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

2005－HPC－102（9）
2005／6／2

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

富　さやか†　須田礼仁†

Multi-master divisible load (MMDL) モデルはヘテロ型クラスタにおける動的負荷分散やデータ再分散のためのモデルである。本研究では，通信性能が均一なヘテロ型クラスタ上での MMDL モデルに対する漸近最適スケジューリングをシミュレータ上で実装する。ここでは，通信と計算をオーバーラップすることはせず，計算の結果は収集しなくてよいものとした。この条件の下で，プロセッサ性能と負荷の初期割り当て量を変化させた時の漸近最適スケジューリングの性能を，動的負荷分散による方法とプロセッサ性能に合わせて負荷を割り当てる方法の 2 つの手法と比較し，その特性を分析する。

Evaluation of asymptotically optimum scheduling algorithms for multi-master divisible load model

Sayaka Tomi† and Reiji Suda†

Multi-master divisible load (MMDL) model is a model for dynamic load balancing and the data redistribution on a heterogeneous cluster system. In this paper, an asymptotically optimum scheduling algorithm for MMDL model on heterogeneous clusters of homogeneous communication performance is implemented on a simulator. The scheduling algorithm assumes no overlap of communication and computations, and no collection of the results. Under these conditions, the asymptotically optimum scheduling algorithm is compared with two other methods, dynamic load balancing and load allocation according to processor performance, for various processor performance and allocation of the amount of an initial load.

1. はじめに

MMDL (Multi-Master Divisible Load) は，1) において提案された，データの再分散のためのモデルである。負荷の均衡化に必要なデータ再分散を最適化するためには，プロセッサの通信所要時間によって割り当てるデータ量を調整しなければならない。

このことを考慮した研究として DLT (Divisible Load Theory)[3] と呼ばれる，マスター・ワーカー型の並列処理をモデル化したものがあげられる。一般に，データの再分散において，一度にまとめて転送するよりも分割して送る方が所要時間が短くて済むが，DLT では，このように分割して転送を行う手法を multi-installment あるいは multi-round と呼ぶ。DLT においてヘテロ環境を含む各種の条件の下でいくつかの応用も行われてきたが，マスター・ワーカーの枠組みは，多様な並列処理のあり方からすると適用範囲が狭い。

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

DLT は，タスクのモデルとしては最もシンプルであり，DLT のマスター・ワーカーという制約を解消したモデルが MMDL である。さらに，1) では，MMDL とともに，計算性能がヘテロでもよいクラスタ環境における漸近最適スケジューリングアルゴリズムが提案された。

本研究では，この漸近最適スケジューリングをシミュレータ上で実装し，既存手法との比較を行い，その有効性と課題を明らかにする。

2. Multi-master divisible load モデル

Divisible load problem では，多くの負荷を複数のプロセッサで分担して処理する。

本節では，MMDL のタスクのモデルを説明し，システムのモデルを設定する。

2.1 タスクのモデル

1) で提案された MMDL のモデルは以下の通りである。

複数のプロセッサがあり，初期状態でそれぞれのプロセッサにタスクが割り当てられている。タスクは相

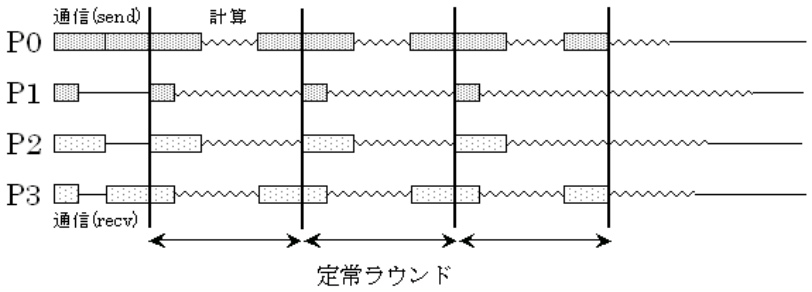

図 1 定常ラウンドスケジューリング
Fig. 1 Scheduling by repeating stable rounds

互に依存が無く，独立に処理することができ，一様で，どのプロセッサでも処理することができる。また，タスクは任意のサイズに切り分けることができ，タスクを移送する場合のメッセージサイズはタスクサイズに比例する。処理の結果は収集する必要はないものとする。

従来の divisible load theory[3] では，初期状態で全てのタスクがマスターと呼ばれる 1 つのプロセッサに置かれていて，他のプロセッサ（ワーカー）には割り当てられていない。これに対し，MMDL ではマスター・ワーカーという区別はなく，タスクを輸出するか輸入するかで結果的にマスターとワーカーに分かれることになる。

2.2 システムのモデル

システム側の問題設定として，今回は以下のように最も簡単なものを扱う。

プロセッサはクロスバースイッチにより接続されており，スループットはどのプロセッサ間でも同じで，単位サイズのタスクに対して β の時間がかかるとする。通信の立ち上がり遅延は無視する。プロセッサの計算性能は異なっても良く，第 i プロセッサ上で単位サイズのタスクを処理するために γ_i の時間がかかるとする。各プロセッサは通信と計算を同時に行うことができないとする。

ここで本質的なのは，通信性能が一様であり，通信の立ち上がり遅延が無視できることである。現実的には，通信の立ち上がり遅延については追加的なオーバーヘッドと見なすことにより，近似的な最適解を構成することが可能である。プロセッサの計算性能はヘテロでもよい。

3. 漸近最適スケジューリング

漸近最適スケジューリングでは，図 1 のように，通信と計算のパターンが同一になる「定常ラウンド」を連続的に反復するものとする。これにより，問題を定常状態にある各ラウンドのスケジューリングに帰着させることができる。[4]

タスクが任意のサイズに分割でき，通信と計算にかかる時間がタスクサイズに比例し，通信の立ち上がり遅延が無視できる，という仮定より，ラウンド内のスケジューリングは，タスクサイズを定数倍しても不変となる。よって，ラウンド数を R とすると，1 ラウンドの所要時間は T/R と表せる。立ち上がり遅延を含めた 1 ラウンドの所要時間は，ある定数 α を用いて $T/R + \alpha$ と表せる。最初のラウンドは通信のみ，最後のラウンドは計算のみを行うので，全体の所要時間は $T + T/R + \alpha R$ となる。よって，所要時間は $R = \sqrt{T/\alpha}$ のとき最小値 $T + 2\sqrt{T\alpha}$ をとる。ここで，α は定数なので，問題サイズを大きくすると所要時間は T に漸近する。

以上より，定常ラウンド 1 つ分の最適なスケジューリングを求め，それを反復することにより全体の所要時間を下限に近付ける。なお，通信の立ち上がり遅延がない場合，ラウンド数 $R \to \infty$ の極限で全体の所要時間は T となる。そこで実験では，この T を全体の所要時間の一つの下限として評価に利用する。

4. 実　　験

4.1 既存手法

比較手法として，比例分割と動的負荷分散の 2 つを扱う。

比例分割では，負荷はそれぞれのプロセッサ性能に応じて割り当てられる。負荷を多く持つものが master となって他のプロセッサに負荷を送信し，それぞれのプロセッサ上で計算が行われる。必要な通信は全て先に行い，通信終了後に計算を行う。

動的負荷分散では，ある一定の負荷量がブロックサイズとして決められ，それぞれのプロセッサはそのブロックサイズ毎に計算を行う。割り当てられた負荷を全て処理し終えたプロセッサは，その時点で負荷を最

表 1 プロセッサ性能の実験パターン
Table 1 Experiment patterns of the processor performance

	ProcSet1	ProcSet2	Procset3
Proc0	1500 MHz	800 MHz	2500 MHz
Proc1	1500	1000	2500
Proc2	1500	1200	2500
Proc3	1500	1400	2500
Proc4	1500	1600	500
Proc5	1500	1800	500
Proc6	1500	2000	500
Proc7	1500	2200	500

も多く持つプロセッサから，ブロックサイズ分の負荷を受け取る。今回は，各プロセッサが持つ負荷情報を通信するためのオーバーヘッドは考慮していない。

4.2 実験パターン

それぞれの手法について，プロセッサ性能の組み合わせと初期負荷量の分割のパターンを変えて，実行時間の変化をシミュレーションによって実験する。ここで，通信遅延は 0.0001 秒，通信速度のスループットは 800Mbps とし，単位仕事あたりの通信量は 1K バイト，計算量は 100K クロックとする。実際の実験のパターンは以下の通りである。

プロセッサ性能については，プロセッサ数は 8 とし，表 1 の 3 パターンとする。それぞれのパターンを”ProcSet 1”, ”ProcSet 2”, ”ProcSet 3”と呼ぶ。

初期負荷量については，負荷の総量は 600,000 とし，表 2 の 5 パターンとする。それぞれのパターンを”Load 1”, ”Load 2”, ”Load 3-a”, ”Load 3-b”, ”Load 4”と呼ぶ。

5. 動的負荷分散と漸近最適スケジューリングの分析

5.1 動的負荷分散

図 2, 図 3, 図 4 は動的負荷分散における，ブロックサイズと実行時間の関係を示している。ブロックサイズが大きくなれば，プロセッサ間での負荷の移動が行われなくなるので実行時間は一定となる。負荷情報通信のオーバーヘッドがないので，かなりブロックサイズが小さいところで実行時間が最小になっている。

5.2 漸近最適スケジューリング

図 5, 図 6, 図 7 は漸近最適スケジューリングにおけるラウンド数と実行時間の関係を示している。細線はそれぞれの場合における下限を示す。

ラウンド数が 1 の時には定常状態のラウンドがないため，特殊な結果になっている。

ラウンド数の変化による実行時間の変化の仕方には 2 通りある。1 つは，ラウンド数が 1 の時に最も実行時間が短く，ラウンド数が大きくなると実行時間が長くなる場合である。もう 1 つは，ラウンド数が小さいときに実行時間が長く，ラウンド数が大きくなると実行時間が短くなる場合である。

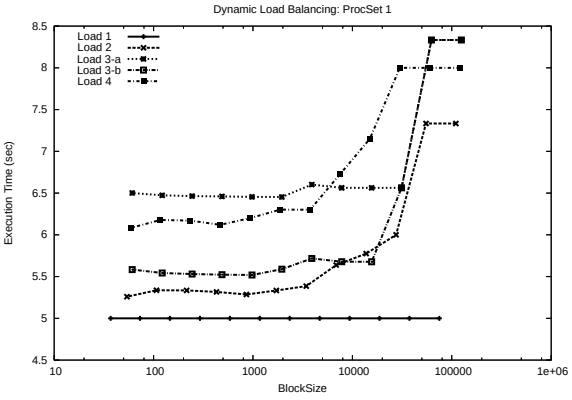

図 2 動的負荷分散：Processor Set 1
Fig. 2 Dynamic Load Balancing: Processor Set 1

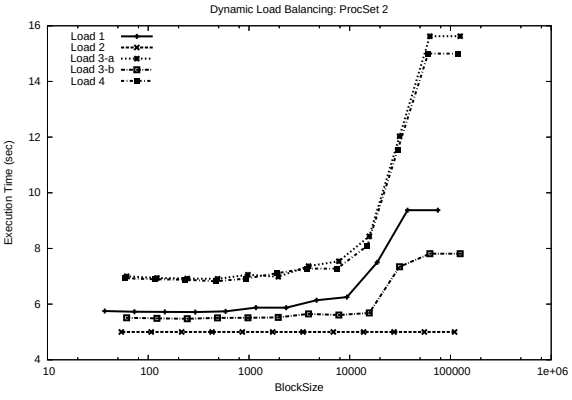

図 3 動的負荷分散：Processor Set 2
Fig. 3 Dynamic Load Balancing: Processor Set 2

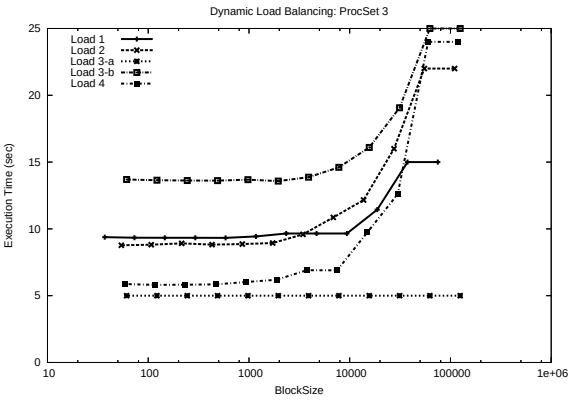

図 4 動的負荷分散：Processor Set 3
Fig. 4 Dynamic Load Balancing: Processor Set 3

表 2 初期負荷量の実験パターン
Table 2 Experiment patterns of the amount of initial load

	Load 1	Load 2	Load 3-a	Load 3-b	Load 4
Proc0	75,000	40,000	125,000	25,000	120,000
Proc1	75,000	50,000	125,000	25,000	120,000
Proc2	75,000	60,000	125,000	25,000	120,000
Proc3	75,000	70,000	125,000	25,000	120,000
Proc4	75,000	80,000	25,000	125,000	120,000
Proc5	75,000	90,000	25,000	125,000	0
Proc6	75,000	100,000	25,000	125,000	0
Proc7	75,000	110,000	25,000	125,000	0

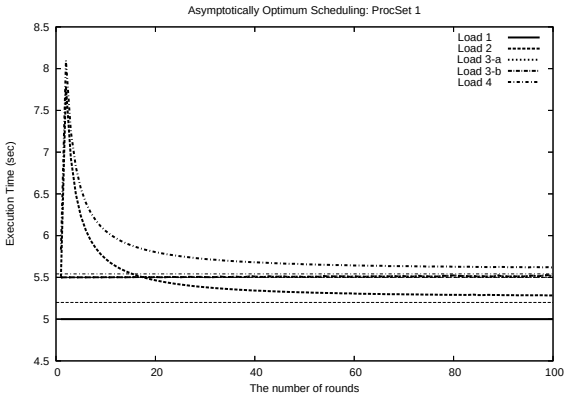

図 5 漸近最適スケジューリング：Processor Set 1
Fig. 5 Asymptotically Optimum Scheduling: Processor Set 1

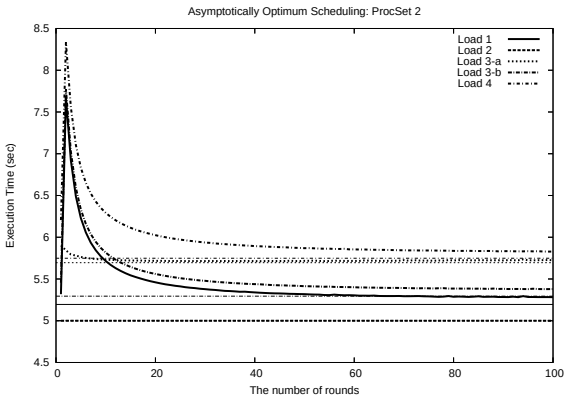

図 6 漸近最適スケジューリング：Processor Set 2
Fig. 6 Asymptotically Optimum Scheduling: Processor Set 2

ここで，前者を”AOS-Type 1”，後者を”AOS-Type 2”と呼ぶ事にする。

実行時間が最小になると，その後はラウンド数を大きくすると実行時間が長くなる。

5.3 負荷総量

図 8 は，負荷総量を変化させたときの，実行時間を最小とするラウンド数の変化を示している。

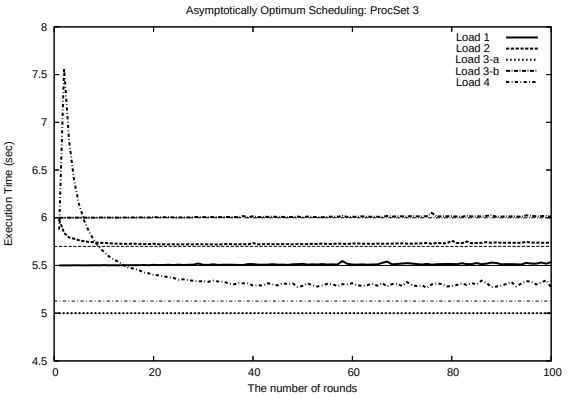

図 7 漸近最適スケジューリング：Processor Set 3
Fig. 7 Asymptotically Optimum Scheduling: Processor Set 3

プロセッサ性能のパターンは ProcSet 1 と ProcSet 2 の 2 通りを用いる。負荷総量は 1600000, 800000, . . . と変化させ，初期分割のパターンは，Load 3-a に従う。

1. 1,600,000 (320,000 × 4, 80,000 × 4)
2. 800,000 (160,000 × 4, 40,000 × 4)
3. 400,000 (80,000 × 4, 20,000 × 4)
4. 200,000 (40,000 × 4, 10,000 × 4)
5. 100,000 (20,000 × 4, 5,000 × 4)
6. 50,000 (10,000 × 4, 2,500 × 4)
7. 25,000 (5,000 × 4, 1,250 × 4)
8. 12,500 (2,500 × 4, 625 × 4)
9. 6,250 (1,250 × 4, 312 × 4)
10. 3,125 (625 × 4, 156 × 4)
11. 1,562 (312 × 4, 78 × 4)

ProcSet 1+Load 3-a のパターンは，5.2 の AOS-Type 1 の例であり，実行時間を最小にするラウンド数は，負荷の総量に関わらず 1 あるいは 2 となる。

表 3 各手法の実行時間の比較
Table 3 Comparison of the best execution time

		比例分割	動的負荷分散	漸近最適	下限
ProcSet 1	Load 1	5.0000 sec	5.0000 sec	5.0000 sec	5.0000 sec
	Load 2	5.8007	5.2581	5.2798	5.2000
	Load 3-a	5.5001	5.9789	5.5002	5.5000
	Load 4	7.2507	6.0750	5.6105	5.5422
ProcSet 2	Load 1	5.8007	5.6475	5.2748	5.1951
	Load 2	5.0000	5.0000	5.0000	5.0000
	Load 3-a	7.8007	6.8730	5.7129	5.6951
	Load 3-b	6.2007	5.4664	5.3742	5.2947
	Load 4	8.0007	6.8705	5.8211	5.7489
ProcSet 3	Load 1	5.5001	8.2362	5.5004	5.5000
	Load 2	7.8007	8.6559	5.7193	5.7000
	Load 3-a	5.0000	5.0000	5.0000	5.0000
	Load 3-b	6.0001	13.2966	6.0001	6.0000
	Load 4	5.9507	5.8080	5.2655	5.1263

表 4 効率の比較
Table 4 Comparison of the efficiency

		比例分割	動的負荷分散	漸近最適	下限
ProcSet 1	Load 1	1.000	1.000	1.000	1.000
	Load 2	1.116	1.011	1.015	1.000
	Load 3-a	1.000	1.087	1.000	1.000
	Load 4	1.308	1.096	1.012	1.000
ProcSet 2	Load 1	1.117	1.087	1.015	1.000
	Load 2	1.000	1.000	1.000	1.000
	Load 3-a	1.370	1.207	1.003	1.000
	Load 3-b	1.171	1.032	1.015	1.000
	Load 4	1.392	1.195	1.013	1.000
ProcSet 3	Load 1	1.000	1.497	1.000	1.000
	Load 2	1.369	1.519	1.003	1.000
	Load 3-a	1.000	1.000	1.000	1.000
	Load 3-b	1.000	2.216	1.000	1.000
	Load 4	1.161	1.133	1.027	1.000

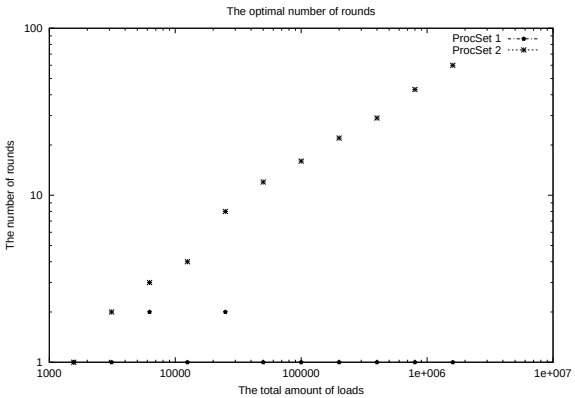

図 8 負荷総量と最適ラウンド数
Fig. 8 The optimum number of rounds and the total amount of loads

ProcSet 2+Load 3-a のパターンは，AOS-Type 2 の例であり，実行時間を最小にするラウンド数は，負荷の総量によって変わる。

3 節で述べた通り，実行時間を最小にするラウンド数は，$R = \sqrt{T/\alpha}$ となるので，負荷総量が 2 倍になるとラウンド数は $\sqrt{2}$ 倍になると考えられる。図 8 の ProcSet 2 の結果を見ると，ほぼ理論値通りになっていることが分かる。

6. 既存手法との比較

表 3 は比例分割・動的負荷分散・漸近最適スケジューリングによる実行時間を示している。実行時間は，それぞれのパターンにおける最小値である。どのような手法を用いても，実行時間を下限よりも小さくすることはできない。

表 4 はそれぞれのパターンにおける実行時間の効率を，効率 = (実行時間/下限) により求めた結果を示している。

ProcSet 2+Load 3-a, ProcSet 2+Load 4, ProcSet 3+Load 2, ProcSet 3+Load 4 のように比例分割・動的負荷分散の両方で効率が落ちるようなパターンにお

いても，漸近最適スケジューリングによって効率的に処理が行われることがわかる。これらは，処理性能の低いプロセッサの持つ負荷が多く，通信回数が多くなるために，通信の順序を効率よく行わないと実行時間の効率が悪くなると考えられる。

比例分割と比較すると，漸近最適スケジューリングはどのパターンについても実行時間が短い。動的負荷分散と比較すると，ProcSet 1+Load 2 で動的負荷分散の方が速いが，実行効率の差は 0.004 となっており，その差は小さいといえる。

比例分割の効率は 1.000〜1.392，動的負荷分散の効率は 1.000〜2.216 であるのに対し，漸近最適スケジューリングの効率は 1.000〜1.027 となっており，漸近最適スケジューリングではパターンによるばらつきが少ないことが分かる。このことから，漸近最適スケジューリングでは，様々なパターンにおいて効果的であるといえる。

7. ま と め

本研究では，シミュレータ上で multi-master divisible load モデルに対する漸近最適スケジューリングの実装を行い，既存手法と比較して，その特性を分析した。

比例分割や動的負荷分散によっても，漸近最適値に近い効率が得られる場合も多いが，問題のパターンに依存する部分が大きく，漸近最適スケジューリングでは，様々な問題に対して安定した結果が得られることが分かった。

今後の課題として，定常ラウンド以外のラウンドのスケジューリングがあげられる。最初と最後のラウンドではそれぞれ通信のみ，計算のみを行うために，各プロセッサの終了時刻が一致しない。そのため，実際の問題では，最初と最後のラウンドについてのスケジューリングを別に扱う必要があることが分かった。例えば，最初のラウンドにおいてプロセッサが初期状態で持っている負荷を処理するといったことが考えられる。

また，漸近最適スケジューリングの拡張として，ラウンドの大きさを固定しない方法が考えられる。[5] master が送る負荷の量に対して，slave の受け取り可能な量が十分に多ければ，その余地に応じて通信を多く行い，次のラウンドのサイズを大きくする。このように，ラウンドのサイズを次第に大きくすると，その結果，ラウンドの数が減少する。ラウンドの数が減少することで，通信の立ち上がり遅延の影響を減らす事ができると考えられる。

さらに，今後，複数のクラスタを接続したマルチクラスタ環境への拡張[2] と，実機上での実験による漸近最適スケジューリングの評価を行う必要がある。

謝辞 本研究は，文部科学省の 21 世紀 COE プログラムと科学研究費補助金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 2003-ARC-157/2003-HPC-97, Mar. 2004, 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, Jul. 2004, pp. 103–108
3) T.G.Robertazzi, *Ten Reasons to Use Divisible Load Theory*, IEEE Computer, Vol.36, No.5, May 2003, pp. 63–68.
4) O.Beaumont, A.Legrand and Y.Robert, *Scheduling divisible workloads on heterogeneous platforms*, Parallel Computing, Vol.29, No.9, Sep. 2003, pp. 1121–1152.
5) Y.Yang and H.Casanova, *RUMR: Robust Scheduling forDivisible Workloads*, In Proceedings of the 12th IEEE Symposium on High Performance Distributed Computing(HPDC-12), Jun. 2003, pp. 114–125.