データベースシステム 113－24
（１９９７．７．１５）

共有メモリマルチプロセッサ上での トランスポーズドファイルを用いた 並列関係問合せ処理

武藤精吾、田村孝之、中野美由紀、喜連川優
東京大学 生産技術研究所

概要

本研究は関係データベースにおける非定型問合せ処理の高速化に対してトランスポーズドファイルの有効性を明確化することを目的とする。トランスポーズドファイルを用いるとディスク入出力が減少し、CPU 負荷が増大する。意志決定支援システムなどの大規模なデータにアクセスするアプリケーションではディスク入出力がボトルネックとなるが、それに対して CPU での処理は近年の著しい性能向上や複数の CPU を用いた並列処理などにより高速に行うことが可能である。したがって、トランスポーズドファイルを用いることにより全体の処理性能は向上することが期待できる。共有メモリ型並列計算機において並列問合せ処理を実装し、ベンチマークを用いて性能評価を行う。

Parallel Relational Query Processing Using Transposed Files on Shared Memory Multiprocessors

Seigo Muto, Takayuki Tamura, Miyuki Nakano, Masaru Kitsuregawa
Institute of Industrial Science, University of Tokyo

Abstract

In decision support application the disk I/O load can be very high since ad hoc query processing tends to access very large volumes of data. With transposed files, where each file consists of the values contained in single attribute of a relation, it is possible to avoid reading data which is not needed for query processing, and therefore disk I/O can be reduced. With complex queries however, the use of transposed files results in increased CPU load due to the additional processing required to recompose the relation which causes performance degradations. This thesis discusses how parallel query processing with transposed files should be implemented and clarifies the issues involved in using transposed files.

1 はじめに

意志決定支援システムなどで要求される関係データベースの非定型問合せ処理では、大容量のファイルアクセスが頻繁に行われるためディスク入出力に対する負荷が高く性能向上が望まれている。そのためにはディスク入出力コストの低減が必要であるが、非定型問合せ処理では事前にアクセスパターンを予測することができないために索引などを用いた高速化を行うことができない。そこで本研究ではトランスポーズドファイルというファイル形式を用いることにより性能改善を図る。

従来のリレーション単位でディスクからデータを読み出す手法に対し、トランスポーズドファイルを用いた手法ではアトリビュート単位でデータを読み出すため、問合せに使用されないアトリビュートへのアクセスを避けることが可能となり、ディスク入出力を減少させることができる。その反面、関係データベースにおいて処理負荷の高い結合演算を用いてリレーションを再構成しなければならないため、CPU コストが増大し、問合せによってはかえって性能劣化につながってしまうということがかつては問題となっていた。しかしながら、ディスク入出力性能にくらべ CPU 処理性能の向上は著しく、また並列計算機の実用化により高性能なアルゴリズムを用いた並列結合演算処理が可能となったため、CPU コストが低減され、トランスポーズドファイルによる効果が期待できるようになった。

そこで実際に共有メモリ型の並列計算機上において問合せの処理系を実装し、ベンチマークを用いた性能評価を行うことによりトランスポーズドファイルの有効性を明らかにする。

2 トランスポーズドファイル

関係データベースではデータはリレーションと呼ばれる単位でまとめられており、ファイルへの格納形式もリレーションごとに行うのが普通である。しかしながら、トランスポーズドファイルでは図 1に示すようにリレーションをさらにアトリビュートごとに分割して格納するため、アトリビュート単位でデータにアクセスすることが可能となり、不要なアトリビュートを読み出すことなく問合せを行うことができる [1]。

トランスポーズドファイルでは分割前のデータ間のつながりを保つためにタプル ID(TID) と呼ばれるアトリビュートを新たに付与する。実際の処理ではディスクからデータを読み出したあとにこの TID をもとにしてアトリビュート同士の結合処理を行い、リレーションを再構成する。

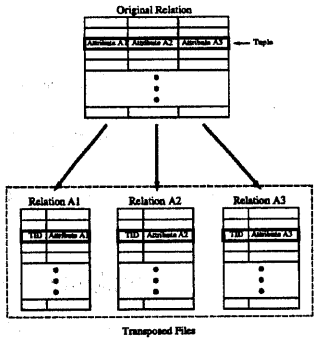

図 1: トランスポーズドファイル

例として、リレーション A、B に対して図 2のような問合せを行うことを考える。リレーション A、B はそれぞれ A1 ～ A6、B1 ～ B6 のアトリビュートからなっているものとする。

```
select A2, B4
from   A, B
where  A1 = B1 and
       A5 < XXX
```

図 2: 問合せの例

通常の手法を用いた場合とトランスポーズドファイルを用いた場合の問合せ処理の様子をそれぞれ図 3、図 4に示す。通常の手法を用いた場合には各リレーションのアトリビュートはすべてディスクから読み込んでいるが、結合演算は 1 回で済んでいる。それに対して、トランスポーズドファイルを用いた場合には問合せで実際に使われるアトリビュートのみを読み出しているが、結合演算は 4 回に増加しているのがわかる。

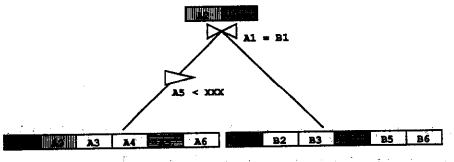

図 3: 通常の場合

3 実装方式

関係データベースでは問合せは演算により構成される。問合せ処理では主に射影演算、選択演算、結

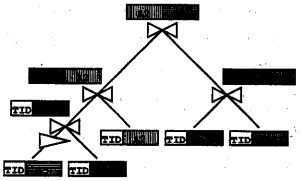

図 4: トランスポーズドファイルを用いた場合

合演算、集約演算がよく用いられる。このうち射影演算、選択演算の実装は単純であるが、結合演算に関しては処理負荷の高い演算であるため、ネストループ方式、ソートマージ方式、ハッシュ方式などさまざまな実装方式が研究されている。今回はその中でもとりわけ高い性能を示しているハッシュ結合演算方式を用いる [2]。なお、集約演算については今回は考慮しないことにする。

3.1 ハッシュ結合演算方式

ハッシュ結合演算方式はビルドフェイズ、プローブフェイズの 2 つのフェイズからなっている。結合演算の対象となる 2 つのリレーションに対して、次のような操作を行う。

ビルドフェイズ

一方のリレーションの各タプルに対して、結合演算の対象となるアトリビュートにハッシュ関数を適用し、それらをハッシュテーブルにエントリさせていく。

プローブフェイズ

もう一方のリレーションの各タプルに対して、結合演算の対象となるアトリビュートにビルドフェイズで用いたものと同様のハッシュ関数を適用し、ハッシュテーブルを検索する。そして条件を満たすタプルが存在すれば結合処理を行う。

3.2 並列ハッシュ結合演算方式

今回の実装では HP 社の分散共有メモリ型並列計算機 Exemplar SPP1200 の 1 ノードを共有メモリ型並列計算機として用いる。CPU は PA-RISC (120MHz)8 台、メモリ(512MB)、ディスク(1GB)1 台からなる。

図 5に示すようにメモリの一部をディスク入出力のためのリードバッファ、ライトバッファに割り当て、残りの部分をハッシュテーブルに用いる。

ハッシュ結合演算方式の大部分は並列化が可能である。

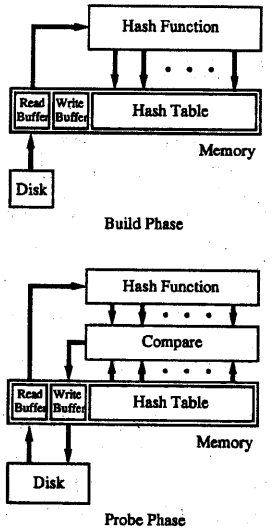

図 5: ハッシュ結合演算方式

ビルドフェイズでの並列化

メモリ上に読み込まれた各タプルに対してハッシュ関数を適用する処理については並列に行うことが可能である。タプルをハッシュテーブルにエントリさせるときには、異なる CPU 同士の同一エントリへの同時書き込みが起こる可能性があるため、排他的な操作が必要である。しかし、一般に CPU 数にくらべハッシュエントリ数は極めて多く、競合が起こる可能性は低い。

プローブフェイズでの並列化

ビルドフェイズと同様にハッシュ関数の適用処理は並列化が可能である。また、ハッシュテーブルの検索、タプルの結合処理も並列に行うことができる。

リードバッファ、ライトバッファに対するアクセスは排他的に行うことが必要である。しかしながら、バッファサイズを大きくすることで競合を緩衝させることが可能である。

ハッシュ結合演算処理はディスクとの読み書きを管理する入出力処理と CPU でハッシュ関数の適用やハッシュテーブルの検索、タプルの比較などを行う結合処理の 2 つに大きく分けることができる。今回はそれぞれの処理ごとに独立したプロセスを用いて実装し、入出力プロセスを 1 台の CPU に割り当て、残りの各 CPU に結合プロセスを割り当てるこ

ととした。

3.3 多重結合演算方式

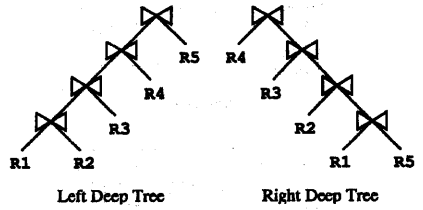

図 6: レフトディープ木とライトディープ木

基本的な多重結合演算方式にはその問合せ木の形状からレフトディープ方式、ライトディープ方式の2つの方式がある(図6)(それ以外の問合せ木はBushy木と呼ばれる) [3]。ハッシュ結合演算方式を用いる場合には、問合せ木は左側がビルドリレーション、右側がプローブリレーションを表す。したがって、レフトディープ方式とライトディープ方式はそれぞれ対象となるn個のリレーションに対して次のように処理が行われる。

レフトディープ
: 2つのリレーションを用いて最初の結合演算を行い、以降は直前の結合演算の結果をビルドしてから残りのリレーションの1つでプローブを行う結合演算を繰り返していく。

ライトディープ
: n-1個のリレーションに対してビルドを行い、複数のハッシュテーブルを作成してから残った1つのリレーションによりプローブを行う。

レフトディープ方式では同時に必要とされるハッシュテーブルは高々2つで済むのでメモリの制約が少ないが、結合率の大きな結合演算が途中に存在すると中間結果がメモリに納まらなくなってしまう。それに対してライトディープ方式ではリレーションの数が多いときには多数のハッシュテーブルを同時に生成しなければならないためメモリに対する制約が大きいが、レフトディープ方式のように中間結果の管理を必要としない。そのため、通常の結合演算はライトディープ方式を用いて行った。

トランスポーズドファイルを用いた多重結合演算では通常の結合演算に加えてTIDに関する結合演算も行わなければならないが、TIDによる結合演算では結果のタプル数が元のリレーションよりも多くならないという特徴がある。しかも、通常結合演算を行う前に選択演算によるタプルの絞り込みがあるため、実際には結果のタプル数はかなり少なくなることが予想される。レフトディープ方式では中間結果を次の結合演算のビルドフェイズで用いるため、タプル数が絞り込まれるとその分ビルドフェイズの処理負荷が減少する。また、中間結果のタプル数が少なければメモリからあふれることもない。したがって、TIDをキーとする結合演算に関してはレフトディープ方式を用いた。

4 性能評価

性能評価は意志決定支援システム用のベンチマークであるTPC-Dを用いて行った[5]。全部で17の問合せが提供されており、今回はそのうち結合演算数がそれぞれ0、2、5である問合せ6(図7)、3(図8)、5(図9)についてスケールファクタ0.5で測定を行った。各リレーションのサイズを表1に示す。

```
select sum(l_extendedprice * l_discount)
from   lineitem
where  l_shipdate >= date '1994-01-01'
   and l_shipdate <  date '1995-01-01'
   and l_discount >= 0.05
   and l_discount <= 0.07
   and l_quantity <  24
```

図 7: TPC-D 問合せ 6

```
select l_orderkey,
       sum(l_extendedprice
           * (1 - l_discount)),
       o_orderdate,
       o_shippriority
from   customer, order, lineitem
where  c_mktsegment = 'BUILDING'
   and c_custkey    = o_custkey
   and l_orderkey   = o_orderkey
   and o_orderdate < date '1995-03-15'
   and o_shipdate  > date '1995-03-15'
group  by l_orderkey,
          o_orderdate,
          o_shippriority
order  by 2 desc, o_orderdate
```

図 8: TPC-D 問合せ 3

選択演算、結合演算に用いられるアトリビュートのデータ型にはすべて整数を用いてシミュレーショ

```
select n_name,
       sum(l_extendedprice
           * (1 - l_discount))
from   cutomer, order, lineitem
       supplier, nation, region
where  o_custkey    = c_custkey
   and o_orderkey   = l_orderkey
   and l_suppkey    = s_suppkey
   and c_nationkey  = s_nationkey
   and s_nationkey  = n_nationkey
   and n_regionkey  = r_regionkey
   and r_name       = 'ASIA'
   and o_orderdate >= date '1994-01-01'
   and o_orderdate <  date '1995-01-01'
group  by n_name
order  by 2 desc
```

図 9: TPC-D 問合せ 5

リレーション	タプル数	タプル長 [バイト]	サイズ [バイト]
part	100,000	164	16,400,000
supplier	5,000	197	985,000
partsupp	400,000	219	87,600,000
customer	75,000	223	16,725,000
order	750,000	134	100,500,000
lineitem	2,999,671	141	422,953,611
nation	25	185	4,625
region	5	181	905
合計	4,329,701	1,444	645,169,141

表 1: TPC-D のリレーション

ンを行った。また、通常集約演算により出力は小さくなるが、今回は集約演算を行っていないため出力結果のディスクへの書き込みは行わないようにした。

表 2に各問合せにおいてトランスポーズドファイルを用いることによるディスク入出力量と結合演算数の変化を示す。各問合せともディスク入出力量は減少し、結合演算数は増加しているのがわかる。

問合せ	ディスク入出力量		結合演算数	
	NTP	TP	NTP	TP
Q6	403MB	126MB	0	4
Q3	515MB	139MB	2	9
Q5	516MB	133MB	5	15

表 2: ディスク入出力量と結合演算数の変化

トランスポーズドファイルを用いて行った各問合せの問合せ木は図 10〜 12のようになる。最初に TID に関する結合演算をレフトディープ方式で行い、リレーションが再構成されたところで通常の結合演算をライトディープ方式で行った。また、通常の手法を用いた場合はライトディープ方式のみを用いている。

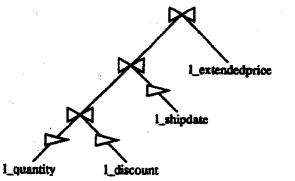

図 10: TPC-D 問合せ 6 の問合せ木

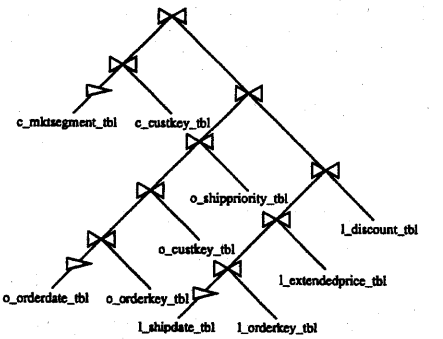

図 11: TPC-D 問合せ 3 の問合せ木

図 13〜 15に各問合せにおいて結合プロセス数を変化させたときの実行時間の様子を示す。図で「Nontransposed」と示されているのが通常の手法を用いた場合であり、「Transposed」と示されているのがトランスポーズドファイルを用いた場合である。「I/O Time(Transposed)」と示されているのはトランスポーズドファイルを用いた場合におけるディスク入出力時間である。

図 13〜 15から通常の手法を用いた場合ではどの問合せにおいても入出力バウンドとなっていることがわかる。トランスポーズドファイルを用いた場合には、どの問合せにおいても CPU バウンドとなっているが、問合せ 6 については結合演算数が少ないため、結合プロセス数が多くなるとほぼ入出力バウンドになっている。両手法による実行時間を比較してみると、結合演算数の少ない問合せ 6 では結合プロセス数に関係なく常にトランスポーズドファイルを用いた場合の方が良い性能を示している。結合演

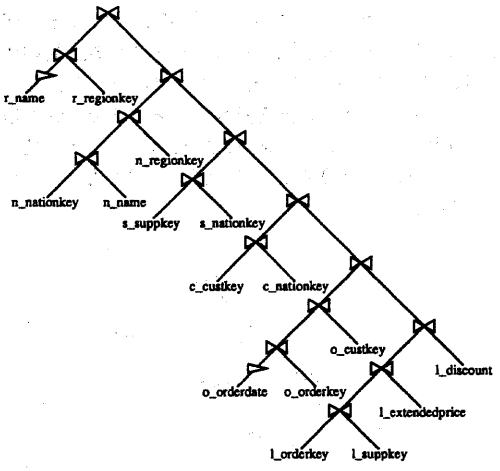

図 12: TPC-D 問合せ 5 の問合せ木

算数の多い問合せ 3 や 5 では、結合プロセス数の少ないときには通常の手法を用いた場合の方が性能が良いが、多くなるにつれてトランスポーズドファイルを用いた場合の方が優れた性能を示すことがわかる。

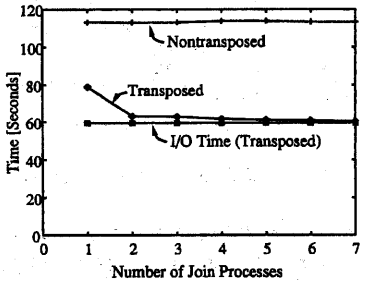

図 13: 結合プロセス数の変化に対する実行時間 (TPC-D 問合せ 6)

5 まとめ

トランスポーズドファイルを用いた問合せ処理について、共有メモリ型の並列計算機上に処理系を実装し、TPC-D を用いて性能評価を行った。結合演算処理を並列に行うことにより、トランスポーズドファイルの有効性を引き出すことができることを明らかにした。今後は集約演算も含めた性能評価をしていきたい。

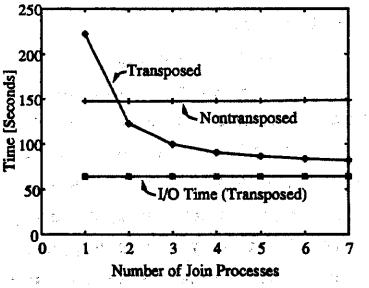

図 14: 結合プロセス数の変化に対する実行時間 (TPC-D 問合せ 3)

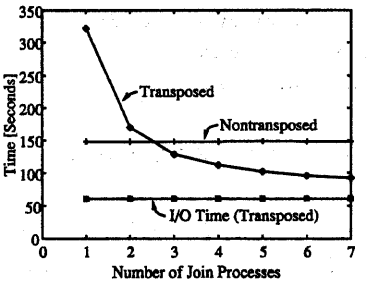

図 15: 結合プロセス数の変化に対する実行時間 (TPC-D 問合せ 5)

参考文献

[1] Don S. Batory, "On Searching Transposed Files", ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, pp. 531-544.

[2] D. J. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", Proceedings of the ACM SIGMOD, Portland, Oregon, 1989, pp. 110-121.

[3] D. A. Schneider and D. J. DeWitt, "Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines", Proceedings of the 16th VLDB Conference Brisbane, Australia, 1990, pp. 469-480.

[4] 喜連川、津高、中野、"共有メモリ型マルチプロセッサによる並列ハッシュ結合演算処理とその評価"、情報処理学会論文誌、Vol. 34, No. 5, May 1993, pp. 1019-1030.

[5] "TPC Benchmark™ D (Decision Support) Standard Specification Revision 1.1", Transaction Processing Performance Council (TPC), San Jose, CA 95112, 19 December, 1995.