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

2007－ARC－172 (26)
2007－HPC－109 (26)
2007／3／2

動的にマスタ数を増加するマスタ・ワーカ方式の提案

水 谷 泰 治†

本稿では，マスタ・ワーカ（MW）型の並列プログラムにおいて，プログラムの実行状況に応じてマスタ数を増加させる方式を提案する．提案方式では，マスタに到着しているワーカからのメッセージの数に着目してマスタが過負荷状態にあるか否かを判定する．マスタが過負荷状態にあると判断した場合，そのマスタが管理するワーカ集合の中から新たなマスタを選択し，その新しいマスタにワーカとタスクを分配して各マスタが独立に処理を進める．このようなマスタの増加を繰り返すことで，MW 方式のスケーラビリティの向上を図る．評価実験により，単一のマスタによる MW 方式では性能が飽和する場合において，提案方式を適用することで性能の飽和を回避できることを確認した．

Master-Worker Paradigm with Dynamically Increasing Masters

YASUHARU MIZUTANI†

This paper proposes a method that is capable of raising the scalability of master-worker parallel programs by dynamically increasing the number of masters. In the method, a master selects a new master from its workers and assings workers and tasks to the new master, when the master detects that its workload is too high. We define the workload of the master as the number of messages which arrive to the master. The experimental result shows that the method avoids the performance saturation by distributing the workload of the master to other masters which are dynamically generated.

1. はじめに

マスタ・ワーカ（MW）方式とは，並列計算の際に利用可能なプロセッサ集合をマスタとワーカに分割し，マスタはワーカへのタスク割り当てとその処理結果の管理を担当し，ワーカは割り当てられたタスクの処理を担当する並列実行方式である．MW 方式は，動作が単純でありながら動的な負荷分散によって高い性能を得ることができるため，様々な分野におけるアプリケーションの並列化に利用されている[5),10)]．

MW 方式には，性能に影響するパラメータとしてタスク粒度，ワーカ数，マスタ数がある．これらのパラメータの適切な値は対象とする問題や実行環境に依存し，不適切な値を設定した場合，マスタへの通信の集中などによってプログラム全体の性能が低下する．近年ではグリッド環境[9)]に代表されるように，実行環境の大規模化と多様化が進んでおり，このような環境で良い性能を得るためには MW 方式の性能に影響するパラメータの値を適切に設定することが重要である．

従来，MW 方式ではマスタ数を 1 に固定することが多かった．しかし，マスタ数が 1 の場合，ワーカ数の増加に対する性能のスケーラビリティが制限される[4)]．そのため，近年，複数のマスタを用いる MW 方式の研究が行われている[1),3)]．これらの方式では，マスタの個数はプログラムの実行開始から終了まで固定的である．しかし，各タスクの実行時間の分布や実行環境によっては，適切なマスタ数がプログラムの実行時に変化する可能性もある．また，一般に適切なマスタ数は，実行環境や対象問題に依存するため，これらが変化する度にマスタ数を適切に設定する必要がある．

そこで，本研究では，様々な環境でより柔軟に，かつ効率良く MW 方式を利用可能とすることを目的に，プログラムの実行時にその実行状況に応じて自動的にマスタ数を増加させる MW 方式を考案する．提案方式では，マスタに到着しているワーカからのメッセージ数に基づいて，マスタを増加するか否かを判定する．

以降では，まず 2 章で MW 方式の性能改善に関する関連研究について述べる．次に，3 章で提案する MW 方式について述べる．その後，4 章で提案方式の性能を評価し，5 章でまとめを述べる．

2. 関連研究

MW 方式に基づくプログラム（MW プログラム）の性能に影響を与えるパラメータの値を適切に調整するための研究がいくつかある[1)〜3),7),8),11),12)]．

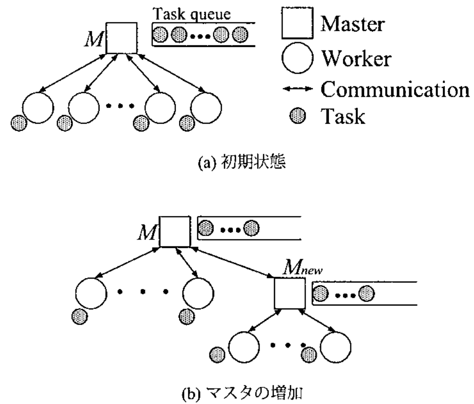
† 大阪工業大学情報科学部
Faculty of Information Science and Technology, Osaka Institute of Technology

文献 2)，7)，8)，11) では，各イタレーションの実行時間が不規則であるような DOALL 型ループの断片をタスクとし，プロセッサに割り当てるタスクの大きさをプログラムの実行時に調整する方式について述べている．これらの方式は，対象を MW 方式には限定していないが，多くの研究が MW 方式に適用してその方式の性能を評価している．文献 7)，8) では，プログラムの実行開始時にはタスクサイズを大きくし，実行が進むにしたがって一定の規則でタスクサイズを小さくする．開始時にタスクサイズを大きくすることで通信のオーバヘッドの削減を図り，終了時にタスクサイズを小さくすることで，各ワーカの負荷の均一化を図っている．また，文献 11) の方式では，プログラムの実行時に各タスクの実行時間に関する統計情報を取得し，その情報を元にタスクサイズを動的に決定する．

一方，MW プログラムの実行時にワーカ数を調整する方式も提案されている．文献 12) では，MW プログラムの動作をモデル化し，実行環境の性能とタスクの計算時間から最適なワーカ数を算出するための式を導き出し，プログラムの実行時に収集した情報から，ワーカ数を実行時に調整するツールを提案している．

一般に MW 方式ではマスタ数を 1 に固定することが多いが，この場合，マスタが性能ボトルネックとなり，ワーカ数の増加に対するスケーラビリティが制限される．この問題に対して，マスタを複数にすることで，マスタの負荷を分散させ，スケーラビリティの向上を実現する研究もある[1),3)]．文献 1) では，グリッド環境において MW プログラムの性能を向上させるために，複数のマスタを用意し，それらを階層的に管理することで，マスタの負荷を分散している．また，文献 3) では，グリッド環境において最適なスループットを得るためのマスタの配置決定手法を理論的な立場から考案し，マスタを複数用意することの有用性をシミュレーションによって示している．

このようにマスタを複数にした MW 方式の研究は行われているが，現状ではマスタ数はプログラムの実行前に決めておかなければならない．しかしながら，プログラムの開始時に決めたマスタ数が，必ずしもプログラムの実行全体を通じて適切であるとは限らない．例えば，PC グリッドのようにプログラムの実行中にプロセッサの計算能力が変化する環境では，プログラムの実行開始時から計算能力が変化した場合に，実行開始時に設定したパラメータ値が適切であるとは限らない．近年では，このような環境の利用も増加しており[6),13)]，実行状況に応じてマスタ数などのパラメータ値を自動的に調整することが重要であると考える．



(a) 初期状態

(b) マスタの増加

図 1 マスタの動的な増加

3. 動的にマスタを増加する MW 方式

本章では，プログラムの実行時にマスタ数を増加させる方式について述べる．

3.1 マスタの増加

提案方式では，プログラムの実行時にマスタが過負荷状態（3.2 節にて後述）であるか否かを判定し，過負荷状態であると判定した場合にマスタを増加させる．**図 1** に，提案方式におけるマスタの増加の様子を示す．提案方式では，まずマスタ数を 1 としてプログラムの実行を開始する（図 1(a)）．ここで，マスタ M は全タスクを保持しているものとする．その後，M がタスクの割り当てに関して過負荷状態にあるか否かを一定の間隔で判定しながら，MW 方式の動作を進める．このとき，M が過負荷状態であると判定した場合，M は自身が管理するワーカ集合の中から新たなマスタ M_{new} を 1 つ選択し，M_{new} に M が持つタスク集合とワーカ集合の一部を送信する．ここで，M_{new} から見て M を親マスタと呼び，M から見て M_{new} を子マスタと呼ぶ．また，マスタが直接管理するワーカと子マスタをまとめて子と呼ぶ．その後，M と M_{new} はそれぞれが互いに素なワーカ集合を管理し，独立に MW 方式の動作を進める（図 1(b)）．以降，各マスタは，同様の判定とマスタの増加を行なう．このように，提案方式では，マスタが過負荷状態にある場合にはマスタの数を増やし，タスク割り当てに関する負荷の分散を図る．これにより，MW 方式の性能ボトルネックを解消する．

3.1.1 マスタの増加の抑制

マスタ数が過剰となった場合，タスクを処理するワーカが少なくなり，プログラム全体の性能が低下する可能性がある．そこで，マスタを増加したときに

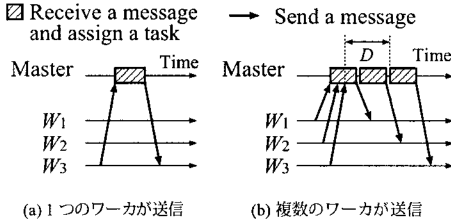


図 2 マスタへのメッセージの到着

M または M_{new}の子の数が 1 以下になる場合，すなわち，図 1(a) においてワーカ数が 3 以下である場合，マスタの増加を抑制する．MW 方式において子の数が 1 の場合，子の間で動的に負荷を分散するという MW 方式の長所を発揮できなくなるためである．

3.2 過負荷状態の判定

マスタが過負荷状態であるとは，単位時間にマスタが割り当てることができるタスクの数よりも，マスタに届く処理結果の数の方が多い状態をいう．

提案方式では，マスタがワーカに 1 つのタスクを割り当てたときにマスタに既に到着しているメッセージの数を用いて，マスタが過負荷状態であるか否かを判定する．この根拠を，**図 2** を用いて説明する．

まず，図 2(a) のように，ワーカ W_3 からマスタに処理結果を送信したときに，他のワーカからのメッセージがマスタに到着していない場合を考える．このとき，マスタは W_3 からのメッセージを直ちに受信し，次のタスクを W_3 に割り当てることができる．一方，図 2(b) のように，3 つのワーカ W_1, W_2, W_3 がこの順序でほぼ同じ時刻にマスタに処理結果を送信した場合，最初に到着した W_1 からのメッセージに対しては，マスタは直ちに受信するが，W_2 と W_3 からのメッセージの受信は W_1 へのタスク割り当てが完了するまで待たされる．例えば，図 2(b) では，W_3 からのメッセージがマスタに到着した後，マスタがそのメッセージに対する受信を始めるまでに D の遅延が生じる．このような状態が継続している場合，MW プログラムは過負荷状態であり，ワーカ数を増加しても性能向上は見込めない．このとき，W_1 や W_2 に対するタスク割り当てを完了した直後の時点で，マスタに到着しているメッセージ数は 1 以上であることがわかる．

以上より，提案方式では，タスクの割り当てが完了したときにマスタに到着している平均メッセージ数が 1 以上の場合，マスタが過負荷状態にあると判定する．

3.3 マスタ間のタスク割り当て

動的に増加したマスタ M_{new}は，M_{new}が管理する全てのタスクを完了したとき，親マスタにタスクの処理結果を送信し，M_{new}と M_{new}が管理する全ワーカを親マスタのワーカに戻す．このとき，早くに処理を完了したワーカは，他の全ワーカの処理が完了し，M の親マスタにタスクを割り当てられるまでアイドル状態となり，ワーカの稼働率が低下する．

そこで，このアイドル時間を短縮するために，M が保持するタスクの数が少なくなってきたとき，隣接するマスタにタスク集合の追加割り当てを要求する．これにより，M へのタスク集合の割り当てと M に残っているタスクの処理をオーバラップすることができ，ワーカがタスク割り当てを待つ時間が短くなると期待できる．このオーバラップ手法は既に提案されており[14]，その有用性が確認されている．

3.4 提案方式のアルゴリズム

これまでの説明をまとめると，マスタとワーカのアルゴリズムはそれぞれ**図 3** と**図 4** のようになる．図 3 の 14 行目では，3.2 節の方法により，過去 n 回のタスク割り当て後の平均到着メッセージ数が 1 以上であるか否かによって，マスタが過負荷状態であるか否かを判定する．マスタが過負荷状態であると判定した場合，マスタのアルゴリズムに必要な入力データをワーカに送信する．この送信に対応するワーカの処理は，図 4 の 6 行目に対応する．

また，3.3 節で述べたマスタ間のタスクの移動は，図 3 の 22 行目に対応する．ここでは，マスタがその時点で最後のタスクをワーカに割り当てたとき，隣接するマスタにタスク集合の割り当てを要求する．これにより，隣接マスタとの通信と，次の処理結果がマスタに到着するまでの通信および計算をオーバラップすることが期待できる．

5 行目で定義している rflag は，隣接するマスタにタスク集合を要求中であり，要求中にマスタのアルゴリズムを終了させないために使用する．

4. 評価実験

本節では，提案方式によるマスタの増加の効果を確かめるための実験を行なう．

4.1 実験環境

実験は 18 台の PC からなる PC クラスタを用いて行なった．各 PC は，イーサネットで接続されており，通信バンド幅は 1Gb/s である．また，各 PC は CPU として Pentium4 3GHz をもつ．対象問題として，並列プログラムのベンチマークに用いられるマンデルブロ集合の画像描画プログラムを用いた．この問題を MPI を用いたメッセージ通信プログラムとして実装した．本実験で用いた MPI の実装は MPICH である．マスタへの到着メッセージ数は，`MPI_Iprobe` 関数を用い，全ワーカからメッセージが到着しているか否かを調べることで取得した．

入力: ワーカ集合$\mathcal{W}$，親マスタ m，タスクキュー Q_i，　**出力:** タスクの計算結果を格納したキュー Q_o

1　● 待機状態のワーカ集合$\mathcal{W}_i \leftarrow \mathcal{W}$，計算状態のワーカ集合$\mathcal{W}_r \leftarrow \emptyset$
2　● $Q_o \leftarrow \emptyset$，子マスタ集合$\mathcal{M} \leftarrow \emptyset$，タスク要求の送信先候補集合$\mathcal{R} \leftarrow \{m\}$
3　● $\mathcal{W}_i$中の各ワーカに対して，Q_i から 1 個のタスクをデキューし，割り当てる．
4　● タスクを割り当てたワーカを$\mathcal{W}_i$から$\mathcal{W}_r$へ移動する．
5　● タスク要求フラグ rflag $\leftarrow 0$
6　○ $|\mathcal{W}_r| + |\mathcal{M}| > 0$ かつ rflag $= 1$ である間，以下を繰り返す
7　　　● メッセージの到着を待つ．到着したメッセージの種別を k，送信元のプロセスを p とする．
8　　　▷ k が「計算結果報告」ならば
9　　　　　● 受信データを Q_o へエンキューする．
10　　　　　▷ $p \in \mathcal{W}_r$ ならば
11　　　　　　　▷ Q_i が空ならば
12　　　　　　　　　● $\mathcal{W}_r \leftarrow \mathcal{W}_r - \{p\}$，$\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{p\}$
13　　　　　　　そうでないならば，
14　　　　　　　　　▷ M が過負荷状態かつ $|\mathcal{W}_r| \geq 4$ ならば
15　　　　　　　　　　　● $\mathcal{W}_r \leftarrow \mathcal{W}_r - \{p\}$，$\mathcal{M} \leftarrow \mathcal{M} \cup \{p\}$，$\mathcal{R} \leftarrow \mathcal{R} \cup \{p\}$
16　　　　　　　　　　　● $\mathcal{W}_r$を互いに素な集合$\mathcal{W}_a$と$\mathcal{W}_b$に分割し，$\mathcal{W}_a$を新たな$\mathcal{W}_r$とする
17　　　　　　　　　　　● Q_i から複数のタスクをデキューし，それらを Q_j にエンキューする
18　　　　　　　　　　　● p に$\mathcal{W}_b, M, Q_j$ を送信する．メッセージの種類を「マスタ起動」とする．
19　　　　　　　　　そうでないならば，
20　　　　　　　　　　　● Q_i から 1 個のタスクをデキューし，ワーカ p に割り当てる．
21　　　　　　　　　　　▷ Q_i が空ならば
22　　　　　　　　　　　　　● $\mathcal{R}$の中の 1 つに「タスク要求」のメッセージを送信する．
23　　　　　　　　　　　　　● rflag $\leftarrow 1$
24　　　　　そうでなく，$p \in \mathcal{M}$ならば
25　　　　　　　● 計算結果を受信し，Q_o にエンキューする
26　　　　　　　● 子マスタ p が管理するワーカ集合$\mathcal{W}_t$を p から受信する
27　　　　　　　● $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \mathcal{W}_t \cup \{p\}$，$\mathcal{M} \leftarrow \mathcal{M} - \{p\}$，$\mathcal{R} \leftarrow \mathcal{R} - \{p\}$
28　　　　　　　● $\mathcal{W}_i$中の各ワーカに対して，Q_i から 1 個のタスクをデキューし，割り当てる．
29　　　　　　　● タスクを割り当てたワーカを$\mathcal{W}_i$から$\mathcal{W}_r$へ移動する．
30　　　そうでなく，k が「タスク割当」ならば
31　　　　　● 受信データを Q_i へ追加する
32　　　　　● $\mathcal{W}_i$中の各ワーカに対して，Q_i から 1 個のタスクをデキューし，割り当てる．
33　　　　　● タスクを割り当てたワーカを$\mathcal{W}_i$から$\mathcal{W}_r$へ移動する．
34　　　　　● rflag $\leftarrow 0$
35　　　そうでなく，k が「タスク要求」ならば
36　　　　　▷ Q_i が空ならば
37　　　　　　　● p にメッセージ「タスク要求の拒否」を送信
38　　　　　そうでないならば，
39　　　　　　　● Q_i から複数のタスクをデキューし，p に送信する．メッセージ種別は「タスク割当」とする．
40　　　そうでなく，k が「タスク要求の拒否」ならば
41　　　　　● $\mathcal{R} \leftarrow \mathcal{R} - \{p\}$
42　　　　　● rflag $\leftarrow 0$
43　　　　　▷ $\mathcal{R}$が空でないならば
44　　　　　　　● $\mathcal{R}$の中の 1 つに「タスク要求」のメッセージを送信する．
45　　　　　　　● rflag $\leftarrow 1$
46　▷ 親マスタ m が存在するならば
47　　　● $\mathcal{W}_i$の全てのワーカに「動作終了」のメッセージを送信する．
48　そうでないならば，
49　　　● m に Q_o と W_i を送信する．メッセージの種類は「計算結果報告」とする

図 3　マスタ M のアルゴリズム

図 3 の 16, 17 行目では，それぞれワーカ集合とタスク集合の半分を新しいマスタに分配するものとした．また，3.4 節における n の値は，使用プロセッサ数の倍の値とした．

MW プログラムの性能は，タスク粒度や通信性能によってはマスタ数が 1 の場合でも良い性能を得ることができる．しかし，本実験の目的は提案方式の効果を確認することであるので，マスタ数が 1 のときにマスタを過負荷状態となるようにタスク粒度と通信命令のオーバヘッドを調整した．

4.2　マスタの増加の効果

図 5 に，提案方式と従来方式の性能の比較を示す．

入力: なし，　**出力:** なし

1　● メッセージの到着を待つ．到着したメッセージの種別を k, 送信元のプロセスを p とする
2　○ k が「動作終了」でない間，以下を繰り返す
3　　▷ k が「タスク割当」ならば
4　　　● 受信したタスクを処理する
5　　　● 処理結果を p へ送信する．メッセージの種類は「計算結果報告」とする．
6　　そうでなく，k が「マスタ起動」ならば
7　　　● ワーカ集合 $\mathcal{W}$, タスクキュー Q_i を受信する
8　　　● $(\mathcal{W}, p, Q_i)$ を入力とし，マスタとしての動作を開始する（図 3）
9　　　● 計算結果を p へ送信する．メッセージの種類は「計算結果報告」とする．
10　　　● $\mathcal{W}$を p へ送信する．
11　　● メッセージの到着を待つ．到着したメッセージの種別を k, 送信元のプロセスを p とする

図 4　ワーカ W のアルゴリズム

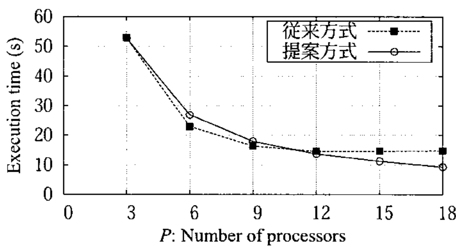
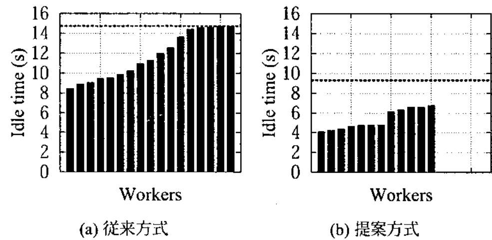
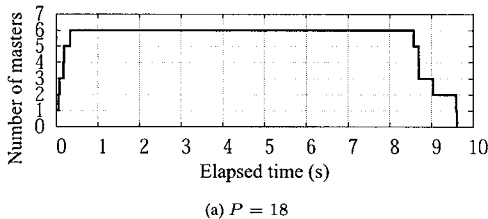


図 5　提案方式と従来方式の比較

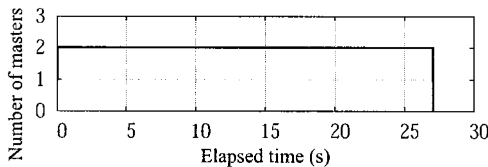


(a) 従来方式　(b) 提案方式

図 7　$P = 18$ における各ワーカのアイドル時間の比較



(a) $P = 18$



(b) $P = 6$

図 6　提案方式におけるマスタ数の変化

ここで，従来方式とは，マスタ数を 1 に固定した MW 方式を表す．各実行時間は，9 回計測した中の中央値を示している．また，**図 6** に，提案方式を実行したときのマスタ数の変化を示す．図 5 より，従来方式では $P = 12$ において性能が飽和しているのに対し，提案方式では P の増加に従って実行時間が短くなっている．これは，提案方式ではプログラムの実行時にマスタ数が増加し，ワーカへのタスク割り当てが効率良く進められるようになったためであると考えられる．図 6(a) では，プログラムの実行を開始してから 0.33 秒が経過した時点で，マスタ数が 6 まで増加している．このように，提案方式では，プログラムの実行時にマスタ数を増加させることで，マスタの負荷を分散し，プログラム全体の性能を向上させることができる．

図 5 では，$P = 6$ のときには提案方式の実行時間は従来方式の実行時間よりも 18%長い．これは，提案方式ではワーカの中から新たなマスタを選ぶことにより，タスクを処理するプロセッサが減少したためである．$P = 6$ のとき，従来方式ではワーカ数が 5 である．一方，図 6(b) より，提案方式では $P = 6$ のときにマスタ数は 2，すなわちワーカ数は 4 であり，従来方式と比べてワーカ数が 20%少ない．この差は，$P = 6$ にときの提案方式と従来方式の実行時間の差と概ね一致する．これより，提案方式はマスタの増加によってマスタの過負荷状態を解消できる反面，ワーカ数が減少することによって従来方式よりも性能が低下する場合があることがわかる．したがって，マスタを増加する際に，マスタの負荷分散による性能向上とワーカ数の減少による性能低下を見積もって，マスタを増加するか否かを判断する機構を考案する必要がある．

次に，提案方式と従来方式におけるワーカのアイドル時間を比較する．アイドル時間とは，マスタに処理結果を送信してから次のタスクを受信するまでの時間

である．図7に，$P = 18$ におけるワーカ毎の合計アイドル時間の比較を示す．ここで，各ワーカの合計アイドル時間は昇順に並べてある．また，図中の破線はプログラム全体の実行時間を表す．なお，図7(b)では，実行時にマスタに変化した5つのワーカの合計アイドル時間は記載していない．従来方式と提案方式の平均アイドル時間は，それぞれ11.7秒と5.34秒であり，提案方式の方が短い．これは，マスタが増加することでタスク割り当ての能力がプログラム全体として増加し，ワーカへ割り当てられるタスクが増加したためであると考えられる．図5において，従来方式と比較して提案方式の実行時間が短くなった理由は，このアイドル時間の短縮によるものである．なお，提案方式においても実行時間の約半分はアイドル時間であるが，これは4.1節で述べた調整により，相対的にタスク粒度が細かくなっているためである．

このように，提案方式はプログラムの実行時にマスタを増加することで，ワーカのアイドル時間を短縮でき，MW方式の性能飽和を解消できる．さらに，プログラムの実行開始時にユーザがマスタ数を調整することなく上述の効果を得られることから，提案方式はスケーラビリティと利便性の点で有用であると考える．

5. おわりに

本稿では，プログラムの実行時にマスタが過負荷状態であると判定したときに，動的にマスタ数を増加させるMW方式を提案した．提案方式では，マスタが1つのタスクの割り当てを終えた時点において，マスタに到着している平均メッセージ数が1以上の場合，マスタが過負荷状態にあると判定する．また，評価実験により，単一のマスタによるMWプログラムでは性能が飽和する場合において，提案方式を適用することで性能の飽和を回避できることを確認できた．これより，提案方式はスケーラビリティと利便性の点で有用であると考える．今後の課題として，より大規模な並列計算環境における詳細な評価があげられる．また，MW方式の他のパラメータを自動調整する方式と併用した場合の性能評価があげられる．

謝辞　本研究の一部は，科学研究費補助金若手研究(B)(18700072)の補助による．

参考文献

1) Aida, K. and Natsume, W.: Distributed Computing with Hierarchical Mater-worker Paradigm for Parallel Branch and Bound Algorithm, *Proc. 3rd IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid'03)*, pp.156–163 (2003).

2) Banicescu, I. and Velusamy, V.: Load Balancing Highly Irregular Computations with the Adaptive Factoring, *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS'02)*, pp.87–98 (2002).

3) Banino, C.: Optimizing Locationing of Multiple Masters for Master-Worker Grid Applications, *Proc. 7th Int'l Conf on Applied Parallel Computing (PARA 2004)*, pp.1041–1050 (2004).

4) Banino, C.: Scalability Limitations of the Master-Worker Paradigm for Grid Computing, *Workshop on state-of-the-art in scientific computing* (2004).

5) Brockington, M. G. and Schaeffer, J.: APHID: Asynchronous Parallel Game-Tree Search, *J. of Parallel and Distributed Computing*, Vol.60, No.2, pp. 247–273 (2000).

6) Chien, A., Calder, B., Elbert, S. and Bhatia, K.: Entropia: architecture and performance of an enterprise desktop grid system, *J. of Parallel and Distributed Computing*, Vol.63, No.5, pp.597–610 (2003).

7) Chronopoulos, A. T., Benche, M., Grosu, D. and Andonie, R.: A Class of Loop Self-Scheduling for Heterogeneous Clusters, *Proc. 3rd IEEE Int.l Conf. on Cluster Computing (Cluster'01)*, pp. 282–291 (2001).

8) Chronopoulos, A.T., Penmatsa, S. and Yu, N.: Scalable Loop Self-Scheduling Schemes for Heterogeneous Clusters, *Proc. 4th IEEE Int'l Conf. on Cluster Computing (Cluster'02)*, pp.353–359 (2002).

9) Foster, I. and Kesselman, C.(eds.): *The Grid 2: Blueprint for a New Computing Infrastructure, second ed.*, Morgan Kaufmann Publishers, San Mateo, CA (2003).

10) Kawasaki, Y., Ino, F., Mizutani, Y., Fujimoto, N., Sasama, T., Sato, Y., Tamura, S. and Hagihara, K.: A High Performance Computing System for Medical Imaging in the Remote Operating Room, *Proc. 10th Int'l Conf. High Performance Computing (HiPC 2003)*, pp.162–173 (2003).

11) Kejariwal, A., Nicolau, A. and D., C.: History-aware Self-Scheduling, *Proc. 35th Int'l Conf. on Parallel Processing (ICPP'06)*, pp.185–192 (2006).

12) Morajko, A., César, E., Caymes-Scutari, P., Mesa, J.G., Costa, G., Margalef, T., Sorribes, J. and Luque, E.: Development and Tuning Framework of Master/Worker Applications, Vol.5, No.3, pp.115–120 (2005).

13) Muraki, K., Kawasaki, Y., Mizutani, Y., Ino, F. and Hagihara, K.: Grid Resource Monitoring and Selection for Rapid Turnaround Applications, *IEICE Trans. Information and Systems*, Vol.E89-D, No.9, pp.2491–2501 (2006).

14) van Nieuwpoort, R.V., Kielmann, T. and Bal, H.E.: Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications, *Proc. 8th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'01)*, pp.34–43 (2001).