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

2005－HPC－103（7）
2005／8／3

Multi-master divisible load の漸近最適スケジューリングの実機への実装

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実機上で実装する。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。行列ベクトル積をタスクとして実機上で実行したときの様子を，シミュレーションによるスケジュールと比較する。スケジューリングの際のパラメタを調整することで，所要時間の予測値と実測値がほぼ等しくなり，実際の所要時間を短縮することができた。

Asymptotically optimum scheduling of multi-master divisible load problems evaluated on actual machines

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 in actual machines. The scheduling algorithm assumes no overlap of communication and computations, and no collection of the results. The result of matrix-vector multiplication on a actual machines is compared with the scheduling by the simulation. The scheduling parameters are adjusted and the predicted and observed execution time became nearly equal. Furthermore, the observed execution time is shortened.

1. はじめに

MMDL (Multi-Master Divisible Load) は，2) 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)[4] 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，2) 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。1) 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した。

本研究では，漸近最適スケジューリングを実機上で実装し，行列ベクトル積をタスクとして性能を分析する。

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

2) で提案された MMDL のモデルは以下の通りである。

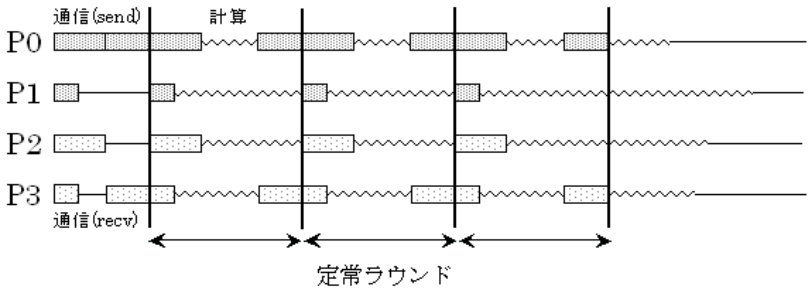

図 1 定常ラウンドスケジューリング
Fig. 1 Scheduling by repeating stable rounds

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[4] 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。

2.3 定常ラウンド

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。[5]

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。

3. スケジューリング最適化

実機上で実行するための最適なスケジューリングを求めるには，実際の環境に合わせてスケジューラに与えるパラメタを変更する必要がある。

パラメタには，CPU・ネットワークの性能等のハードウェア的なパラメタと，タスクあたりの計算量・通信量等のソフトウェア的なパラメタがある。これらのパラメタは，実際にタスクを実行してみなければ決めることができない。

そこで，図 2 のようにまず適当なパラメタをスケジューラに与え，そのスケジューリングに従って実行した結果とスケジューリングを比較し，パラメタを変更する。

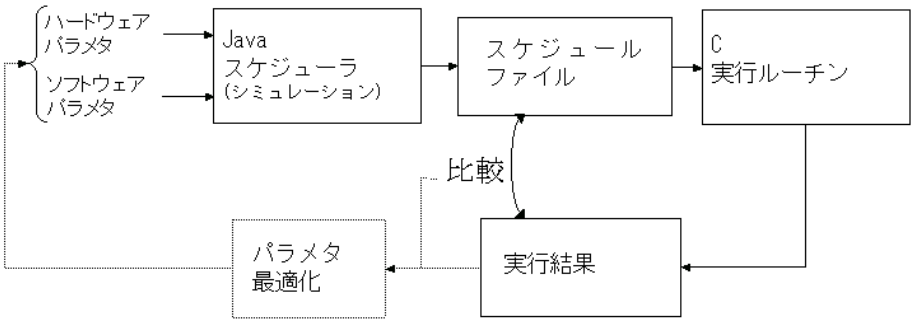

図 2 最適化の流れ
Fig. 2 Optimization flowchart

スケジューラに与える入力は,

- プロセッサの数と性能
- それぞれのプロセッサの負荷量
- ネットワーク性能 (通信遅延，スループット)
- 単位タスクあたりの計算量と通信量
- ラウンド数

である。これらの入力に対して，スケジューラは各プロセッサが行う計算・通信の量と順序を返す。計算は処理する負荷の量によって指定し，通信は通信相手と負荷の量によって指定する。

C の実行ルーチンは，スケジューラの出力したスケジュールファイルを読み込み，スケジュールに従って送信・計算・受信のサブルーチンを呼び出して処理を行う。各サブルーチンの開始・終了時刻を記録し，実行結果とする。

この実行結果とスケジューリングを比較し，パラメタの最適化を行う。今回の実験では，ハードウェアパラメタを固定し，タスクあたりの計算量・通信量のパラメタのみを変更する。計算量パラメタは，単位タスクの計算にかかった時間と CPU の性能から,

$$\text{計算所要時間 (s)} \times \text{CPU 性能 (MHz)} \quad (1)$$

により求めることができる。

通信量パラメタは,

$$\text{通信所要時間 (s)} \times \text{通信性能 (MB/s)} \quad (2)$$

により求める。

変更したパラメタをスケジューラに入力して再スケジューリングを行うことで，より良いスケジューリングを求めることができる。

4. 実　　験

4.1 タ ス ク

処理するタスクとして行列ベクトル積を用いて実験を行う。図 3 のように，各プロセッサが共通の行列と大量のベクトルを持っているとする。このベクトルをスケジューリングに従って送受信し，行列との積を求める。従って，1 つのベクトルの処理が単位タスクとなり，単位タスクの通信はベクトルの送受信，計算は行列との積となる。計算の結果は収集しなくてよいものとする。

行列のサイズを $m \times n$ とすると，計算は

$$\underset{(m \times n)}{\text{行列}} \quad * \quad \underset{(n \times 1)}{\text{データベクトル}} = \quad \underset{(m \times 1)}{\text{結果ベクトル}}$$

となる。行列は初期状態で全てのプロセッサが持っていて，通信する必要はなく，通信はデータ用のベクトルと，結果用のベクトルとする。

よって，単位タスクあたりの計算量は $O(mn)$，通信量は $O(m+n)$ となる。

このように，行列ベクトル積では，行列のサイズを変更することにより，単位タスクあたりの計算量と通信量を自由に設定できる。MMDL モデルの評価において，タスクの特性を自由に設定できる必要があるため，今回の実験では行列ベクトル積を用いた。

4.2 実験パターン

プロセッサ性能の組み合わせ，ベクトル本数の分配パターンを変えて，実行の過程を計測する。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 2 パターンとする。それぞれのパターンを”ProcSet 1”, ”ProcSet 2”と呼ぶ。プロセッサ同士は Gigabit Ethernet で接続されている。

初期状態でそれぞれのプロセッサが持っているベクトルの本数については，ベクトルの総本数は 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。

行列のサイズは 20×400 とする。

表 2 ベクトル本数の分配パターン
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

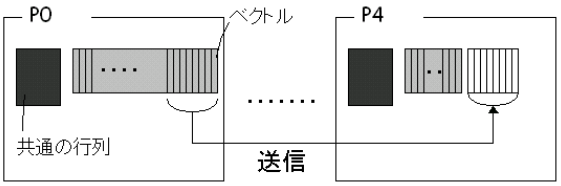

図 3 行列ベクトル積の実行
Fig. 3 Execution of matrix-vector multiplication

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
Proc0	2400 MHz	2400 MHz
Proc1	2400	2400
Proc2	2400	2400
Proc3	2400	2400
Proc4	2400	2400
Proc5	2400	2400
Proc6	2400	2000
Proc7	2400	2000

最初の実行では，計算量パラメタを 0.1，通信量パラメタを 0.003 としてスケジューリングを行った。これは，シミュレータ上で，単位タスクあたりの計算量 100K クロック，通信量 3K バイトに対応する。また，通信遅延は 100μ 秒，通信速度のスループットは 800Mbps と仮定した。

4.3 パラメタの変更

実行結果から，通信量と計算量のパラメタを変更して再スケジューリングを行う。

実機上での計算・通信の所要時間から，式 1，式 2 を用いてパラメタを求め，表 3 のようにそれぞれのパラメタを変更した。行列サイズが同じならば，パラメタはプロセッサ性能とベクトル分配のパターンに依存しない。

ここで，行列サイズ 20×400 より，単位タスクあたりに行う通信は長さ 400 のベクトル (データ用) と，長さ 20 のベクトル (結果用) となるので，約 3.36Kbyte と考えられる。所要時間から求めたパラメタは 0.0034(3.4Kbyte に相当) であり，これにほぼ一致する。

表 3 パラメタの変更 (行列サイズ 20×400)
Table 3 Change of parameter

	計算量	通信量
変更前	0.100	0.0030
変更後	0.294	0.0034

5. シミュレーションとの比較

5.1 計算量と通信量の分析

表 4 に，パラメタの変更による実行時間の変化の様子を示す。パラメタの変更によって，実機での全体の実行時間がシミュレーションとほぼ等しくなっていることが分かる。

実行過程も，図 6 と図 7 を比較するとほぼ同じようになっていることが分かる。プロセッサとベクトル本数の分配のパターンの組み合わせを変えてもほぼ同様の結果が得られる。

さらに，表 4 で，パラメタ変更前後の実機での実行時間を比較すると，ほとんどの場合でパラメタ変更によって実行時間が短縮されており，実機の環境に合わせて再スケジューリングを行うことで，より良いスケジューリングを得られていることが分かる。

5.2 ラウンド数

表 5 は，ProcSet 2，Load 2 で，ラウンド数を 1 から 20 まで変化させてスケジューリングを行ったときの実行時間である。ラウンド数を変化させると，シミュレーションでの値と同様に実際の実行時間が変化することが分かる。

これより，図 8 のように，シミュレーションによって最適なラウンド数を求め，それを用いて実機上でも実行時間を短くすることができると考えられる。

6. ま と め

本研究では，実機上で 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

表 4 実行時間の予測値と実測値の比較 (ラウンド数:5)
Table 4 Comparison of the predicted and observed execution time(round: 5)

		シミュレーション		実機	
		変更前	変更後	変更前	変更後
ProcSet 1	Load 1	3.125000 s	9.187500 s	9.217236 s	9.208515 s
	Load 2	4.589715	11.845532	12.633339	11.835162
	Load 3-a	4.626095	10.888976	10.884279	10.941214
	Load 4	5.238346	12.886128	13.015991	12.822947
ProcSet 2	Load 1	4.159893	11.724372	11.839234	11.712632
	Load 2	4.883831	12.431613	13.245593	12.474350
	Load 3-a	4.712693	11.253531	11.591924	11.417429
	Load 3-b	5.045411	11.501171	12.467642	11.568004
	Load 4	5.287006	13.214013	13.520686	13.477915

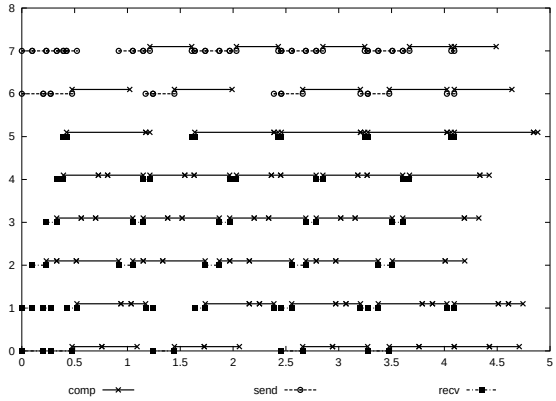

図 4 シミュレーション パラメタ変更前：ProcSet 2, Load 2
Fig. 4 Simulation before: ProcSet 2, Load 2

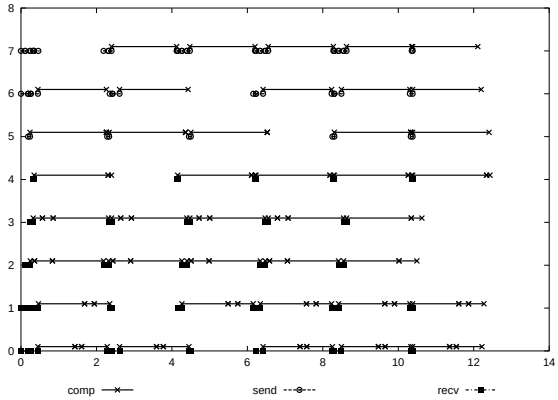

図 6 シミュレーション パラメタ変更後：ProcSet 2, Load 2
Fig. 6 Simulation after: ProcSet 2, Load 2

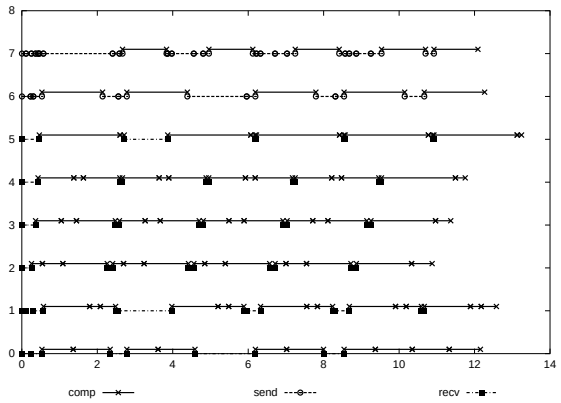

図 5 実機 パラメタ変更前：ProcSet 2, Load 2
Fig. 5 Actual machine before: ProcSet 2, Load 2

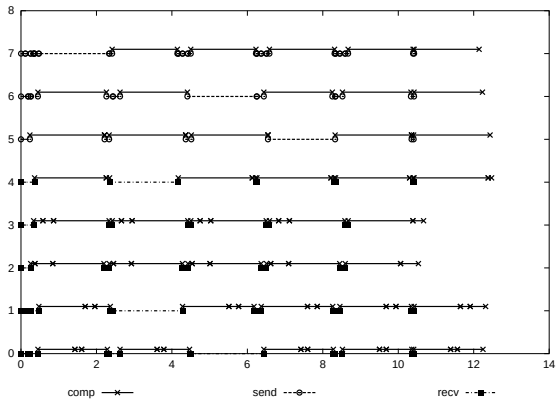

図 7 実機 パラメタ変更後：ProcSet 2, Load 2
Fig. 7 Actual after: ProcSet 2, Load 2

い，行列ベクトル積をタスクとして，シミュレーションとの比較を行った。

パラメタの調整を行うことで，シミュレーションと実機上での実行過程と実行時間はほぼ等しくなることが分かった。また，パラメタ変更によるスケジューリングの改善によって，実機上での実行時間が短縮された。これより，シミュレータ上で最適なスケジューリングを求めることで，実機上でも良い結果を得られると考えられる。

今後の課題として，

- スケジューラ内での最適なラウンド数決定
- パラメタ最適化の自動化
- スケジューラの改善

などがあげられる。

表 5 ラウンド数と実行時間 (ProcSet 2, Load 2)
Table 5 The number of rounds and the execution time

	シミュレーション	実機
1	11.996034 s	11.895846 s
2	15.506518	15.542628
3	13.798204	13.837349
4	12.944644	12.993227
5	12.431613	12.474350
6	12.090941	12.117615
7	11.846690	11.878990
8	11.665072	11.691494
9	11.522636	11.542078
10	11.408744	11.440548
11	11.316546	11.331306
12	11.238737	11.271751
13	11.173820	11.198440
14	11.118194	11.113453
15	11.070093	11.078544
16	11.029194	11.019053
17	10.989639	10.986812
18	10.956554	10.957962
19	10.928810	10.924403
20	10.900142	10.918454

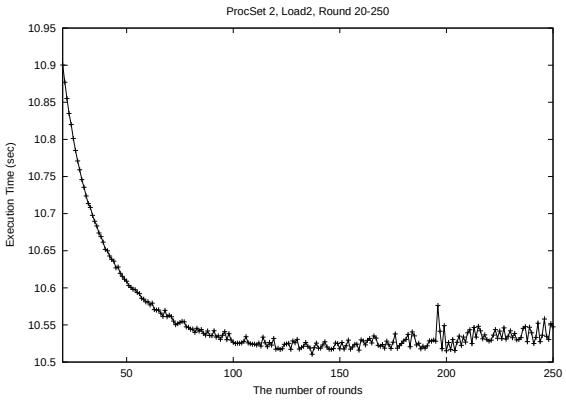

図 8 シミュレーション: ラウンド数と実行時間
Fig. 8 Simulation: The number of rounds and the execution time

最適ラウンド数はプロセッサと初期分配の組み合わせ，負荷の総量によって変化する[1] ので，これを考慮してシミュレーションを行って決定する。

パラメタ最適化の部分は，ハードウェアパラメタの最適化も含めて，実行結果の所要時間等から自動的に最適なパラメタを求めて再スケジューリングを行うようにする。

また，スケジューラは定常ラウンド以外のラウンドのスケジューリングを行い，全体のスケジューリングを最適化する必要がある。そのために，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い，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，あるいは，漸近最適スケジューリングを拡張して，ラウンドの大きさを固定しない方法[6] を用いることを考える。

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[3] と，行列ベクトル積以外のタスクでの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) 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, Jun. 2005, pp. 51–56
2) 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
3) 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
4) T.G.Robertazzi, *Ten Reasons to Use Divisible Load Theory*, IEEE Computer, Vol.36, No.5, May 2003, pp. 63–68.
5) O.Beaumont, A.Legrand and Y.Robert, *Scheduling divisible workloads on heterogeneous platforms*, Parallel Computing, Vol.29, No.9, Sep. 2003, pp. 1121–1152.
6) 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.