システムソフトウェアと
オペレーティング・システム 70−13
（１９９５．８．２３）

NUMA マルチプロセッサにおける メモリ管理を考慮した２レベルスケジューリング

大石幸雄† 藤木亮介‡ 最所圭三† 福田晃†

†: 奈良先端科学技術大学院大学
‡: 九州大学
(現 西日本鉄道株式会社)
Email: {yukio-o,sai,fukuda}@is.aist-nara.ac.jp

NUMA (Non-Uniform Memory Access) マルチプロセッサを対象とする，メモリ管理を考慮した 2 レベルスケジューリングについて述べる．本稿ではまず，メモリ管理を考慮した 2 レベルスケジューラの構造について述べる．メモリ管理とスケジューリングの協調動作の有効性，および実現方策における基本的な選択肢を評価するために，(1) プロセス空間のローディング方式，(2) ページアクセス方式，(3) フリープロセッサの割り当て方式の 3 項目についてシミュレーションによる比較・検討を行なった．その結果，(1) に対しては，スケジューラからのフリープロセッサ (どのグループにも属していないプロセッサ) 数に関する情報提供が有効であり，また，(2) に関しては，システムの負荷を考慮することが必要であるということがわかった．そして，(3) については，プロセス空間の割り当てに関する情報の提供が効果的であるということがわかった．

Two-level Scheduling in Collaboration with Memory Management for NUMA Multiprocessors

Yukio Oishi† Ryosuke Fujiki‡ Keizo Saisho† Akira Fukuda†

†: Nara Institute of Science and Technology
‡: Kyushu University
(Currently: Nishi Nippon Railway Co.Ltd)
Email: {yukio-o,sai,fukuda}@is.aist-nara.ac.jp

Two-level scheduling strategy in collaboration with memory management for NUMA (Non-Uniform Memory Access) multiprocessors is discussed. In this paper, we describe about the structure of two-level scheduler . Through simulation experiments, we evaluate the performance of the alternatives from the viewpoints of the following: 1) process loading , 2) page access, and 3) free-processor allocation. For process loading, information about the number of free processors is effective. For page access, system load should be considered. For free-processor allocation, information about allocation of address space of process is effective.

1 まえがき

NUMA(Non-Uniform Memory Access) マルチプロセッサにおけるスケジューリングアルゴリズムの提案，及びシミュレーションによる評価を行なう．

我々は，空間分割方式の 1 つである 2 レベルスケジューリングについて，その有効性および本スケジューリングにおける基本的な選択肢を整理し，それらをシミュレーションにより評価してきた [1]．文献 [1] では，スケジューリングをメモリ管理とは独立としてきた．

しかし，スケジューリング方策は，プロセスのメモリへの割り当て方策と密接な関係があり，両者の間には何らかの協調が必要である．特に，メモリの不均一性を有する NUMA マルチプロセッサにおいては，メモリを含めてスケジューリングを考えることは不可欠であると考えられる．

本稿では，マルチプログラミング環境下の NUMA マルチプロセッサ上で，メモリ管理を考慮した 2 レベルスケジューリングに関する研究の第一歩として，その有効性および各種の基本的な選択肢をシミュレーションにより評価する．

2 構造

スケジューラとメモリマネジャの関係を図 1 に示す．なお，各インタフェースの説明は省略する．

(1) スケジューラ

スケジューラはグローバル・スケジューラとプライベート・スケジューラから構成される．グローバル・スケジューラはさらに，プロセッサの動的グルーピングを行なうグループ・サーバと，フリープロセッサを置くプロセッサ・プールの管理を行うプール・サーバから構成される．各プロセスはグループ・サーバによって 1 つのプロセッサ・グループに割り当てられる．グローバル・スケジューラはシステム内に一つ存在し，カーネル空間で動作する．プライベート・スケジューラは，グループ内プロセッサを用いて対応するプロセス内のスレッドをスケジューリングし，対応するプロセス空間に存在する．

(2) メモリマネジャ

スケジューラと同様に，グローバル・メモリマネジャとプライベート・メモリマネジャから構成される．グローバル・メモリマネジャは，複数のプロセス空間の物理メモリへのグローバルな割り当て／管理を行う．プライベート・メモリマネジャは，プロセス毎に存在し，対応するプロセス空間と物理メモリとの割り当て／管理を行う．

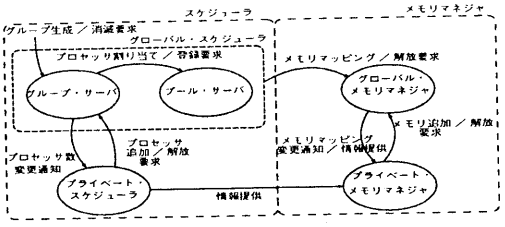

図 1: スケジューラとメモリマネジャ

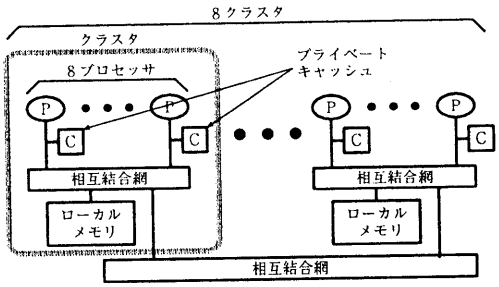

図 2: NUMA マルチプロセッサモデル

3 対象シミュレーションモデル

3.1 マルチプロセッサモデル

複数のクラスタからなる NUMA マルチプロセッサシステムを対象とする (図 2)．各クラスタは，8 個のプロセッサとプロセッサ対応のキャッシュ，1 つの物理メモリから構成される．システムは全 8 クラスタ，計 64 台のプロセッサから構成される．

3.2 プロセスモデル

本稿では，プロセスモデルとして FJ(Fork-Join) 型 (図 3)，MVA(Mean Value Analysis) 型 (図 4) の 2 つを取り扱う．

FJ 型のプロセスは，複数スレッドの同時生成／消滅と逐次実行を繰り返すパターンとする．MVA 型は，ある種の待ち行列網を MVA(Mean Value Analysis) を用いて解析する際に表れるモデルである [2]．

全てのプロセスのアドレス空間は 32 の論理ページから構成される．各ページは読み出し専用 (Read-Only, R-O) または，読み書き (Read-Write, R/W) の属性を持つ．スレッドの実行はアドレス空間内の論理ページの時系列で与えられる．

各論理ページに関する実行時間は，後述するメモリアクセスオーバヘッドがない場合，ページの種類にかかわらず，一定 (T) とする．これは，後述する表 1 において，アクセスするページがローカルメモリに

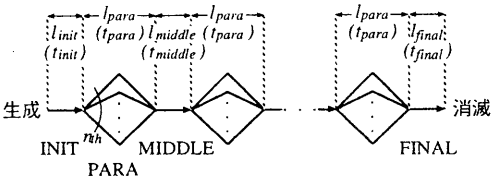

図 3: FJ 型プロセスモデル

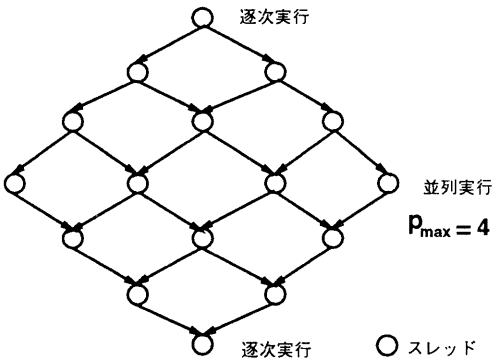

図 4: MVA 型プロセスモデル

あり，かつページの一部がキャッシングされている場合に相当する．FJ 型において，INIT, PARA, MIDDLE, FINAL の各フェーズにおけるスレッドの実行時間，$t_{init}, t_{para}, t_{middle}, t_{final}$は，論理ページの時系列長，$l_{init}, l_{para}, l_{middle}, l_{final}$を用いて，$t_{phase\_name}$ = (論理ページ当たりの実行時間)×l_{phase_name}，となる．但し，論理ページの当たりの実行時間は，後述する表 1 となる．また，MVA 型では，各スレッドにおける論理ページの時系列長は，l_{para}としている．

今回のシミュレーションでは，パラメータとして T=3, $l_{init} = l_{middle} = 4, l_{final} = 10$ を固定とし，$l_{para} = U(11,33)$，FJ 型のプロセスで生成されるスレッド数を $n_{th} = U(2,28), U(20,28)$，MVA 型のプロセスの最大並列度を $p_{max} = U(14,16)$ とし，プロセス毎に変化させた．ここで，$U(a,b)$ は a から b までの一様分布関数である．

3.3 オーバヘッド

(1) メモリアクセスオーバヘッド

異なる種類のメモリアクセス時間を持つ NUMA マルチプロセッサにとって，割り当てられるメモリの位置やキャッシングの方法によって，メモリアクセスのオーバヘッドが変化する．本稿ではメモリアクセスのオーバヘッドをモデル化してシミュレーションを行なっている．

表 1: メモリアクセス・オーバヘッド

	アクセスするページがローカルメモリにある	アクセスするページがリモートメモリにある
ページの一部がキャッシングされている	T	4T
キャッシングされていない	2T	6T

Tは必要処理時間 (=3)

ページコピー及び移動にかかるオーバヘッド：70

スレッドの実行は，読み出し専用／読み書きの属性を持つ論理ページの時系列で与えられる．時系列中の各論理ページに関する実行時間は，アクセスする論理ページのキャッシュ内へのキャッシングの有無，主記憶内での位置，および，ページコピーの有無により変化する．キャッシングの有無は，プロセッサが直前にアクセスしたページと同一の論理ページをアクセスしているかどうかによって判断する．主記憶内における位置は，プロセス空間のロードやページコピーによってローカルメモリに論理ページがあるかどうかによって判断する．ページコピーは，プロセッサがアクセスする読み出し専用ページがリモートメモリにある場合に行なわれる．

以上のような分類をもとに各論理ページに関する実行時間を表 1 に示す．メモリアクセスおよびページコピー・移動のオーバーヘッドはマシンの実測値 [3],[4] を考慮したものである．

(2) スケジューラとメモリマネジャ

シミュレーションではスケジューラ／メモリマネジャの実行によるオーバヘッドをモデル化している．グローバル・スケジューラ内のグループ／プール・サーバ，プライベート・スケジューラ，および，グローバル／プライベート・メモリマネジャはそれぞれいくつかの関数を提供している．シミュレーションではグループ・サーバが提供する全ての関数の実行時間を同一 ($O_{global\_sche} = 2$) とする．また，プール・サーバ／グローバル・メモリマネジャはグループ・サーバによって呼び出されるため，それぞれが提供する関数の実行時間は O_{global_sche} に含まれている．同様にプライベート・スケジューラの実行時間も全て $O_{private\_sche}(= 1)$ としている．

また，各サーバ内での実行は排他制御される．すなわち，あるサーバが実行中であれば，そのサーバに発行される他の全ての要求はサーバの実行終了まで待たされる．このようにサーバの実行は逐次化される．

4 スケジューリング方策

ここでは，プロセスの生成／消滅，スレッドの動的生成／消滅などにおける，スケジューリング方策について述べる．

4.1 フリープロセッサの分類

(1) クラスタによる分類

フリープロセッサは，プロセッサを要求するプロセスとの位置関係により，次の3つに大別される．

- **ホームプロセッサ：** プロセス空間がローディングされているクラスタのプロセッサ．プロセスへのアクセスは，最悪でもローカルアクセスとなる．
- **ローカルプロセッサ：** 仮想ページのコピー，移動などによって，プロセス空間の一部が割り当てられているクラスタのプロセッサ．プロセスへのアクセスがローカルアクセスになる可能性がある．
- **リモートプロセッサ：** その他のクラスタのプロセッサ．リモートアクセスのオーバヘッド，または，コピー，移動のオーバヘッドが生じる．

(2) プロセス属性による分類

フリープロセッサがフリーとなった契機とプロセッサを要求するプロセスとの関係より，次の3つに大別される．

- **同一属性プロセッサ：** フリーとなる直前に割り当てられていたプロセスと，今プロセッサを要求しているプロセスが同一であるプロセッサ．キャッシュを活用できる可能性がある．
- **空属性プロセッサ：** プロセスの終了によりフリーとなったプロセッサ．キャッシュの活用は見込めない．
- **異属性プロセッサ：** フリーとなる直前に割り当てられていたプロセスと，プロセッサを要求しているプロセスが異なるプロセッサ．本プロセッサは，将来，直前に割り当てられていたプロセスからの要求があり，そのプロセスに対するキャッシュの活用が見込まれる．

4.2 グループ・サーバ

(1) グループ生成

プロセスが生成されると，グループ・サーバはグループ生成要求を受けてプロセッサグループを生成する．また，プロセス空間を物理メモリにローディングするために，グローバル・メモリマネジャに対してメモリマッピング要求を発行する．その後，プライベート・スケジューラが起動される．

(2) グループ消滅

グループ・サーバは，グループ消滅要求を受けとると，該当するプロセッサグループ内の全てのプロセッサを解放し，プロセッサグループを削除する．生じたフリープロセッサは，プロセッサを要求しているプロセスがあればそれらのプロセスに割り当てられるが，割り当てるプロセスの優先順位に関して，以下の2つの選択肢がある [1]．

1. 実行プロセス優先方式
2. 待ちプロセス優先方式

実行プロセスが複数存在する場合の選択順序として，さらに降順方式，昇順方式の2つが存在する [5]．その後，グローバル・メモリマネジャに対してメモリマッピングの解除を要求する．

(3) プロセッサ追加

プライベート・スケジューラからのプロセッサ追加要求により，プール・サーバからフリープロセッサを該当グループに追加する．

(4) プロセッサ解放

プライベート・スケジューラからのプロセッサ解放要求により該当するグループからプロセッサを解放する．解放されたプロセッサの割り当てに関しては，グループ消滅の場合と同様である．

4.3 プール・サーバ

プール・サーバは，フリープロセッサのプロセス属性として，プール内のフリープロセッサがプールに登録される直前に属していたグループのID(プロセス属性) を記憶している．この情報は，プロセッサの解放時にグループ・サーバから与えられ，プロセッサとキャッシュの親和性を利用するために用いられる．

(1) プロセッサ割り当て

プロセッサ割り当て要求を受けると，プール・サーバはグループ・サーバから与えられた情報を用いて，プロセッサの割り当てを行なう．選択の順序としては，4.1節の分類を基に，以下のような選択肢が考えられる．

1. メモリ管理を考慮しない割り当て方式 [5]：
   (a) 同一属性プロセッサ
   (b) ホームプロセッサ
   (c) 空属性プロセッサ
   (d) その他のプロセッサ
2. メモリ管理を考慮した方式：
   (a) ホームプロセッサ
   (b) ローカルプロセッサ
   (c) フリープロセッサの多いクラスタのプロセッサ

(2) プロセッサ登録

該当プロセッサをプロセッサプールに登録する．この時，該当プロセッサが解放されたグループにより，フリープロセッサのプロセス属性を記憶する．

4.4 プライベート・スケジューラ

本稿で考慮している方策では，プロセッサの横取りは行なわれない．よって，プロセッサ数変更通知イン

タフェースは使用しない．各プライベート・スケジューラは，プロセス内でFCFSスケジューリングを行なう．また，スレッドの生成，消滅に対して以下の動作を行なう．

(1) スレッド生成

スレッドの動的生成時にグループのプロセッサが不足した場合，プライベート・スケジューラは，生成されたスレッドをレディキューにつなぎ，不足数分のプロセッサをグループ・サーバに要求する．

(2) スレッド消滅

スレッドの実行終了により生じたアイドルプロセッサへの対処として，以下の2つが存在する [1]．

1. アイドルプロセッサ保持方式
2. アイドルプロセッサ解放方式

4.5 グローバル・メモリマネジャ[6]

(1) メモリマッピング

プロセスが生成されると，グローバル・スケジューラからメモリマッピング要求が発行される．これを受けてグローバル・メモリマネジャは，ある1つのクラスタの物理メモリに対してプロセス空間のローディングを行なう．このとき，ローディングするクラスタの選定方策に関して，いくつかの選択肢が考えられる．

1. スケジューラからの情報を必要としないもの
   (a) 巡回方式
   (b) ランダム選択方式
   (c) プロセス数最小方式
2. スケジューラからの情報を利用するもの
   (d) フリープロセッサ数最大方式
   (e) フリープロセッサが存在するクラスタ内での，プロセス数最小方式

この後，生成されたプロセスに対応するプライベート・メモリマネジャが起動される．

(2) メモリマッピング解除

プロセスが消滅すると，グローバル・スケジューラからメモリマッピング解除要求が発行される．これを受けてグローバル・メモリマネジャは，プロセス空間がマッピングされている全ての物理メモリを解放する．

4.6 プライベート・メモリマネジャ

(1) メモリ追加

前述のように，プロセス空間全体は，ある1つのクラスタの物理メモリにローディングされる．スレッドの実行中，プロセッサによってアクセスされる論理ページがリモートメモリに存在する場合，次の様な選択肢がある (ページアクセス方式と呼ぶ)．

(1) コピーや移動を行なわず，リモートアクセスのままとする．

(2) 読み出し専用ページであればローカルメモリへのページコピーによるキャッシングを行ない，読み書きページであればキャッシングは行なわずリモートアクセスのままとする．

(3) n回までは，読み出し専用ページはコピー，読み書きページは移動を行ない，n回を越えると全てリモートアクセスとする．

(2) メモリ解放

スレッドの実行中，コピーページのコヒーレンス制御や論理ページの移動などによって物理メモリが不要となる場合がある．この場合プライベート・メモリマネジャは，不要となった物理メモリを解放するために，メモリ解放要求をグローバル・メモリマネジャに対して発行する．

5 評価する方式

ここでは，4節で述べた方策の選択肢の中で，今回比較した選択肢についての整理を行なう．

また，4章で述べた比較方式の内，これまでの研究から評価が得られた項目については，良い結果が得られた選択肢に固定する [5]．すなわち，アイドルプロセッサ処理方式としてはアイドルプロセッサ保持方式，フリープロセッサ処理方式としては実行プロセス優先方式，実行プロセス選択方式としては降順方式をそれぞれ選択する．

5.1 プロセス空間のローディング

プロセスの生成時に，グローバル・メモリマネジャによってプロセス空間の物理メモリへのローディングが行なわれる．今回のシミュレーションでは以下の4方式について比較する．

1. スケジューラからの情報を必要としないもの
   (a) 巡回方式
   (b) プロセス数最小方式
2. スケジューラからの情報を利用するもの
   (c) フリープロセッサ数最大方式
   (d) フリープロセッサが存在する中での，プロセス数最小方式

5.2 ページアクセス方式

アクセスする論理ページがローカルメモリにない場合の動作について，4.6節で述べた，以下の3方式について比較する

(1) 全てリモートアクセス

(2) 読み出し専用ページはコピー，読み書きページはリモートアクセス．
(3) n 回までは，読み出し専用ページはコピー，読み書きページは移動．それ以降は全てリモートアクセス．

5.3 フリープロセッサの割り当て

プロセス，スレッドの生成時にプロセッサの割り当てが要求された場合，フリープロセッサのグループへの割り当てが行なわれる．本論文では，4.3節で述べた，以下の 2 つの方式を比較する．

(A) メモリ管理を考慮しない従来の割り当て方式
(B) メモリ管理を考慮した方式

6 結果および考察

本章では，5節で述べた 2 レベルスケジューリングにおける種々の選択肢を，シミュレーション結果を基に検討する．

シュミレーションにおけるシステムの負荷はプロセスの平均生成頻度で与えられるが，それがどのどの程度の負荷レベルを示すものかわかりにくい．そこで，システムの負荷レベルの目安となる値 Load level[%] (L) を用いている．この値は，各論理ページ当たりの実行時間が，キャッシュミスとリモートメモリアクセスのオーバヘッド (表 1 の 6T) を受けるとした場合の，プロセッサの平均利用率として求めており，以下のような関係になっている．

$$t_{process} \times \overline{F} = \frac{P \times L}{100}$$

P：システムのプロセッサ台数
$t_{process}$：プロセスの平均逐次実行時間 [単位時間]
$\overline{F}$：プロセスの平均生成頻度 [プロセス／単位時間]

$t_{process}$は，FJ 型の場合，平均 fork 数を 24，繰り返し回数を 5 とすると，次のようになる．

$$t_{process} = t_{init} + (t_{para} \times 24) \times 5 + t_{middle} \times (5-1) + t_{final}$$

6.1 プロセス空間のローディング

まず，プロセス生成時におけるプロセス空間のローディング方式に関する比較を行なう．FJ 型，MVA 型の場合を各々図 5-6，図 7-8 に示す．図中の **a**〜**d** は各々5.1節の (a)〜(d) に対応する方式である．

ページアクセス方式は 5.2 節 (2)，フリープロセッサの割り当ては 5.3 節 (B) を用いている．

図 5，7からわかるように，ローディング方式 (c)(d) が (a)(b) に比べ良い結果を示し，プロセスのローディング時にスケジューラからの情報を利用するのが有効

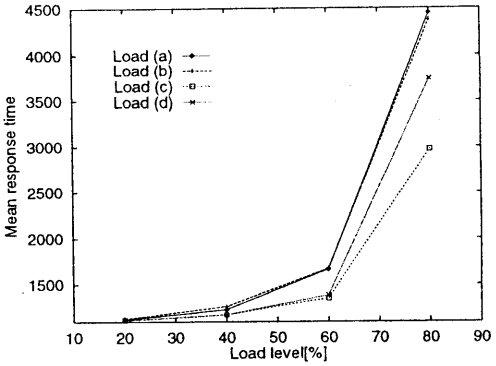

図 5: 平均応答時間 (FJ2 回，fork24)

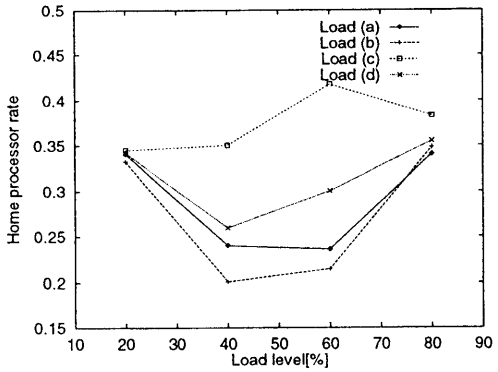

図 6: ホームプロセッサの割合 (FJ2 回，fork24)

であると言える．また，その中でも特に (c) の方式が最も応答時間は短い．

これは図 6，8に示されるように，フリープロセッサ数が多いクラスタにプロセスをローディングすることによりホームプロセッサの割合が増え，結果としてローカルメモリアクセスが増加し，リモートメモリアクセスのオーバヘッドを削減できるからである．

6.2 ページアクセス方式

プロセスの実行中にアクセスしようとするページがローカルメモリにない場合のメモリマネジャの動作について比較を行ない，その結果を図 9，10に示す．

プロセスのローディング方式は 5.1 節 (c),(d)，フリープロセッサの割り当ては 5.3 節 (B) を用いている．

結果を見てみると，平均 fork 数 15(図 9) では，方式 (2) が全体的に良いが，平均 fork 数 24(図 10) では，システム負荷が 80% となると，プロセスのローディング方式を (c)，ページアクセス方式を (1) とした方

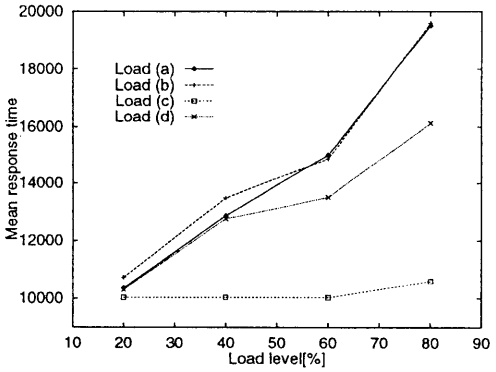

図 7: 平均応答時間 (MVA2 回，最大並列度 15)

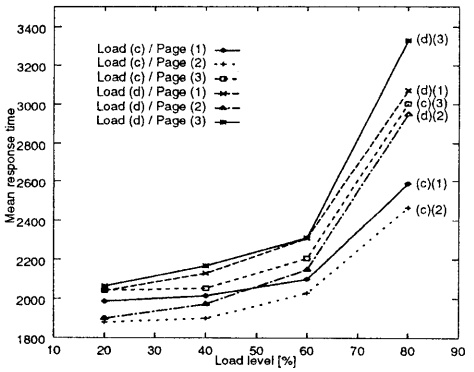

図 9: 平均応答時間（FJ 5 回，fork15）

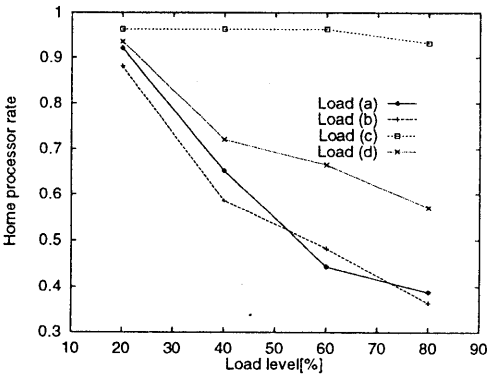

図 8: ホームプロセッサの割合 (MVA2 回，最大並列度 15)

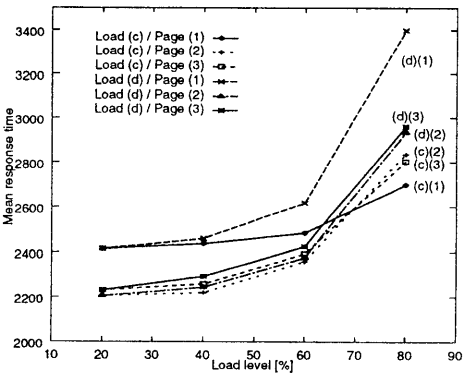

図 10: 平均応答時間（FJ 5 回，fork24）

式が良い．これは，負荷が重くなってシステム中のフリープロセッサの数が少なくなると，ローカルヒット率が低くなり，コピーのオーバヘッドがシステムに影響を及ぼしているためであると考えられる．

6.3 フリープロセッサの割り当て

次にフリープロセッサの割り当て時における，プロセッサの選択順序に関する選択肢の比較を行ない，図 11，12 にその結果を示す．

ページアクセス方式としては 5.2 節 (2)，プロセスのローディング方式としては 5.1 節 (c),(d) を用いている．

図 11を見ると，全体的に (B) 方式の方が優れている．これは，(B) 方式ではメモリ位置を考慮するために，プロセッサグループを構成する各プロセッサがまたがるクラスタ数は減少し (図 12)，局所化が行なえることにより，リモートメモリアクセスが削減できるからである．また，ホームプロセッサ，ローカルプロセッサを割り当てることによって，キャッシュミスした場合でもローカルメモリアクセスの可能性が高くなっていることも考えられる．

以上より，スケジューリングにおいてメモリ管理を考慮することは重要であり，システムの性能向上には欠かせないことがわかった．

7 おわりに

7.1 研究のまとめ

本稿では，NUMA マルチプロセッサを対象とするメモリ管理を考慮した 2 レベルスケジューリングにおいて，シミュレーションを用いて，方式 (プロセス空間のローディング方式，ページアクセス方式，フリープロセッサの割り当て方式) を評価した．

その結果，次のような結論に達した．

- プロセス空間のローディング方式にはスケジュー

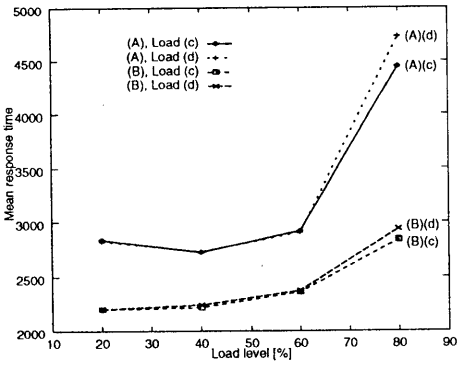

図 11: 平均応答時間 (FJ5 回, fork24)

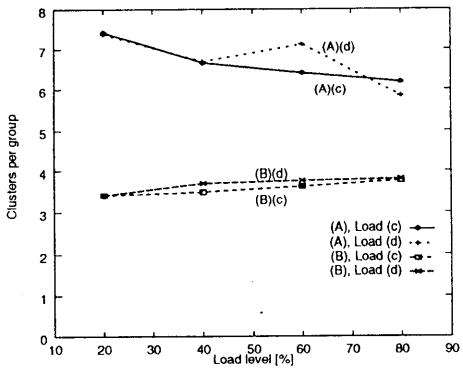

図 12: 平均使用クラスタ数 (FJ5 回, fork24)

ラからの情報提供を受けたフリープロセッサ数最大方式が優れている.

- フリープロセッサの割り当て方式としてはメモリ管理を考慮した方式が優れている.

7.2 今後の課題

今後の課題としては以下のようなものが挙げられる.

- メモリに関するモデルの詳細化.
  本研究では, 簡単のため, 実マシンとは異なった仮定をして, シミュレーションを行なった. 例えば, メモリ容量に関しては制限を設けておらず, 実際には二次記憶システムとの間でページイン・ページアウトが起こるであろう. ページの Read/Write の際にも, アクセスは逐次的とし理想化されている. また, ページングに関しても, Write 時の無効化の動作も考慮する必要があるであろう. これらのことについてモデルを改良すべきだと思われる.
- I/O のモデル化.
  プライベート・スケジューラによるスレッド間のスケジューリングでは, I/O リクエスト等が起こった場合にスレッド切替を起こすことが通常行なわれると考えられるが, 今回のシミュレーションでは I/O を無視したモデル上での評価となっている. 今後さらに I／O を含めて本スケジューリングを評価することは, 最も重要な課題であるといえる.

参考文献

[1] 甲斐久淳, 藤木亮介, 福田晃: マルチプログラミング環境のマルチプロセッサにおける 2 レベルスケジューリング—スケジューリング構造と性能評価—, 情報処理学会論文誌, Vol.35, No10, pp.2115-2127(1994).

[2] Cathy Mccann, Raj Vaswani, and John Zahorjan: A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors, ACM Transactions on Computer Systems, Vol. 11, No.2, pp.146-178(1993).

[3] R.P.LaRowe Jr.,C.S.Ellis,and,L.S.Kaplan: The Robustness of NUMA Memory Management,Proc.the 13th ACM Symp. on Operating Systems Principles,pp.137-151(1991).

[4] Timothy Brecht: On the importance of parallel application placement in NUMA multiprocessors, Proceedings of the USENIX Symposium on Experiences with Distributed and Multiprocessor Systems(SEDMS IV),pp.1-18(1993).

[5] 藤木亮介, 甲斐久淳, 福田晃: NUMAマルチプロセッサにおける 2 レベルスケジューリングアルゴリズムの評価, 情報処理学会「コンピュータシステム・シポジウム」, pp.67-74 (1993).

[6] 甲斐久淳, 藤木亮介, 福田晃: メモリ管理と協調動作する 2 レベルスケジューリング, JSPP'94, pp.319-326(1994).