4K－6

並列データキューブ計算における動的負荷分散方式に関する考察

武藤 精吾　喜連川 優
東京大学 生産技術研究所

1　はじめに

近年新たなアプリケーションとしてデータベース業界で台頭してきているOLAPでは、多次元のデータに対する高速な処理が要求される。特にデータキューブはOLAPの基本部分として注目されており、効率的な処理方式が研究されている。特に次元数の増加にともない処理負荷は急激に高くなるため並列処理による高速化が考えられるが、単純に並列化するだけではデータ分布の偏りによって各プロセッサ間で負荷の偏りが生じるため、十分な並列効果を得ることは困難である。したがって、プロセッサ間において負荷を均等化させる機構が必要となるが、データキューブの計算では事前に負荷を予測することは困難であるため、静的にデータ分布を均等化するだけでは実行中の負荷の偏りにまで対応することはできない。本稿ではデータキューブの並列計算において処理の実行中に動的に負荷を分散させる方式を取り入れ、その効果について考察を行う。

2　並列データキューブ計算

データキューブ演算はSQLの集約演算を拡張したものであり、各次元アトリビュートのすべての組合せに関して集約演算を行うものである[3]。各集約演算の結果はそれぞれキューボイドと呼ばれる。対象となる集約関数はCOUNTやSUMのように[3]にて分配的性質を持つと定義されている関数であるとする。現在までにソートやハッシュを用いたいくつかのアルゴリズムが提案されているが[1, 2, 4, 5]、我々はその中でも特に並列化に適していると思われるパイプハッシュ[1]を用いる。

パイプハッシュではハッシュを用いてパイプライン方式でキューボイドを計算していく。ハッシュテーブルがメモリに納まらない場合にはデータを適度なサイズに分割することにより処理を行っていく。各キューボイドの実行プランは図1に示すように木構造で表現される。分割を行った際には、分割アトリビュートにより実行木の分割を行われる。並列化に関しては各パーティションを各プロセッサに分配することにより容易に実現することができる。

3　動的負荷分散方式

データキューブの並列処理における負荷の偏りは次の2種類に分類することができる。

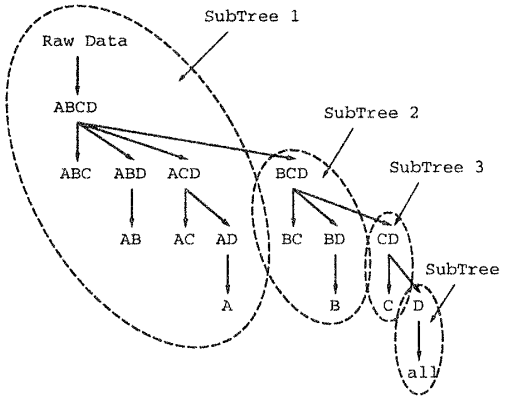

図 1: データキューブ演算の実行木

- PSS(Partition Size Skew)
  データの分割によって生成されたパーティションサイズの分布の偏り
- CSS(Cuboid Size Skew)
  親のキューボイドから生成されていく各キューボイドサイズの分布の偏り

PSSの影響を緩和するためにはデータをパーティションに分割したのち、各パーティションを各プロセッサにおけるパーティションサイズの合計がほぼ等しくなるように再配置するという静的な負荷分散方式が考えられる。しかしながら、CSSの影響により処理を開始した後のプロセッサ間の負荷分布は変化してしまうため、静的に負荷分散をおこうだけでは処理の実行中における負荷の偏りに対応することはできない。十分な並列効果を得るためには実行時において動的に負荷分散を行うことが必要となってくるため、図2に示すアルゴリズムを用いることによりその実現化を図る。図2のアルゴリズムでは、割り当てられた処理の終了したプロセッサに対して他のプロセッサに割り当てられている未処理のパーティションを再割り当てすることにより動的に負荷分散を行っている。特定のプロセッサが各プロセッサにおける処理状況を管理し、動的にパーティションを割り当てていく方式を取っている。

4　シミュレーション結果

図2のアルゴリズムにもとづいてその効果に関して考察を行うためにシミュレーションを行った。最初のパーティション生成時におけるコストはディスクI/Oバウンドであるとし、分割後の処理コストはプロセッサバウンドであると仮定している。ディスクI/Oコストはアクセスされるデータ量と転送速度から算出しており、また、集約演算の処理コストは処理の対象となってい

A Consideration on Dynamic Load Balancing Strategies for Parallel Datacube Computation
Seigo Muto and Masaru Kitsuregawa
Institute of Industrial Science, University of Tokyo
Roppongi 7-22-1, Minato, Tokyo, 106-8558 Japan

```
procedure DynamicParallelCube(cuboid c)
pc = parent cuboid of c;
P = partitions obtained by partitioning pc on some
dimensions;
foreach p ∈ P do
    Allocate p to the processor which has the smallest
    number of tuples already allocated;
end
CC = child cuboids including c of pc whose dimension
     order begins with partitioning dimensions;
OCC = other child cuboids of pc;
UP = partitions to be processed;
while UP ≠ φ do
    foreach processor do
        Compute CC and other child cuboids that can be
        computed from CC;
    end
    if one of the processors completes the computation
    do
        Reallocate a partition to the processor from the
        other one which has the largest number of
        unprocessed partitions;
    end
    UP = partitions unprocessed at this point;
end
while OCC ≠ φ do
    DynamicParallelCube(c ∈ OCC);
end
```

図 2: 動的負荷分散のアルゴリズム

パラメータ	値
次元数	10
タプル数	10^6
濃度	100
ディスク転送速度 (リード)	6.45 (MBytes/sec)
ディスク転送速度 (ライト)	3.5 (MBytes/sec)
集約演算の処理時間 (タプル単位)	7.89 (msec)
アトリビュートサイズ	4 (Byte)

表 1: パラメータ値

るキューボイドのサイズに比例するものとしている。シミュレーションに用いたパラメータ値は表 1のようになっている。PSS や CSS の分布には Zipf-like 分布を用いている。Zipf ファクタは 0 ～ 1 の値を取るものとする。CSS ではプロセッサ間で負荷の偏りが起こるように分布を選んでいる。

図 3、4にプロセッサ数を変化させた時の実行速度を示す。図 3は PSS 分布の Zipf ファクタをを 0 とした時の結果であり、図 4は 1 とした時の結果である。CSS の Zipf ファクタをパラメータとして変化させている。図 3において、静的な分散手法を用いた場合には負荷の偏りが大きくなるにつれて性能が大幅に劣化しているのに対し、動的な分散手法ではほとんど変化していないのがわかる。図 4では動的な手法を用いても並列効果が少なくなっているが、これは PSS の影響によりパーティションを均等に分配するのが難しくなるためである。しかしながら、この場合においても動的な負荷分散の効果はよく表れているのがわかる。

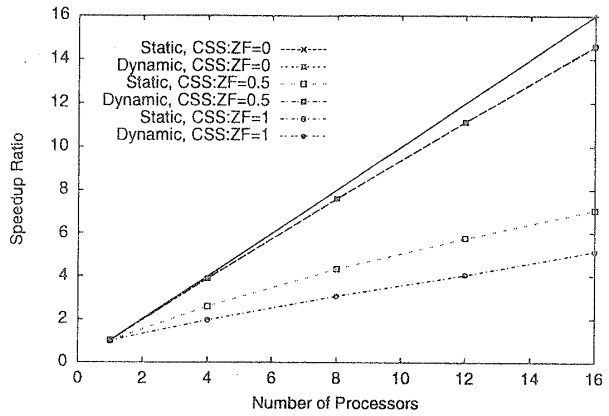

図 3: プロセッサ数に対する実行速度 (PSS:ZF=0)

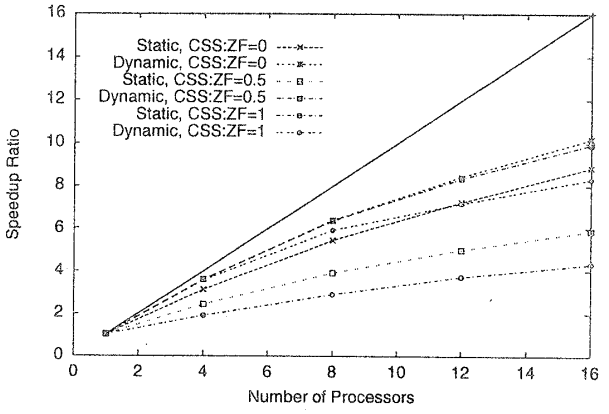

図 4: プロセッサ数に対する実行速度 (PSS:ZF=1)

5 まとめ

本稿ではデータキューブの並列処理においてプロセッサ間の負荷を動的に均等化する手法を導入し、シミュレーションを行うことによりその効果について検討を行った。今回の実験は単純なシミュレーションにもとづいたものであり、実装による性能評価を行っていくのが今後の課題である。

参考文献

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan and S. Sarawagi, "On the Computation of Multidimentional Aggregates", In *Proceedings of the International Conference on Very Large Databases*, pages 506–521, 1996.

[2] K. S. Beyer and R. Ramakrishnan, "Bottom-Up Computation of Sparse and Iceberg CUBEs", In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 359–370, 1999.

[3] J. Gray, A. Bosworth, A. Layman and H. Pirahesh, "A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals", In *Proceedings of the IEEE International Conference on Data Engineering*, pages 152–159, 1996.

[4] K. A. Ross and D. Srivastava, "Fast Computation of Sparse Datacubes", In *Proceedings of the International Conference on Very Large Databases*, pages 116–125, 1997.

[5] Y. Zhao, P. M. Deshpande and J. F. Naughton, "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates", In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 159–170, 1997.