2E-2 GRACEに於けるディスクモジュールの構成

伏見 信也† 喜連川 優†† 田中 英彦† 元岡 達†
(†東京大学 工学部 ††東京大学生産技術研究所)

1. はじめに

関係データベースマシンGRACE[1]はプロセッシングモジュール(ハードウェアソータを用いた演算処理)、メモリモジュール(中間記憶)、ディスクモジュール(二次記憶)及びコントロールモジュール(システムの統括、制御)の4種のモジュールから構成され、hashとsortを用いたアルゴリズムによりjoin等の処理負荷の重い演算を高速に実行することが出来る。一方、数GB程度の巨大なリレーションの処理の際には、ディスクモジュールからstaging されるタプルを全てメモリモジュールの記憶空間に収容することは一般に不可能である。本稿では、ディスクモジュール空間を用いてメモリモジュール空間を仮想化し[2]、この様な巨大なリレーションを効率良く処理する技法を提案する。

2. GRACEに於けるjoin処理方式

GRACEではjoin演算の対象となるリレーションはディスクモジュールから一旦複数台のメモリモジュールにstaging され(staging phase)、その後メモリモジュールが送出するタプルストリームに沿った演算処理が複数台のプロセッシングモジュールによって並列になされる(processing phase)。staging phase ではjoin演算に関する属性に対してhashing が行なわれ、そのhash値を基にリレーションは当該演算に関して互いに独立な小さなバケットに分割されてメモリモジュールに格納される(動的clustering)。動的clusteringによって、リレーション全体のjoin演算は複数個の小さなバケット内のjoinに還元され、join処理の負荷は大幅に減少する。

3. メモリモジュール空間の仮想化

一般に数GB程度のリレーションの処理に於いては、ディスクモジュール内でのselection 演算等によるタプルのふるい落し処理の後も 1GB～数百MB程度のタプルがメモリモジュールにstaging されてくることが予想され、1join演算に割り当てられたメモリモジュールがこれら全てのタプルを収容することは一般に不可能である。そこで本稿では先ずリレーションを当該演算に割当てられたメモリモジュールの記憶空間容量程度の相互に独立な cluster (staging cluster)に分割し、各staging clusterに対して2.で述べた従来のjoin処理アルゴリズムを適用することにより、この様な巨大なリレーションに対しても効率の良い処理を行なう技法を提案する。このことはメモリモジュールの記憶空間をディスクモジュールの記憶空間を用いて仮想化したことに相当する。動的clusteringの効果から、この仮想化に於いては各バケット内に完全なアクセスローカリティが存在し、逆にバケットは処理に対して相互に独立であり、その処理順序は任意でよい。従ってこの仮想化は実際には動的clusteringによって生成された小さなバケットを集め、メモリモジュールの記憶容量に等しくなる様にstaging cluster を生成、これをstaging することを繰返すことによって実現される。

4. GRACEに於ける仮想化の実現

4.1 概要

前章で述べたメモリモジュール空間の仮想化を実現する為には効率の良いstaging cluster の生成機構をディスクモジュール内に設けることが必要である。ここではベースリレーションを格納しているディスクモジュールとは別に新たに作業用ディスクモジュールを設け、本機構を持たせることとした。従ってjoin演算に対する処理の概要は次の様になる。ベースリレーションはselection 等のon-the-fly処理を受けつつ一旦メモリモジュールに向けて stageされる。メモリモジュールは到着してくるタプルをバケット毎に管理し、溢れが生じた場合には適当なバケットを選んで(4.3参照)そのバケットに属するタプルを作業用ディスクモジュールにdestage することを繰返す。ベースリレーションのstaging が完了した時点でメモリモジュール内には最初に処理されるstaging cluster が生成される。一方、以降のstaging cluster は作業用ディスクモジュールがこれを生成、stage する。この構成によりタプル数N(>>1)の巨大なリレーションに対してほぼ2Nの時間でstaging (従ってjoin演算)を終了することが出来る。

4.2 作業用ディスクモジュールの構成

一般にディスクの入出力は低速であり、staging cluster を効率良く生成する為には作業用ディスクのページ空間に於いてhash値に関するclusteringが必要になる。一方、入出力動作そのものを効率化する為には半導体記憶によるタプルのバッファリング（ディスクキャッシュ）が有効である。ここでこのclusteringを実現しつつ効率の良いディスク入出力を保証しようとすると、ディスクキャッシュとしてバケット数・ページ容量程度の大きさが必要となり、現実的ではない。しかしほぼメモリモジュールの容量に等しいstaging cluster を生成する為には個々のバケットをディスク内で完全にclusteringして保持する必要はなく、いくつかのバケットをまとめてメモリモジュール容量の数十分の１程度の大きさの大まかなバケット（staging subcluster）として管理すれば十分である。この場合ディスクキャッシュのエントリ数はstaging subclusterの数（１００程度）あれば良く、１ページを１トラック（数十ＫＢ）とすればコスト的に妥当と考えられる。以上の考察から得られた作業用ディスクモジュールの構成を図１に示す。ここではstaging subclusterとページアドレスとの対応は半導体記憶中にページテーブルとして保持される。タプルのdestaging 時には転送されてくるタプルはstaging subcluster毎にディスクキャッシュ内にバッファリングされ、ページ容量分タプルが到着したディスクキャッシュエントリは新しく割当てられたトラックに書出される。一方、タプルのstaging 時には、staging すべきsubcluster を上記テーブルから得られる各々の大きさを基に決定し、ディスクキャッシュを読出し用のバッファとして使用し、ページ読出し動作と重畳化しながらこれを実行する。

4.3 destageバケットの選択

メモリモジュールの記憶空間に溢れが生じた時、destage するバケットは一般に任意で良い。一方、メモリモジュールの記憶空間の仮想化を要する程の巨大なリレーションに対しては動的clusteringによって生成される個々のバケットはかなり大きい場合が予想される。この様な場合、プロセッシングモジュール内の記憶空間にバケットを収容出来ず、更にメモリモジュール空間の使用効率も低下することが予想される。そこでメモリモジュールに溢れが生じた際のdestaging バケット選択アルゴリズムとして、その時点で最も大きいバケットを選ぶこととし、初回のstaging 時に生成されるstaging cluster はなるべく小さなバケットから構成する。次回以降の作業用ディスクモジュールからのstaging の際には、大きいバケットに対しては作業用ディスクモジュール内で更に細かいhashを施すことにより、充分なバケットの細分化が得られ、上記二点を回避することが出来る。

5. おわりに

現在、本技法を拡張し、join演算等の実行によって一時的に生成される中間リレーションに対するメモリモジュール・ディスクモジュール間の仮想化技法を検討中である。

参考文献

[1] Kitsuregawa,M. et al. "Application of Hash to Data Base Machine and Its Architecture", New Generation Computing, 1, 1983

[2] Ozkarahan, E., et al. "Analysis of Architecural Features for Enhancing the Performance of a Database Machine", TODS, Vol.2 No.4, 1977

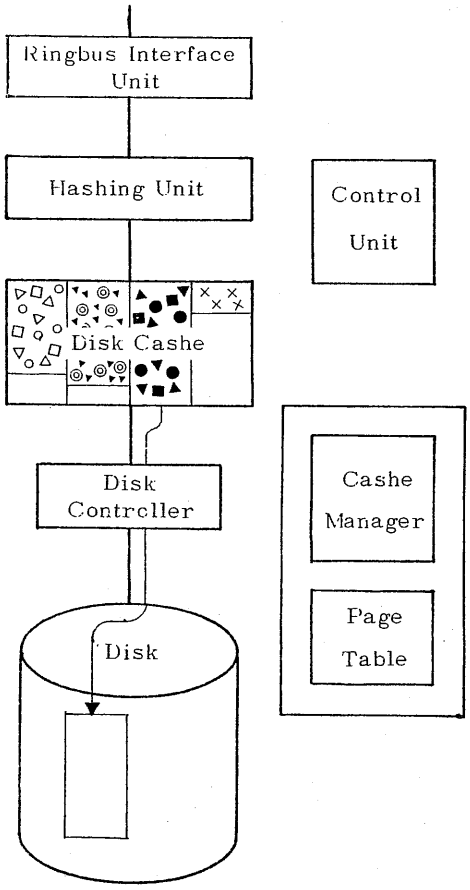

Fig. 1 Architecture of Working Disk Module