3Aa－1

並列関係データベース処理システムに於ける動的負荷分散：実装設計

安井 隆宏, 田村 孝之, 小口 正人, 喜連川 優
東京大学生産技術研究所

1 はじめに

近年, データベースサイズが大規模化する一方で, 意思決定支援システム等に見られるような, 複雑な問合せ, 特に大規模リレーションへの問合せに対する高速な応答への要求が高まっている. しかし, 分散メモリ並列計算機環境においては, ノード間の負荷の偏りのために充分な並列効果を得られないことがあり, タスクスケジューリング等の研究が盛んに行われて来た. 並列データベースシステムにおいても, データの偏り (skew) のためにノード間の負荷にばらつきが生じる事が多い. これに対し, 我々は, 関係データベースにおける Right-deep 方式の多重結合演算処理についてハッシュラインマイグレーションという動的負荷分散アルゴリズムを提案すると共に, シミュレーションによる評価を行い, その有効性を示してきた [1]. 今回, この動的負荷分散アルゴリズムの PC クラスタ [2] への実装設計を行い, そのプロトタイプを実装した. 本稿では, 実験結果をもとに大幅な性能改善が可能であることを示す.

2 Right-deep 結合演算の実行モデル

N ステージで構成された Right-deep 結合演算 を P ノードで処理した際の実行モデルを図1に示す. まず, リレーション $R_1, R_2, \cdots, R_N$ をハッシュ値により処理すべきノードに送出し, 各ノードでハッシュテーブルを作成する. この各ノードのハッシュテーブルをステージフラグメントと呼ぶ. 全ステージのハッシュテーブルの作成が完了すると, リレーション R_P を ディスクから読み出し, ハッシュ値により処理すべきノードへ送出し, プローブを行い, 結合属性が条件に一致した場合には, タプルの結合を行い結果タプルを生成する. さらに, その結果タプルにより次ステージのプローブを行う.

3 動的負荷分散機構

負荷の偏りを解消するため, 結合演算実行中に同一ステージのステージフラグメント間でハッシュラインをマイグレートさせることにより動的に負荷分散を行う (ハッシュラインマイグレーション). 結合演算処理を行うノードの負荷を監視するため, フォアマンと呼ぶ別の制御ノードを一台用意し, これにより集中管理する. 各ノードでは, ステージフラグメント及びハッシュラインに対し, 入力タプル数 N_{in}, 比較回数 N_{cmp}, 出力タプル数 N_{out} の統計を取り, また, 各ノードでは, ハッシュラインのマイグレート先を登録するマイグレーションテーブルを保持する.

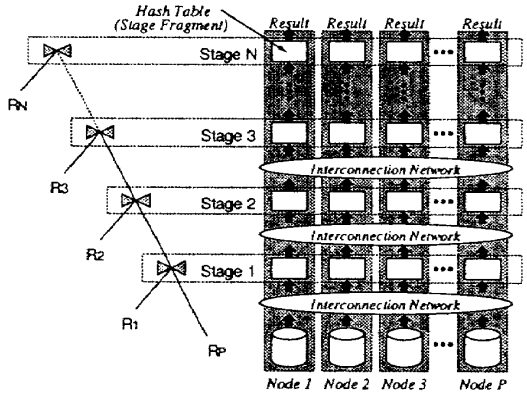

図 1: Right-deep 結合演算実行モデル

フォアマンは, 一定のインターバルで各ノードからステージフラグメントの統計情報を収集し, ノード i, ステージ j の負荷 $SF(i,j)$ を

$$SF(i,j) = N_{in}(i,j) \times C_{recv} + N_{cmp}(i,j) \times C_{cmp} + N_{out}(i,j) \times C_{gen} + N_{out}(i,j) \times C_{send}$$

により算出する. $C_{recv}, C_{cmp}, C_{gen}, C_{send}$ は, それぞれ,1 タプルあたりの受信コスト, 比較コスト, 結果生成コスト, 送信コストを表す. 同一ステージ内のステージフラグメント間で負荷の比較を行い, 負荷の偏りを検出した際には, 更に, 偏りの検出されたステージにおける各ハッシュラインの統計情報を収集する. そして, ハッシュライン k の負荷 $HL(i,j,k)$ を

$$HL(i,j,k) = N_{in}(i,j,k) \times C_{recv} + N_{cmp}(i,j,k) \times C_{cmp} + N_{out}(i,j,k) \times C_{gen} + N_{out}(i,j,k) \times C_{send}$$

により算出し, ステージ内のステージフラグメントの負荷が均等になるように, ハッシュラインマイグレーションのプランニングを行い, 全ノードにマイグレーションプランを送る. マイグレーションが完了すると, マイグレーションテーブルにハッシュラインのマイグレート先を登録する.

4 PC クラスタへの実装と性能評価

PC クラスタ は, 200MHz Pentium Pro マイクロプロセッサを搭載する PC 100 台を 155Mbps ATM および 10Mbps イーサネットの 2 系統のネットワークで相互結合したシステムである. 各ノードには, 研究室で開発した並列データベースサーバ DBKernel が実装されている. この DBKernel にマイグレーション機構を追加することにより, 負荷分散機構を実装し, 10 ノードを用い, 2

Implementation Issues of Dynamic Load Balancing in Parallel Relational Database Systems

T.Yasui, T.Tamura, M.Oguchi, M.Kitsuregawa
Institute of Industrial Science, University of Tokyo roppongi 7-22-1, Minato-ku, Tokyo, 106-0032 Japan.

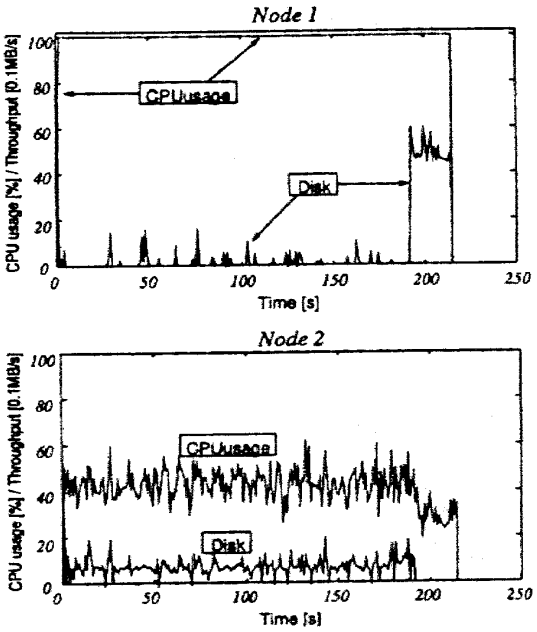

図 2: 動的負荷分散を行わない場合の CPU の利用率と Disk のスループットの時間変化

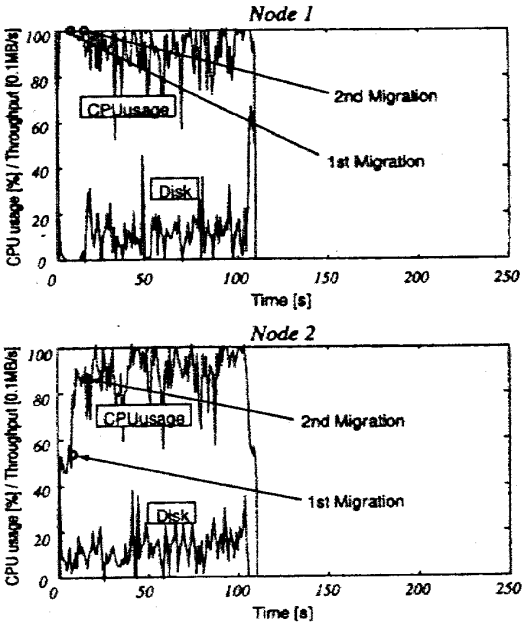

図 3: 動的負荷分散を行った場合の CPU の利用率と Disk のスループットの時間変化

ステージからなる問合せを実行した. 今回の実験では, プローブタプルの属性値を偏らせ, ステージフラグメント間で入力タプル数を偏らせることにより, 負荷の偏りを実現した. ステージ 1 の負荷の比率をノード 1 から順に, 9 : 1 : 1 : ⋯ : 1 とし, ステージ 2 については, 各ノードで均等となるようにした. 各ステージフラグメントのハッシュライン数を 3000 本, ハッシュラインの長さを 1, プローブタプルの総数を 5,000,000 [tuples] とした. この時のノード 1, ノード 2 における CPU の利用率ならびに Disk のスループットの時間変化を 図2に示す. また, 動的負荷分散を行った場合を図3に示す.

動的負荷分散を行わない場合, 負荷の高いノード 1 の CPU の利用率は, ほぼ 100% であるのに対し, 負荷の軽い Node 2 では 50% にも満たない. また, Node 1 では結合演算の処理に忙しくディスクの読み出しを行うことができず, 負荷の軽いノードがディスクからの読み出しを全て終えた後に, 負荷の高いノードでのディスクのスループットが高くなっていることが判る.

これに対し, 図3より動的負荷分散を行うことにより, CPU の負荷およびディスクのスループットをノード間で均衡化させることができる. また, 実行時間も 109.55 [sec] となり, 動的負荷分散を行わない場合の 227.09[sec] に対して, 大幅に短縮出来ることが判る.

今回の実験ではステージ 1 においてノード 1 から他ノードへの 2 回のマイグレーションが確認された. マイグレート先のノードとハッシュラインのマイグレート本数の組の関係を表 1 に示す. 図 3 と比較することにより, 1 回目のマイグレーションでは, 多数のハッシュラインのマイグレートを行い, 大幅に負荷の偏りが改善され, 2 回目のマイグレーションによって微調整が行われていることが判る.

5 まとめ

並列処理システムにおいて問題となる負荷の偏りに対処すべく, ハッシュラインマイグレーション技法を用いた動的負荷分散機構の実装を行った. プローブリレーションに偏りをもたせた実験を行い, 大幅な実行時間の改善を確認した. 今後, 実験をかさね, スケジューリング及び処理方式の検討を行っていく予定である.

source → dest	# of Hash lines	
	1st	2nd
1 → 2	221	45
1 → 3	279	–
1 → 4	199	71
1 → 5	246	38
1 → 6	211	46
1 → 7	162	72
1 → 8	256	32
1 → 9	185	72
1 → 10	231	–

表 1: 動的負荷分散によるマイグレーションプラン

参考文献

[1] S. Davis, M. Kitsuregawa. Simulation Study of a Runtime Load Balancing Algorithm for Pipelined Hash Multi-Joins, 電子情報通信学会 データ工学研究会, Vol.96・No.469, pp.13-18, 1997.1

[2] T.Tamura, M.Oguchi, M.Kitsuregawa Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining SC97: High Performance Networking and Computing, 1997.