社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－ARC－167 (19)
2006－HPC－105 (19)
2006／2／28

有限サイズの multi-master divisible load 問題に対する再分散スケジューリングアルゴリズム

須田礼仁† 富さやか†

Multi-master divisible load は通信と計算を同時にスケジュールするデータ再分散のためのモデルである．著者らはタスクが無限に多くなる極限で最適解に漸近する漸近最適スケジューリングを提案し，シミュレーション・実機で評価を行ってきたが，本稿ではタスク量の有限性を意識したスケジューリングの改良を提案する．改良手法ではプロローグにも計算が導入され，各ラウンドのサイズは可変である．厳密な最適化にはやや計算量がかかるが，1 ラウンドが最適になる場合は容易に判別できる．シミュレーションによる評価では，従来手法よりも理論的下限にずっと近い makespan を達成した．

A Redistribution Scheduling Algorithm for Finite-Sized Multi-Master Divisible Load Problems

Reiji SUDA † and Sayaka TOMI†

Multi-master divisible load provides data redistribution schemes with co-scheduled communication and computation. The authors proposed an asymptotically optimum scheduling that approaches asymptotically to the optimum solution at the limit of infinitely many tasks, and evaluated it via simulation and implementation. This paper proposes an improved algorithm aware of finiteness of tasks, where computations are introduced into the prologue, and the sizes of the rounds are variable. Exact optimization may require high computational complexity, but the cases of a single round being optimum are easily checked. In simulations, the new algorithm attains makespans much nearer to theoretical lower bounds than the old ones.

1. はじめに

負荷の均衡化は並列処理の基本問題の一つである．負荷均衡化の問題設定にはいろいろあるが，タスクの初期配置がプロセッサの処理能力に適合していない場合や，タスクやシステムが動的に変化するような場合には，負荷の均衡化のためにタスクの再分散が必要となる．

タスク再分散では，通信と計算を別フェーズとして扱うのが古典的である．すなわち，負荷の均衡化を制約条件として再分散のための通信コストを最小化するのである．しかし，一般には通信と計算を同時にスケジューリングするほうが短い makespan を実現できる．また，同一のプロセッサ間でも通信を分割して複数回にわたってタスクを転送するほうが，一度しか転送しない手法よりも短い makespan を達成する．しかし計算量的な問題から厳密な最適化は現実的ではない．

これに対して，モデルを簡単化することにより高速に良質の近似解を求める提案のひとつが我々が提案してきた Multi-Master Divisible Load (MMDL) Model[1] である．我々はこのモデルに対する漸近最適スケジューリングアルゴリズムを提案し，シミュレーション[3] および実機実装[4] により評価を行い，またマルチクラスタに拡張[2] してきた．

本稿では，以前のシミュレーション評価において見られた問題の解決として，タスクサイズの有限性を考慮したスケジューリングの改善を提案する．

2. 従来の漸近最適スケジューリング

2.1 問題設定

まず，本稿で考えるモデルと問題設定を明確にしておく．

タスクは任意の正の実数のサイズに分割することができる．これが divisible の名前の由来である．実用の際には整数に丸めるなどして近似解を得ることになる．

タスクは均一であり，タスクの転送や処理（実行）にかかる時間は，タスクのサイズのみで決まる．処理時間はプロセッサに依存してもよいが，本稿では通信時間はプロセッサに依存しないことを仮定する．具体的には，サイズ x のタスクを任意のプロセッサ間で通信する際の通信時間は $\alpha+\beta x$，プロセッサ i で処理する際の計算時間は $\gamma_i x$ とする．各プロセッサはシングルポートとするが，ネットワークはクロスバーであって通信は衝突しないとする．また本稿では通信と計算は同時に実行できないものとするが，計算と通信が同時に実行できるモデルにも，計算中の通信性能が低下しないという仮定をおけば，本稿の結果を容易に拡張できる．

† 東京大学 情報理工学系研究科 コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, the University of Tokyo

タスクは独立で，相互に通信を行うことなく並列に処理することができるとする．計算結果のデータをそのタスクを持っていたプロセッサに収集するようにモデルを修正することは難しくない．

初期状態で第 i プロセッサに割り当てられている**初期タスク量**を x_i とする．古典的な divisible load model では，唯一つのプロセッサを除いて $x_i = 0$ となる**マスタワーカモデル**であった．MMDL モデルではこの制約が取り除かれ，任意のプロセッサが任意の量のタスクを初期状態で持つことができる．これにより動的負荷分散のような問題にも適用できるようになる．

解きたい問題は，プロセッサにタスクの処理や転送を行わせ，処理や転送の開始からすべてのタスクの処理が終了するまでの時間 (**makespan**) を最小にするスケジュールを求めることである．

2.2 従来の漸近最適スケジューリング

つづいて従来の漸近最適スケジューリングのアルゴリズムを略述する．手法の詳細は原報告[1] にゆずる．

MMDL 問題の厳密な最適解はマスタワーカモデルに限定しても容易には得られない．しかし，通信の立ち上がり遅延 α を 0 と仮定すると，自明な最適解が得られる．

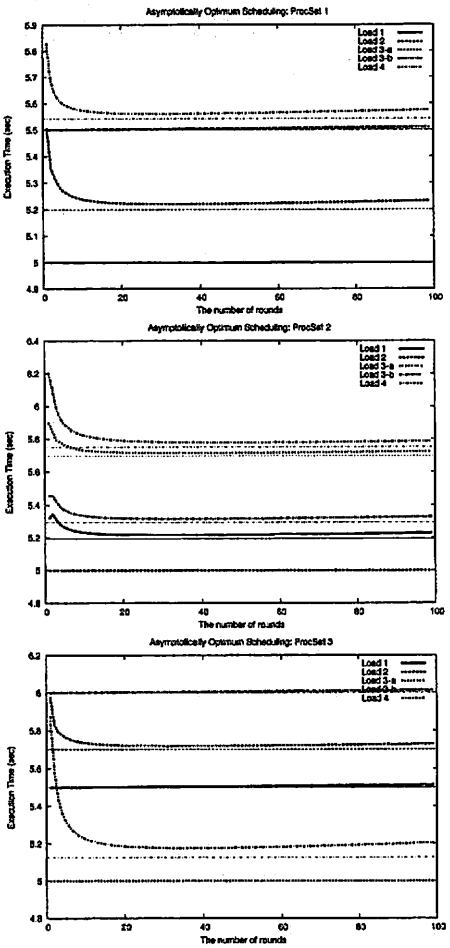
$\alpha = 0$ の仮定の下では，通信時間も計算時間もタスクサイズに比例するので，タスクを分割することによるロスがまったくない．そこで，**ラウンド**と呼ばれる一定の通信・計算パターンを反復するスケジューリングを仮定し，その**定常状態**を最適にスケジューリングする．すると，**プロローグ**と**エピローグ**を除いて最適にスケジューリングされるが，**ラウンド数**を無限に大きくするとプロローグとエピローグの影響は無限に小さくなる．これが $\alpha = 0$ における最適解である．この最適解の makespan を T とすると，$\alpha \geq 0$ における最適解は T よりも短い makespan を持つことはない．

定常状態のスケジューリングでは，あるラウンドの計算に必要となる通信は，前のラウンドで行うことにする．これにより，ラウンド内のスケジューリングでは通信と計算の依存性を考慮する必要がなくなる．プロローグは次ラウンドで必要となる通信のみ，エピローグは計算のみとなる．定常状態の最適スケジューリングはプロセッサ数を p として $O(p \log p)$ の計算量で求めることができる．

従来の漸近最適スケジューリングは，この最適定常スケジューリングをそのまま有限の α の場合に適用する．分割数 (エピローグを除くラウンドの数) を k とすると，$\alpha = 0$ の場合には 1 ラウンドの所要時間は T/k となるが，有限の α では $T/k + A$ となるものと仮定する．プロローグとエピローグを含めて全体の makespan は $T + T/k + kA$ 以下と評価できる．これは $k = \sqrt{T/A}$ で最小値 $T + 2\sqrt{TA}$ を取る．最適解の makespan は T 以上であるから，近似解の makespan は最適解の $1 + 2\sqrt{A/T}$ 倍以下である．A を定数と見なせば，$T \to \infty$ においてこの比は 1 に漸近する．これが**漸近最適スケジューリング**である．

図 1　従来の漸近最適スケジューリングにおけるラウンド数と makespan の関係 (訂正版)

Fig. 1　The number of rounds vs makespan in the original asymptotically optimum scheduling (corrected version)



2.3 従来の評価と問題点

我々は上記の漸近最適スケジューリングをシミュレーションにより評価した[3]．負荷均衡を条件として通信量を最小化する古典的手法である「比例分割」と，「動的負荷分散」の手法の一つである work stealing と比較したところ，漸近最適スケジューリングは様々な問題設定において安定してよいスケジュールを与えることが示された．なお，報告 3) では漸近最適スケジューリングの実装にバグがあり，実際には 3) で報告されたものよりも高い性能が得られる．図 1 は報告 3) の図 5〜7 の修正版，表 2 は報告 3) の表 3 の修正版である．

同報告ではいくつかの課題も示された．ひとつは実際の動作が前述の解析で想定したものとは相当に異なることである．この原因はプロローグが通信だけからなる点にある．すなわち，プロセッサによってプロローグの通信

の終了時刻が異なるので，ばらばらなタイミングで「定常状態」に入るのである．その影響の一つとして，図1の中央の図のようにラウンド数が1のときのmakespanがその他に比べて不規則に短い場合が生じる．

もう一つは，ラウンドの大きさを変化させることでmakespanを小さくできる可能性が追求されていない点である．前述の$T+T/k+kA$というmakespan評価のうち，T/kの項はプロローグの所要時間であり，kAの項はラウンドごとのオーバヘッドの和である．従って，小さなプロローグで開始し，順次ラウンドサイズを大きくすることでラウンド数を抑えることができれば，makespanをさらに短くできるはずである．

次節ではこれら2つの課題を解決する手法を提案する．

3. 漸近最適スケジューリングの改良

従来の漸近最適スケジューリングは定常状態のスケジューリングのみを考えており，プロローグとエピローグに関する考察が不十分である．改良手法は特にプロローグに注目してスケジューリングを改良する．以下の手法は従来手法の狭義の改良になっており，理論的には従来手法よりもmakespanは決して大きくならない．

3.1 漸近最適スケジューリングの正規化

まず，従来の定常状態のスケジューリングに修正を加えて問題を正規化する．従来手法の以下の議論に必要となる部分について若干説明を追加する．

3.1.1 タスク量の正規化

漸近最適スケジューリングは大きく分けて2つのステップからなっている．第1ステップでは，**所要時間の下限**Tと，各プロセッサが通信するタスク量y_i（**追加タスク量**と呼んでいる．$y_i>0$ならプロセッサiはワーカ，$y_i<0$ならマスタ）を決定する．

プロセッサiの通信と計算の所要時間の合計はmakespan以下であるから

$$\gamma_i(x_i+y_i)+\beta|y_i|\leq T$$

である．すべてのプロセッサに対してこれが等号で成り立てば完全に負荷が均衡しているが，問題設定によってはそれが不可能な場合もある．不等号はアイドル時間を意味しているが，このアイドル時間は最適スケジューリングでも生じるものであり，アルゴリズムの工夫では避けられない．

そこで，全プロセッサについて

$$\gamma_i(x_i+y_i)+\beta|y_i|=T$$

となるように，**ダミーのタスク**を追加してx_iを調整しておくことにする．こうしておけば，避けられないアイドル時間がペナルティと見なされずにすむ．

3.1.2 スケジューリングの正規化

漸近最適スケジューリングの第2ステップはマッチング（各マスタがどのワーカにどれだけのタスクを送るかの決定）と通信スケジューリングからなる．マッチングには**区間交差マッチング**という手法を用いる．おおざっぱに説明すると，プロセッサをその通信量$|y_i|$の長さの区間に対応させて，マスタ・ワーカそれぞれ一列にならべると同じ長さになるので，重なった部分の長さをマスタからワーカに送るタスク量とする．

このようにしておくと，**逆順通信スケジューリング**により効率的な通信スケジューリングを決めることができる．すなわち，マスタが複数のワーカに送信する場合には，マッチングの際のワーカの順序に従って送信し，ワーカが複数のマスタから受信する場合には，マッチングの際のマスタの順序の逆順で受信する．

このようにすると，複数の通信をするプロセッサにとって最初の通信は，通信相手にとっても最初の通信となる．よってこの場合には通信待ちが生じない．同様に最後の通信は相手にとっても最後となる．それ以外の場合，マスタとワーカの少なくとも一方は通信相手が1プロセッサだけなので，複数の通信を行うプロセッサとって都合のよいタイミングで通信を行えばよい．以上のことから，複数の通信相手を持つプロセッサは，最後の通信以外は待たされることがない．従って，各プロセッサの通信パターンは

- ラウンドの最初にのみ通信がある
- ラウンドの最初と途中（または最後）の2箇所に通信がある
- ラウンドの途中（または最後）にのみ通信がある

の3種類に限定されることがわかる．

通信はラウンド内でできるだけ早くスケジュールすることにする．ラウンド内にどのプロセッサも通信を行わない時間帯がある場合には，それをエピローグに移すことにより，全体として通信のタイミングを早めることができる．このように，マッチングと通信スケジューリングを固定した条件下でできるだけ通信を早めておく．以下では，エピローグに移した計算だけの時間帯は，スケジューリングが自明であるため，考慮から外す．

3.2 スケジュール改良のための定義

以降の議論のために必要となるいくつかの定義をまとめて導入しておく．

まず，第jラウンドの**サイズ**をr_jとする．すなわち，基準となるスケジュールのタイミングをr_j倍して第jラウンドのスケジュールとする．従来手法は$r_j=1/k$に相当する．なお，最初のラウンドは第0ラウンドと数える．さらにR_jを

$$R_j=\sum_{i=0}^{j-1}r_i$$

（ただし$R_0=0$）と定義する．エピローグを除いてkラウンドあるとすると，すべての処理がぴったり完了するためには

$$R_k=1$$

が満たされなければならない．

さて，ラウンド中に通信が2回あるプロセッサiの場

合を考える．ラウンドの最初は通信であり，その時間を βw_{i1} とする（ラウンドサイズが 1 の場合．実際には第 j ラウンドではこの r_j 倍となる）．次は計算であり，その時間を $\gamma_i w_{i2}$ とする．次は通信であり，その時間を βw_{i3} とする．最後に計算が来る場合にはその時間を $\gamma_i w_{i4}$ とする．通信で終わる場合は $w_{i4}=0$ とおけばよい．

ラウンドの最初にのみ通信がある場合は $w_{i3}=w_{i4}=0$，ラウンドの途中（または最後）にのみ通信がある場合には $w_{i1}=0$ とすれば，いずれも上記のモデルに含めることができる．

さらに，w_i を

$$w_i = w_{i4} - w_{i3} + w_{i2} - w_{i1}$$

で定義する．ワーカ i に対して w_i は 1 ラウンドに消費されるタスク量である．また，w_i' を

$$w_i' = \max\{w_i, w_{i2} - w_{i1}\}$$

で定義する．

従来の漸近最適スケジューリングでは，アイドル時間はプロローグとエピローグの部分にしかなかった．そこで以下でも，**途中のラウンドではアイドル時間を許さない**ことにする．プロセッサ i の**プロローグでのアイドル時間**を $\zeta_i = \gamma_i z_i$ とする．

タスク量を正規化しておいたので，アイドル時間が ζ_i であるプロセッサは時刻 $T+\zeta_i$ に処理を完了することになる．従って，$\zeta = \max\{\zeta_i\}$ が makespan を決める．

3.3 各ラウンドのスケジューリング可能条件

上述のパラメタのうち，w_{in}, w_i, w_i' は定常状態の最適スケジューリングから決まる値である．アイドル時間 ζ_i，ラウンド数 k，ラウンドサイズ r_j の 3 つを決めればスケジュールは決まる．まずスケジュールが可能となるパラメタの条件を求める．

最初に，プロローグ（第 0 ラウンド）について考える．プロセッサ i がワーカの場合，自前のタスク量が x_i で，アイドルタスク量が z_i，最初の受信で得られるタスク量が $r_0 w_{i1}$ であるから，これらをあわせて最初の計算で処理するタスク量 $r_0 w_{i2}$ をまかなわなければならない．すなわち

$$r_0 w_{i2} \le x_i + z_i + r_0 w_{i1}$$

が必要である．2 度目の計算では，最初の計算で処理されたタスク量と次の受信で受け取ったタスク量を考慮して

$$r_0 w_{i4} \le x_i + z_i + r_0(w_{i1} - w_{i2} + w_{i3})$$

が必要である．これらをあわせると

$$r_0 w_i' \le x_i + z_i \qquad (1)$$

が必要条件となる．マスタは十分なタスクを持っているから，このような条件は課されない．すなわち，すべてのワーカに対して式 (1) が満たされれば，第 0 ラウンドの計算は実行できる．

同様にして第 j ラウンドの場合について考えると，最初の計算で必要なタスク量から

$$r_j w_{i2} \le x_i + z_i - R_j w_i + r_j w_{i1}$$

2 度目の計算で必要なタスク量から

$$r_j w_{i4} \le x_i + z_i - R_j w_i + r_j(w_{i1} - w_{i2} + w_{i3})$$

となる．これらをあわせると

$$r_j w_i' \le x_i + z_i - R_j w_i \qquad (2)$$

となる．これは $R_0=0$ より $j=0$ （プロローグ）では条件 (1) に一致する．

結局，すべてのワーカ i とすべてのラウンド j に対して式 (2) が成立しなければならないことになる．逆にこれが満たされればスケジュールが可能である．

もし $w_i' \le 0$ であれば，$w_i \le w_i' \le 0$ であるから

$$r_j w_i' \le 0 \le x_i + z_i - R_j w_i$$

となり，条件 (2) は常に成立する．よって $w_i' > 0$ のワーカのみ考えればよい．すると条件 (2) は

$$r_j \le \frac{x_i + z_i}{w_i'} - R_j \frac{w_i}{w_i'} \qquad (3)$$

と書き換えることができる．

ここで $R_j + r_j = R_{j+1} \le 1$ であるが，$w_i \le w_i'$ に注意すると $x_i + z_i \ge w_i'$ の場合には $r_j \le 1 - R_j$ に対して常に式 (3) が成立する．よって条件を確認する必要があるのは $x_i + z_i < w_i'$ となるワーカのみである．

3.4 スケジューリングの完了条件

しかしこれだけではスケジューリングが完了するとは限らない．有限の k に対して $R_k = 1$ が達成されることが必要である．

まず，$r_0 > 0$ が必要である．これは (3) より

$$\min\{x_i + z_i\} > 0 \qquad (4)$$

で得られる．x_i は初期タスク量で非負であるから，$x_i = 0 < w_i'$ なるワーカに対して $z_i > 0$ が要求されることになる．

また，常に $r_j > 0$ でなければならない．これは式 (3) の右辺が $0 \le R_j < 1$ で常に正であれば得られるから，条件 (4) と

$$\min\{x_i + z_i - w_i\} \ge 0 \qquad (5)$$

をあわせればよい．$x_i + z_i - w_i$ はエピローグに残るタスク量であって非負であり，この式は常に成立する．

しかし，式 (5) が等号の場合，有限の k では $R_k = 1$ が得られない．このような場合には $a = (x_i + z_i)/w_i' = w_i/w_i'$ のようになるので，式 (3) は

$$r_j \le a(1 - R_j)$$

と表され，$a < 1$ であれば $R_j + r_j = 1$ は決して達成されない．$a \ge 1$ なら問題ないが，これは $x_i + z_i \ge w_i'$ を意味するから，条件 (3) を考慮しなくてよい場合に相当する．よって $x_i + z_i < w_i'$ なるワーカに対して

$$x_i + z_i - w_i > 0 \qquad (6)$$

が要請される．これは $x_i - w_i$ が正なら常に成り立つ．もし $x_i - w_i = 0$ でも，$z_i > 0$ であれば成立する．

前述の通り，これらの条件を確認する必要があるプロセッサは，ワーカであって $x_i + z_i < w_i'$ を満たすものだけである．すなわち最初から $x_i < w_i'$ に絞り込んでおいてよい．もしこうなるワーカが一つもなければ $z_i = 0$ で $r_0 = 1$ が達成されることになり，**単一ラウンドでスケ**

ジューリングが完了し，それが最適となる．

条件 (4) と条件 (6) は $z_i > 0$ であれば満たされる．また，$w_i' > 0$ かつ $x_i = 0$ または $x_i - w_i = 0$ となるワーカがなければ $z_i = 0$ も可能である．つまり任意の $z_i > 0$ に対してスケジュールが可能であるが，これは $\alpha = 0$ の場合の makespan $T + \max\{z_i\gamma_i\}$ が理論的下限 T にいくらでも近づけられること，すなわち漸近最適性を保存していることを意味している．

3.5 スケジューリングの改良

これまでの議論では，通信の立ち上がり遅延 α を無視してきた．$\alpha > 0$ とするとラウンド数 k が多くなるほどオーバーヘッドになってしまう．そこで，ラウンドサイズはできるだけ大きく取るべきである．すなわち

$$r_j = \min\{\frac{x_i + z_i}{w_i'} - R_j\frac{w_i}{w_i'}\} \tag{7}$$

と取るのがよい (ただし最後のラウンドはもちろん $r_{k-1} = 1 - R_{k-1}$ で定める)．

式 (7) で決まる r_j は z_i が大きいほど大きくなり，ラウンド数を減らす方向に働く．しかし，z_i を大きくするとその分プロローグの終了時刻が遅れることになる．Makespan を考える上ではアイドル時間の最大値のみが問題となるから，すべてのプロセッサに共通にして

$$\zeta_i(= z_i\gamma_i) = \zeta \tag{8}$$

とおいてよい．

すなわち，ζ を与えると式 (8) と式 (7) から一つのスケジュールが得られ，その makespan が決まる．これを最小にするような ζ を求めればよいということになる．

3.6 最適なアイドル時間 ζ の探索

最後にアイドル時間 ζ の最適化について考える．

まず，考慮すべきアイドル時間 ζ の範囲について考える．前節の考察により $\zeta > 0$ では常にスケジューリングが可能であり，場合によっては $\zeta = 0$ も可能である．$x_i + z_i < w_i'$ となるプロセッサがなければ 1 ラウンドで完了するが，こうなる最小の ζ は $\max\{\gamma_i(w_i' - x_i)\}$ である．すなわち

$$0 \le \zeta \le \max\{\gamma_i(w_i' - x_i)\}$$

の範囲で ζ を探索すれば十分である．

以下では，提案手法による所要時間が

$$T(\zeta) = T + Ak(\zeta) + \zeta \tag{9}$$

となるものと仮定する．ここで A は α が 0 でないことによるラウンドあたりのオーバーヘッドで定数，$k(\zeta)$ は ζ により決まるラウンド数である．右辺第 3 項の ζ はアイドル時間の最大で，エピローグの終了時刻の遅れである．

ラウンド数 $k(\zeta)$ が，単調非増加の整数値階段 (区分的定数) 関数であることは容易に確認できる．そこで，

$$\zeta_{min}(k) = \min\{\zeta : k(\zeta) = k\}$$

と定義する．すなわち，$\zeta \in [\zeta_{min}(k), \zeta_{min}(k-1))$ に対して $k(\zeta) = k$ となる．

式 (9) から k の値が同じであれば ζ が小さいほど makespan $T(\zeta)$ が小さくなるのは明らかである．よって，makespan を最小にするラウンド数 (**最適ラウンド数**) k_{opt} がわかれば，$\zeta_{opt} = \zeta_{min}(k_{opt})$ により makespan 最小のスケジュールが得られる．

ここで，次の定理がなりたつ．

定理 Makespan が式 (9) に従うとき，最適ラウンド数 k_{opt} は集合 $\{k(nA) : n = 0, 1, 2, \cdots\}$ に含まれる．

証明 $k(\zeta) = k_{opt}$ となるような ζ の範囲の幅が A 以上あることを示せばよい．そうすれば A おきに ζ をサンプルすれば必ず k_{opt} を踏む．

証明は背理法による．$\zeta_{min}(k_{opt} - 1) < \zeta_{min}(k_{opt}) + A$ と仮定すると，$\zeta' < \zeta_{min}(k_{opt}) + A$ かつ $k(\zeta') \le k_{opt} - 1$ となる ζ' が存在する．すると

$$T(\zeta') \le T + Ak_{opt} - A + \zeta' \le T(\zeta_{min}(k_{opt}))$$

となり，k_{opt} の最適性に矛盾する．■

$T(nA) = T + (k(nA) + n)A$ であるから，$n = 0, 1, 2, \cdots$ に対して $k_n = k(nA)$ を求め，$k_n + n$ が最小となる n を探せばよい．この際，任意の ζ に対して $k(\zeta) \ge 1$ であるから，見つかっている $\min\{k_n + n\}$ 以上の n は調べなくてもよい．

最小値を与える n が複数ある場合には，それぞれに対して $\zeta_{min}(k_n)$ を求めて makespan が最小になるものを選ぶ．この部分についてはもう少し工夫の余地があるが，この単純な方法よりも最悪計算量が少ない手法が得られるかどうかは不明である．

4. シミュレーション結果

本節では修正版のスケジューリングを従来手法と比較する．但し，アイドル時間 ζ の最適化は完全な実装になっておらず，近似的な最適解に留まっている．

表 1 にシミュレーションにおける各プロセッサの速度と初期状態で持っている負荷量を示す．単位タスクの計算量を 100k クロック，通信量を 1k バイトとし，通信性能をスループット 800 Mbps，遅延 100 μs とした．これらはすべて以前の報告[3] にあわせたものである．

表 2 には makespan を秒単位で示している．従来の漸近最適スケジューリングでもすでにすべての問題設定で他の手法 (詳細は原報告?) を参照) に勝っているが，改良手法ではさらに性能が改善していることがわかる．実は改良手法では ProcSet3 Load4 以外はすべて $\zeta = 0$ で 1 ラウンドで終了している．これらのものは当然ながら下限に非常に近い makespan となっているが，提案手法はこのような問題をうまく検出しているということができる．

図 2 は ProcSet3 Load4 における ζ と makespan の関係を示している．選ばれたパラメタは $\zeta = 3.89 \times 10^{-4}$, $k = 5$ となっている．ラウンドサイズは $r_0 = 5.19 \times 10^{-4}$, $r_1 = 0.00338$, $r_2 = 0.0220$, $r_3 = 0.144$, $r_4 = 0.830$ であり，ラウンドごとにほぼ 6.5 倍になっている．これは，式 (7) の最小値を与える i がいつも同じであれば $r_j = c^j r_0$ となることによっている (c は定数)．なお，この例では

表 1 シミュレーション上のプロセッサ性能と初期負荷
Table 1 Processor performance and initial load in the simulation

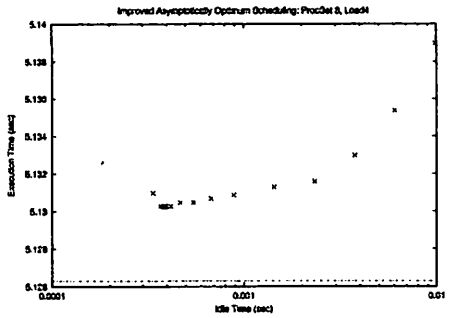
	ProcSet1	ProcSet2	ProcSet3	Load1	Load2	Load3a	Load3b	Load4
Proc0	1.5GHz	0.8GHz	2.5GHz	75k	40k	125k	25k	120k
Proc1	1.5GHz	1.0GHz	2.5GHz	75k	50k	125k	25k	120k
Proc2	1.5GHz	1.2GHz	2.5GHz	75k	60k	125k	25k	120k
Proc3	1.5GHz	1.4GHz	2.5GHz	75k	70k	125k	25k	120k
Proc4	1.5GHz	1.6GHz	0.5GHz	75k	80k	25k	125k	120k
Proc5	1.5GHz	1.8GHz	0.5GHz	75k	90k	25k	125k	0k
Proc6	1.5GHz	2.0GHz	0.5GHz	75k	100k	25k	125k	0k
Proc7	1.5GHz	2.2GHz	0.5GHz	75k	110k	25k	125k	0k

通信スループット 800 Mbps, 通信遅延 100 μs. 単位タスクの計算量 100k クロック, 通信量 1k バイト.

表 2 シミュレーション上の makespan
Table 2 Makespan obtained by the simulation

マシン	タスク	比例分割	動的負荷分散	従来提案	改良提案	理論的下限
ProcSet1	Load2	5.8007	5.2581	5.2205	5.2003	5.2000
ProcSet1	Load3	5.5001	5.9789	5.5001	5.5001	5.5000
ProcSet1	Load4	7.2507	6.0750	5.5607	5.5425	5.5422
ProcSet2	Load1	5.8007	5.6475	5.2150	5.1954	5.1951
ProcSet2	Load3a	7.8007	6.8730	5.7125	5.6953	5.6951
ProcSet2	Load3b	6.2007	5.4664	5.3155	5.2950	5.2947
ProcSet2	Load4	8.0007	6.8705	5.7777	5.7492	5.7489
ProcSet3	Load1	5.5001	8.2362	5.5001	5.5001	5.5000
ProcSet3	Load2	7.8007	8.6559	5.7167	5.7004	5.7000
ProcSet3	Load3b	6.0001	13.2906	6.0001	6.0001	6.0000
ProcSet3	Load4	5.9507	5.8080	5.1770	5.1303	5.1263

図 2 ProcSet3 Load4 での ζ と makespan の関係
Fig. 2 ζ vs makespan for ProcSet3 Load4



ラウンドサイズが順次大きくなっているが，ラウンドサイズが順次小さくなるスケジュールが最適となるような問題を構成することもできる．また，式 (7) が R_j に関して上に凸な関数なので，ラウンドサイズが徐々に大きくなり，途中から逆に次第に小さくなるスケジュールが最適となるような問題もあるものと考えられる．

5. ま と め

本稿では，Multi-Master Divisible Load 問題に対する漸近最適スケジューリングの改良について述べた．従来手法ではタスク量が無限に多くなるときに最適解に漸近するという性質があったが，提案手法では有限のタスク量・有限のラウンド数を意識してスケジューリングを改良した．

実験の結果，問題設定によっては 1 ラウンドで理論的下限に近い makespan が得られることが示された．提案手法はこのような問題をうまく検出できる．またそうでない場合も従来手法に比べてずっと下限に近い makespan を得た．ラウンドサイズは一定の割合で大きく（または小さく）なることが観測されたが，これは提案手法が固定ラウンドサイズからの解放も実現したことを示している．

今回の成果により，一様で独立なタスク，均一なネットワークという条件下での MMDL 問題に対する一通りの解答を得たと考えられる（まだ少し改善の余地はある）．今後はタスクやネットワークが不均一な場合に対する MMDL 問題の漸近最適スケジューリングの研究を進めたいと考えている．また，実用的な場面への応用にも取り組みたい．

謝 辞

本研究の一部は文部科学省 21 世紀 COE プログラムと科学研究費の補助を受けています．

参 考 文 献

1) 須田礼仁,「Multi-master divisible load model における漸近最適スケジューリング」, 情報処理学会研究報告, 2004-ARC-157/2004-HPC-97, pp. 97–102.
2) 須田礼仁,「マルチクラスタ環境での MMDL 漸近最適スケジューリング」, 情報処理学会研究報告, 2004-HPC-99, pp.103–108.
3) 富さやか, 須田礼仁,「Multi-master divisible load モデルに対する漸近最適スケジューリングの評価」, 情報処理学会研究報告, 2005-HPC-102, pp. 51–56.
4) 富さやか, 須田礼仁,「Multi-master divisible load の漸近最適スケジューリングの実機への実装」, 情報処理学会研究報告, 2005-HPC-103, pp. 37–42.