データベース・システム 82－1
（1991. 3. 11）

# 資源共有型マルチプロセッサにおける データベース処理の動的負荷配分法

平野泰宏，佐藤哲司，井上潮，寺中勝美

NTT情報通信処理研究所

本稿ではデータベースを並列処理する際の負荷配分コストを削減する動的な配分量決定法を提案する。負荷を固定の配分単位でプロセッサに配分する従来法では、負荷配分に要する時間とプロセッサのアイドル時間の間にトレードオフがあり、かつ、これらの時間はデータ数、プロセッサ数、データ分布に依存するため、配分単位の最適値を事前に決定することが困難であった。本稿で提案する方法は、処理の進行とともに配分単位を動的に変えていくことによりアイドル時間を増加させずに負荷配分時間を削減するものである。資源共有型のマルチプロセッサを用いた性能評価によって、提案法が従来法と比較して高い性能を安定的に得られることを確認した。

# Load Sharing Algorithm for Parallel Database Processing on Shared Everything Multiprocessors

*Yasuiro Hirano, Tetsuji Satoh, Ushio Inoue, Katsumi Teranaka*

NTT Communications and Information Processing Laboratories

1-2356 Take, Yokosuka-shi, Kanagawa 238-03, Japan

This paper presents a new load sharing algorithm for parallel database processing. In ordinary algorithms system performance varies with the number of jobs allocated at one time, and the optimum number can not be determined in advance because it depends on several factors such as database size, number of processors and data distribution. The proposed algorithm solves these problems by varying the number of jobs allocated at one time, which was fixed in ordinary algorithms. Performance evaluation shows that the performance of the proposed algorithm is independent of those factors and is better than ordinary algorithms.

## 1. はじめに

近年、大規模なリレーショナルデータベース（ＲＤＢ）が構築されるようになり、また、販売情報管理、トラヒック管理などのアプリケーションではリレーション中の全てのデータを調べる問い合わせが実行されるようになってきた。このような大規模ＲＤＢのフルサーチを伴う問い合わせのレスポンス時間短縮のためには複数のプロセッサを用いてデータ毎に並列処理することが有効である。その理由は、

(1) 処理対象となるデータが大量である、

(2) 単純な問い合わせでは、データ毎に独立に処理することができる。また、結合処理を含む複雑な問い合わせでも３フェイズ程度に分割すればデータ毎に独立に処理できる [4]、

ためである。このような並列処理を実現しているシステムとして Tandem[1]や Gamma[2]などがある。

これらは資源非共有型であり、データのプロセッサへの配分はデータ格納時に静的に決定される。このため、各プロセッサに格納されるデータ数や選択されるデータ数に偏りを生じると、最も処理時間の長いプロセッサによって全体の処理時間が決まることになり、プロセッサ全体の処理能力を有効に利用できなくなる。

一方、資源共有型マルチプロセッサでは、非共有型より少ないコストで全てのプロセッサに動的に負荷を配分できる。このため、データ格納の問題とプロセッサ処理の問題を分離できる。本稿では、資源共有型マルチプロセッサを用いて大規模データベースの並列処理を行う際に、データをプロセッサに動的に配分する負荷配分法を扱う。

従来の負荷配分法[3]では、１回に配分するデータ数、即ち配分単位は固定であった。この配分単位の大きさによって負荷配分のオーバヘッドとプロセッサのアイドル時間が変化するため、性能が大きく変動する。しかも、配分単位の最適値を事前に決定できないという問題点があった。

本稿で提案する負荷配分法は、処理の進行にともなって配分ページ数を動的に変更することによって性能の変動を小さくし、しかも従来法より優れた性能を得ることができる。

まず、２章で従来の負荷配分法の問題点を指摘し、３章でその問題を解決した新しい負荷配分法を提案する。そして、４章で提案法の有効性を定量的に示す。

## 2. 従来の負荷配分法とその問題点

### 2.1 対象とするモデル

本稿では、以下の前提条件をおく。

(1) 資源共有型のマルチプロセッサでリレーションのフルサーチを伴う問い合わせの処理を行う。

(2) データベース中のリレーションは固定サイズの複数のページに分割して格納されている。

(3) ページの総数（≫プロセッサ数）が既知である。

(4) 各ページはどのプロセッサに対しても同じ時間で配分できる。

(5) プロセッサの処理はページ毎に独立に実行できる。

(6) ページあたりの処理時間の最大値 $T_{max}$と最小値 $T_{min}$を前もって予測できる。

以上の条件のもとでページのプロセッサへの配分を決定する負荷配分法を検討する。負荷配分の目的は、単一の問い合わせの処理が開始されてから終了するまでの経過時間（以後応答時間という）をできるだけ短くすることである。

並列度を変えても本来の問い合わせ処理自体に要する時間（全てのプロセッサの処理時間の合計）は同じであるから、並列処理することによって生じるオーバヘッド時間をできるだけ少なくすることが負荷配分の目的となる。このオーバヘッド時間は以下の２つに分類できる。

(1) 負荷配分時間
配分するページの決定と配分に要する時間およびプロセッサが配分を待つ時間。

(2) アイドル時間
プロセッサが処理をしないで待つ時間のうち、負荷配分待ちを除いたもの。

### 2.2 従来の負荷配分法

独立に実行可能なタスク（負荷配分の対象となる処理の単位。本稿では１ページの処理）が複数のタスク源で生成され、それらのタスクを複数のプロセッサで実行する場合の負荷配分法が文献[3]で分類、評価されている。[3]では、負荷配分法を、

(1) タスク源主導型
タスクが到着すると、タスク源がプロセッサにタスクを送る。

(2) プロセッサ主導型
プロセッサがタスク実行可能になるとタスク源にタスクを取りに行く。

の２種に大別し、負荷配分に使用する情報のレベルが同じであればプロセッサ主導型の方が性能がよいとしている。

２．１節のモデルに従来のプロセッサ主導型の負荷配分を用いると以下のようになる。

(1) 各プロセッサは、一定数のページを取り出して、処理する。処理を終えると、次のページを取りに行く。

(2) 全てのページが処理されるまで、(1)を繰り返す

(1)で１回に取り出すページ数を配分単位と呼ぶ。従来の負荷配分法の動作例を図１に示す。

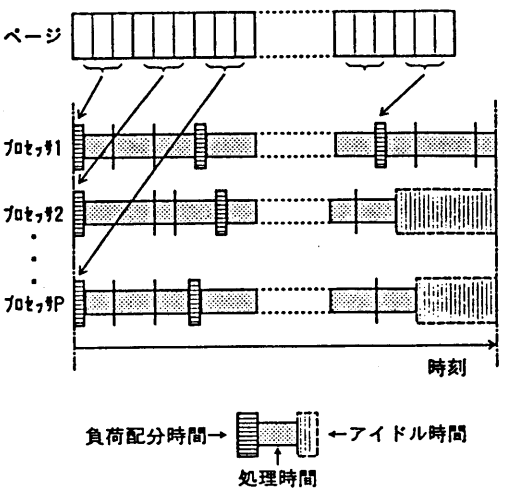



図１．従来の負荷配分法

### 2.3 従来の負荷配分法の問題点

前節で述べた従来の負荷配分法では、オーバヘッド時間は以下のようになる（図１参照）。

(1) 負荷配分時間
取り出すページを決定する時間、その間の排他制御および待ち時間、通信時間など。負荷配分の回数は配分単位に反比例する。

(2) アイドル時間
全てのページの配分が終わった後に処理すべきページがなくなったプロセッサが、最後まで処理しているプロセッサが実行を終えるまで待つ時間。１プロセッサあたりのアイドル時間は配分単位分のページの処理時間未満になる。

負荷配分時間は配分単位を大きくすると減少し、アイドル時間は配分単位を大きくすると増大する。従って、図２(1)のように負荷配分時間とアイドル時間はトレードオフの関係になり、オーバヘッド時間を小さく抑えるには最適な配分単位を決定することが重要になる。最適値は、両者の交叉点付近になる。

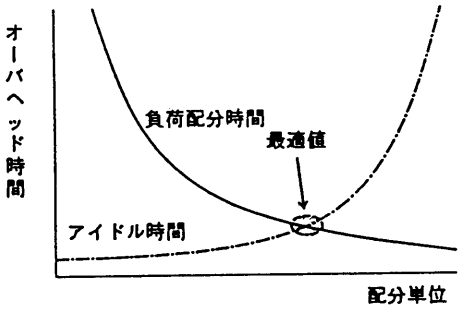
しかし、データベースの並列処理における配分単位の最適値は以下の要因に依存して変動すると考えられる（図２(2)参照）。

(1) ページ数（データベースサイズ）
負荷配分の回数はページ数に比例するので、負荷配分時間はページ数に比例する。一方、アイドル時間はページ数には無関係である。従って、ページ数が多くなると配分単位の最適値は大きくなる。

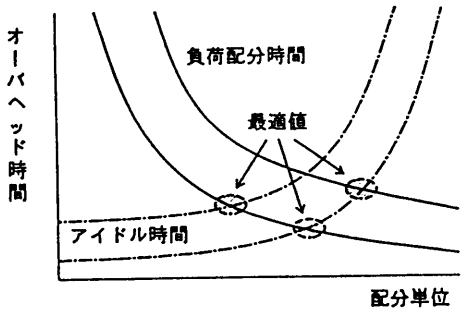
(2) プロセッサ数
プロセッサ数が増えると、アイドル状態になるプロセッサが増えるのでアイドル時間が大きくなる。従って、プロセッサが多くなると配分単位の最適値は小さくなる。

(3) データ分布
ページ毎の処理時間は、ページ内のデータ数や選択されるデータ数によって変化する。アイドル時間は終了間際に取り出されるページの処理時間に依存する。従って、データ分布によって配分単位の最適値が変化する。

上記の内、一般にデータ分布は事前には判らないため、配分単位の最適値を決定することができない。




(1)トレードオフ




(2)最適値の移動

図2．負荷配分時間とアイドル時間

そのため、従来法では必ずしも最適な性能は得られなかった。

## 3. 新しい負荷配分アルゴリズムの提案

### 3.1 方針

従来の負荷配分法では、負荷配分時間とアイドル時間のトレードオフがあり、しかも、配分単位の最適値を事前に決定できないため、必ずしも最適な性能は得られないという問題点があった。

この問題点を解決するために、アイドル時間を小さく抑えたままで負荷配分時間を大幅に削減することを考える。即ち、アイドル時間が小さい（配分単位が小さい）領域で負荷配分時間をアイドル時間程度に小さくできれば、オーバヘッド時間をその領域内でほぼ一定にでき、配分単位の大きさに関係なく最適に近い性能が得られる。図3にその概念を示す。

2.3 で述べたように、負荷配分時間は主に配分回数に依存し、アイドル時間は主に処理終了間際の配分単位に依存する。そこで、図4のように

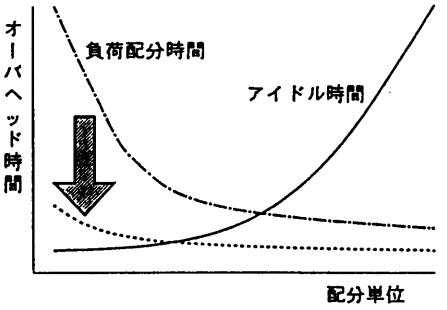



図3．提案法の概念

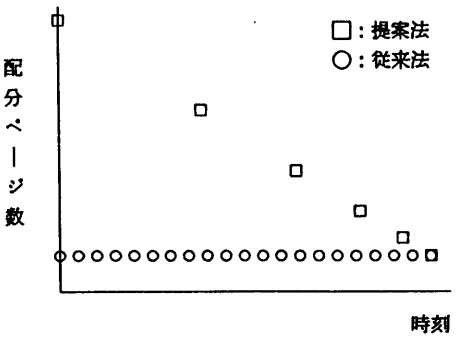



図4．動的な配分ページ数決定

(1) 最初は多くのページを配分して、負荷配分回数を減らし、

(2) 処理の進行に従って配分ページ数を徐々に少なくしていくことによって、アイドル時間を小さくする

ことによって、アイドル時間を小さく抑えたままで配分回数を削減できると考えられる。

### 3.2 配分ページ数決定法

なるべく多くのページを配分して配分回数を削減しつつ、アイドル時間を小さく抑えたままにできるような配分ページ数決定法を導く。

まず、配分ページ数とアイドル時間の関係を調べ、次にそれを許容限度以下にできる配分ページ数の最大値を求める。以下の場合を考える。

- $n$ページを$P$個のプロセッサで処理する。
- プロセッサ$i$ $(1 \leq i \leq P)$に $b_i$ページを配分する。
- プロセッサ $i$が $b_i$ページを処理し終える前に、他の $(P-1)$個のプロセッサが、残りの $(n-b_i)$

ページの処理を終える。

この場合のアイドル時間が最大になるのは、ページあたりの処理時間の最大値を $T_{max}$と最小値を $T_{min}$とすると、

- プロセッサ $i$に配分されたページは全て処理時間が $T_{max}$であり、かつ
- 他の$(n-b_i)$ページは全て処理時間が $T_{min}$である

場合である。

この場合のプロセッサ $i$の処理時間は

$$b_i \times T_{max} \tag{1}$$

であり、残りのページの処理に要する延べ時間は

$$(n-b_i) \times T_{min} \tag{2}$$

であるから、アイドル時間の総和は

$$b_i \times T_{max} \times (P-1) - (n-b_i) \times T_{min} \tag{3}$$

となる。

アイドル時間を小さく抑えたまま多くのページを配分したいのであるから、アイドル時間を与えられた許容限度 $T_{idl}$以下にできる最大の $b_i$を求めると

$$b_i = \frac{n + T_{idl}/T_{min}}{(T_{max}/T_{min}) \times (P-1) + 1} \tag{4}$$

となる。式(4)を、配分の度に繰り返し用いるように書き換えると次式が得られる。

$$b_i(t) = \left\{ n(t) + \sum_{j=1}^{P} m_j(t) + \alpha \right\} / \beta - m_i(t) \tag{5}$$

$$\alpha = T_{idl}/T_{min} \tag{6}$$

$$\beta = (T_{max}/T_{min}) \times (P-1) + 1 \tag{7}$$

ただし、

$b_i(t)$ ：時刻$t$にプロセッサ $i$に配分するページ数 $(i=1,2,\cdots P)$

$n(t)$ ：時刻 $t$における未配分ページ数

$m_j(t)$ ：時刻 $t$におけるプロセッサ $j$の未処理ページ数 $(j=1,2,\cdots P)$

$T_{idl}$ ：アイドル時間の総和の許容限度（ただし、$T_{idl} = T_{max} \times (P-1)$）

$T_{max}$：１ページあたりの処理時間の上限

$T_{min}$：１ページあたりの処理時間の下限

$P$ ：プロセッサ数

である。式(5)で、

$$n(t) + \sum_{j=1}^{P} m_j(t)$$

が時刻$t$においてまだ処理されていないページ数（式(4)では$n$）であり、配分されたがまだ処理されていないページ数 $m_i(t)$を差し引いている。

式(5)を用いた負荷配分法は以下のようになる。

方法１．自プロセッサにのみ配分

配分されたページの処理を終えた（ $m_i(t)=0$となった）プロセッサが、式(5)を計算し、 $b_i(t)$ページを取り出し、処理する。

### 3.3 データ収集回数の削減

前節で述べた方法１では、配分１回につき $P$個の $m_i(t)$を収集する必要があるので、データ収集のために負荷配分１回あたりの時間が長くなることが予想される。本節では、データ収集回数を削減する方法として、データ収集契機を減らす方法と、契機毎の収集回数を減らす方法の２つ示す。

#### 3.3.1 データ収集契機の削減

方法１では $P$個の $m_i(t)$を収集して１つのプロセッサにのみ配分するが、以下のように、同じ量のデータを用いて全てのプロセッサに配分することができる。

方法２．全プロセッサに配分

少なくとも１つのプロセッサが配分されたページの処理を終えると、 $P$個の $m_i(t)$を収集し、式(5)を $P$回計算して、 $P$個全てのプロセッサにページを配分する。

方法２では、方法１と同じ量のデータを用いて $P$個のプロセッサ全てに配分するので、データ収集毎の配分ページ数が増えるため、データ収集回数を減らすことができる。

しかし、$m_i(t)>0$のプロセッサにも配分するため、１回に１つのプロセッサに配分するページ数は方法１より少なくなり、配分回数は多くなる。

### 3.3.2 契機毎の収集回数の削減

式(5)では配分の度に $P$個のプロセッサから未処理ページ数 $m_i(t)$を収集していたが、過去に収集した値を用いることによってデータ収集回数を削減する方法を示す。

未処理ページ数$m_i(t)$は、以下のようになる。

$$m_i(t)=alloc_i(t)-report_i(t)-procc_i(t) \qquad (8)$$

$alloc_i(t)$ ：時刻$t$までにプロセッサ$i$に配分したページ数

$report_i(t)$ ：時刻$t$までにプロセッサ$i$が処理し、既に報告されたページ数

$procc_i(t)$ ：時刻$t$までにプロセッサ$i$が処理したが、まだ報告されていないページ数

式(8)の右辺で利用可能なのは $alloc_i(t)$および $report_i(t)$である。

アイドル時間を許容限度 $T_{idl}$以下にするためには、配分ページ数を式 (5)の $b_i(t)$以下にする必要がある。そこで、式 (5)で

$$+\sum_{j=1}^{P} m_j(t) \ \rightarrow \ +0 \qquad (9)$$

$$-m_i(t) \ \rightarrow \ -\{alloc_i(t)-report_i(t)\} \qquad (10)$$

とし、

$$b_i'(t)=\{n(t)+\alpha\}/\beta-\{alloc_i(t)-report_i(t)\} \quad (5')$$

で配分ページ数を計算する。

式(5′)を用いると、式 (5)より配分ページ数が少なくなるが、時刻 $t$以前の任意の時刻における情報で配分できるので、データ収集回数を削減できる。

そこで、あるプロセッサが配分されたページの処理を終えた時点で、他のプロセッサのデータを収集しないで、そのプロセッサのデータのみを用いて配分ページ数を決定する。これは、他のプロセッサは最後に配分されたページを全く処理していないとみなすのと同じである。式(5′)の第1項は配分の度に少なくなり、他のプロセッサに対しては第2項（最後に配分されたページ数）以下となるので式(5′)≦0となる。従って、一度に1つのプロセッサにしか配分できない。

以上により、次の方法3が得られる。

方法3．過去のデータを使用

配分されたページの処理を終えたプロセッサが次式で計算した数のページを取り出して、処理する。

$$b_i''(t)=\{n(t)+\alpha\}/\beta \qquad (11)$$

方法3では、配分1回につきデータ収集1回で済むが、式(9)の近似により1回の配分ページ数が少なくなるため、配分回数は方法1より多くなる。

## 4. 性能評価

本章では、まず、第3章で提案した3つの方法をシミュレーションにより比較し、負荷配分回数、データ収集回数の観点から優れた方式を選択する。次に、選択した方法と従来法を資源共有型マルチプロセッサに実装して、負荷配分時間とアイドル時間、応答時間を実測により評価する。

### 4.1 シミュレーション結果

方法1～方法3の配分回数およびデータ収集回数をシミュレーションにより評価した。シミュレーション条件と結果を図5に示す。

図5の横軸の最小配分単位は、処理終了間際の配分ページ数で、配分ページ数の最小値となる。アイドル時間の許容限度 $T_{idl}$とは以下の関係がある。

$$\text{最小配分単位}b \ \rightarrow \ T_{idl}=b\times T_{max}(P-1)$$

3つの方法を比較すると、以下のようになる。

(1) 配分回数は

方法1 ＜ 方法3 ＜ 方法2

である。

(2) データ収集回数は、

方法3 ＜ 方法2 ≪ 方法1

である。

方法1と方法3の配分回数の差はわずかであるのに、方法1のデータ収集回数は方法3より極めて大きい（ほぼプロセッサ数倍になる）。従って、方法3が最も優れていると考えられる。

なお、図5の横軸の最小配分単位は従来法の配分単位に相当する。従来法では

$$\text{配分回数} = 10{,}000 \ / \ \text{配分単位}$$

であるから、図5より、提案法によって配分回数を最大1桁以上削減できていることがわかる。

### 4.2 実装と評価モデル

従来の負荷配分法および提案した負荷配分法の方法3を資源共有型マルチプロセッサ Sequent S27に実装して評価した。評価モデルと計算機の仕様を表1に示す。

負荷配分法の効果を明確にするために、データベースは主記憶上にあるとし、検索結果の主記憶上の一時テーブルへの書き込みが完了するまでの時間を測定した。問い合わせの解析やプロセス起動の時間は測定対象から除外した。

評価項目は以下の2点である。

(1) アイドル時間と負荷配分時間

(2) 応答時間

### 4.3 負荷配分時間とアイドル時間

最小配分単位を変えて負荷配分時間とアイドル時間を測定した結果を図6に示す。提案法では、アイドル時間は従来法と同程度のままで、最小配分単位が小さい領域での負荷配分時間を極めて小さくできていることが判る。

### 4.4 応答時間

2章で述べた配分単位の最適値に影響する3つの要因、即ち、

(1)ページ数

(2)プロセッサ数

(3)データ分布

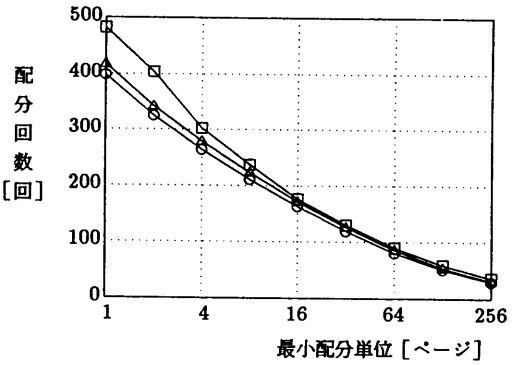
をパラメータとして、最小配分単位対応答時間を測定した結果を図7に示す。

従来法では、負荷配分時間とアイドル時間のトレードオフのために最小配分単位を変えると応答時間が大きく変化している。しかも、配分単位の最適値は上記の要因によって変化しているため、最適値を事前に決定することはできなかった。

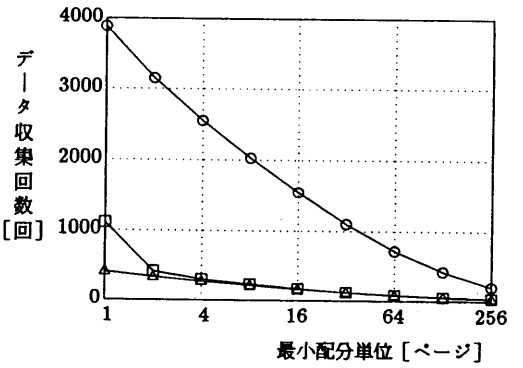
一方、提案法では、最小配分単位が小さい範囲では応答時間が一定になっている。そのため、最小配分単位を1で固定しても、最適に近い性能を得られる。しかも、従来法で配分単位の最適値を選んだ場合よ

| ページ数$n(0)$ | 10,000 |
|---|---|
| プロセッサ数$P$ | 10 |
| ページ毎の処理時間 | 正規乱数 |
| 平均 | 1000 |
| 標準偏差 | 230 |
| 最大値$T_{max}$ | 1917 |
| 最小値$T_{min}$ | 227 |

(1)シミュレーション条件




(2)配分回数




(3)データ収集回数

○：方法1．自プロセッサのみに配分

□：方法2．全プロセッサに配分

△：方法3．過去のデータを使用

図5．シミュレーション結果

表1．評価モデル

| | |
|---|---|
| データベース | |
| テーブル | Wisconsin DB [5] |
| タプル長 | 208B |
| タプル数 | 100,000 |
| ページサイズ | 2KB |
| ページ数 | 11,112 |
| 格納場所 | 主記憶 |
| 問い合わせ | 単純選択 |
| 選択率 | 10% |
| 計算機 | Sequent S27 |
| プロセッサ | i80386(16MHz)×8 |
| ページあたりの処理時間 | |
| 最大値$T_{max}$ | 12.7 [m sec] |
| 最小値$T_{min}$ | 2.2 [m sec] |

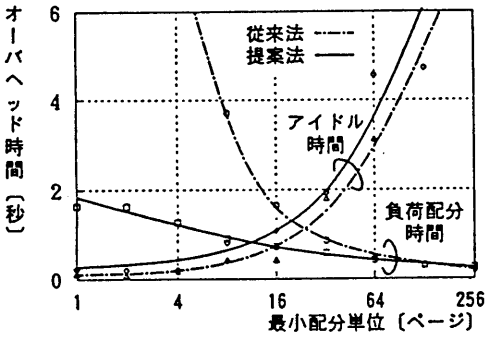



図6．負荷配分時間とアイドル時間

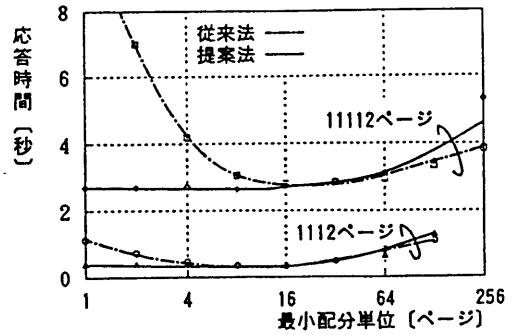
り応答時間を短縮できている。

### 4.5 スピードアップ率

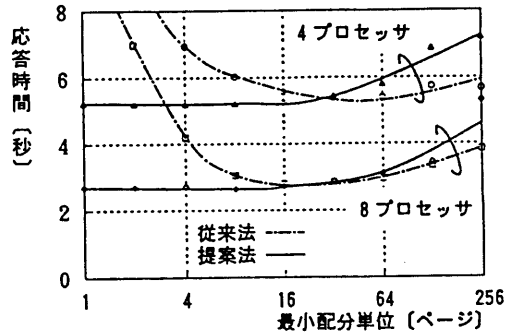
プロセッサ数を変えてスピードアップ率を測定した結果を図8に示す。従来法ではプロセッサ数を6以上にすると速度向上度がプロセッサ数に対してサブリニアになる。提案法では、従来法よりもスピードアップ率が高く、しかも8プロセッサまでリニアに性能が向上している。

## 5. まとめ

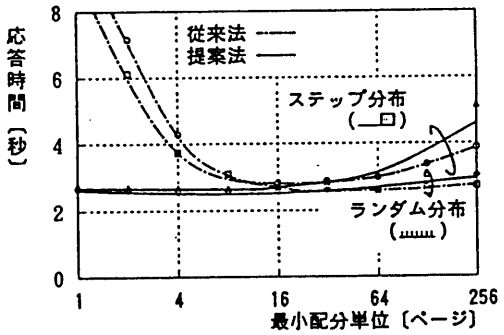
本稿では、データベースへの問い合わせの並列処理を行う際の従来の負荷配分法の問題点を指摘し、それを解決した新しい負荷配分アルゴリズムを提案し、性能評価によってその有効性を示した。




(1)ページ数の影響の削減




(2)プロセッサ数の影響の削減




(3)データ分布の影響の削減

図7．応答時間

ページを固定の配分単位で配分する従来の負荷配分法では、負荷配分時間とアイドル時間のトレードオフがあり、配分単位を変えると性能が大きく変化した。しかも、配分単位の最適値はページ数、プロセッサ数、データ分布によって変化するため、事前に決定することができなかった。

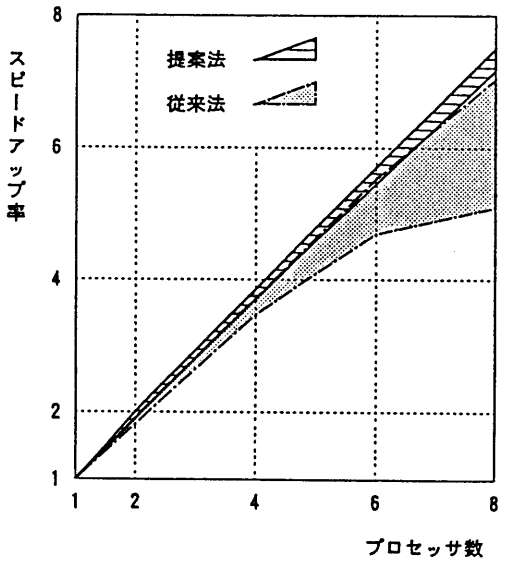



図８．スピードアップ率

そこで、配分ページ数を動的に変えることによって、アイドル時間を増加させることなく負荷配分時間を削減して、最小配分単位に依存しない少ないオーバヘッドを実現できる負荷配分法を提案した。

資源共有型マルチプロセッサを用いた性能評価の結果、提案法の性能は従来法より安定し、しかも高いことが明らかになった。

今後の課題としては、Ｉ／Ｏとオーバラップさせた時の性能評価、事前に予測したページ毎の処理時間の最大値と最小値の誤差の影響の分析などがある。

## 参考文献

[1] S. Englert, J. Gray, T. Kocher, P. Shah : "A Benchmark of NonStop SQL Release 2 Demonstrating Near-Linear Speedup and Scaleup on Large Databases," Technical Report 89.4 Tandem Part No. 27469, (May 1989)

[2] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen : "The Gamma Database Machine Project," IEEE Trans. on Knowledge and Data Engineering, Vol. 2, No. 1 (Mar. 1990)

[3] Y.-T. Wang, R. J. T. Morris : "Load Sharing in Distributed Systems," IEEE Trans. on Computers, Vol. C-34, No. 3, (Mar. 1985)

[4] D. A. Schneider and D. J. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," Proc. of the 1989 SIGMOD, (June 1989)

[5] D. Bitton, D. J. DeWitt, C. Turbyfill : "Benchmarking Database Systems – A Systematic Approach," Proceedings of the 1983 Very Large Database Conference, (Oct. 1983)