDBMSとの比較

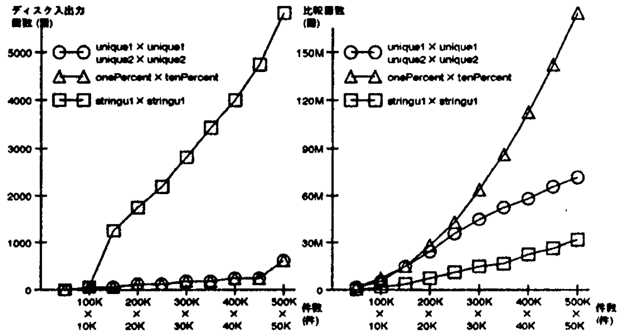

図 3 : ディスク入出力回数

図 4 : 比較回数

いと演算領域に載りきらなく、GRACE ハッシュ結合演算が実行される。そのため、ディスク入出力回数は飛躍的に増加するが、比較回数は unique1 同士の結合よりも少なくなっている。

また、少量データで商用 RDBMS の結合演算 (ソートマージ結合演算) と比較すると、PERCIO の方が数倍性能がよいという結果が得られた。さらに、RDBMS では unique1 同士の結合と unique2 同士の結合とで性能に 2 倍以上の差があるが、PERCIO ではそこまで差がない。これは演算途中で OID を使用していることにより、処理コストを抑えられていることによると思われる。

5 おわりに

本稿では、ハッシュ結合演算方式による OODBMS の結合演算の高速化技法について述べた。また、本方式を実装した PERCIO による性能評価の結果、RDBMS のテーブル間の結合演算よりも高速であることを示した。今後の課題としては、メモリ上における比較回数の削減、ハッシュ結合演算の適用条件の拡大、PERCIO 自身の性能向上がある。

謝辞

PERCIO のアーキテクチャに関して有益な助言をして頂いた NEC C&C 研究所 鶴岡邦敏課長に感謝致します。

参考文献

[1] DeWitt, D.J., "The Wisconsin Benchmark: Past, Present, and Future," in *The Benchmark Handbook, Second Edition*, Gray, J. ed., Morgan Kaufmann, pp. 269–315, 1993.

[2] Kitsuregawa, M., Nakayama, M. and Takagi, M., "The Effect of Bucket Size Tuning in the Dynamic Hybrid GRACE Hash Join Method," *Proc. VLDB*, pp. 257–266, 1989.

[3] Shekita, E.J. and Carey, M.J, "A Performance Evaluation of Pointer-Based Joins," *Proc. ACM SIGMOD*, pp. 300–311, 1990.