オペレーティング・システム　57－3
（１９９２．１２．４）

処理コストに基づくページ置換方式の検討と実験

数藤義明　宮本剛　岩本信一　柴山茂樹

キヤノン株式会社 情報システム研究所

従来のページ置換アルゴリズムはシステム内の全てのページを平等に扱うものであり，実際には異なる処理コストを持つページの性質を生かしたものはなかった．本稿では，ローカルなページ中でもページ退避を行なわなければならないページと即座に解放可能なページとの処理コストの差異や，分散ファイルシステム上でページ毎の処理コストの差異を考慮し，置換選択を行なうページ置換アルゴリズムを提案する．さらに，このページ置換アルゴリズムを実装することによって，ページ置換が頻繁に発生するものに対して十分な効果があることを示す．

An Evaluation of a Cost-Based Page-Replacement Algorithm

Yoshiaki Sudo　Tsuyoshi Miyamoto　Shinichi Iwamoto　Shigeki Shibayama

Information Systems Research Center, Canon Inc.

Current page-replacement algorithms do not take into consideration the differences in page retrieval cost which depends on the nature of the type of memory object a page is in. The present paper presents a cost-based page-replacement algorithm and gives its evaluation. The algorithm selects pages to be replaced considering a priority attached to each page according to its retrieval cost. The evaluation shows that the paging system performance is improved at a maximum of 8% in situations where page-out is frequent.

1 はじめに

仮想記憶システムが提案されて以来，数多くのページ置換アルゴリズムの研究がなされてきた．これほどまでに，ページ置換アルゴリズムが重要になるのは，ページが主記憶中に存在する場合のデータのアクセスは数 ns から数十 ns で終了してしまうが，ページが主記憶に存在しない場合には，主記憶中に存在する他のページを 2 次記憶装置に退避して空きページフレームを作成したのちに，アクセスするページを 2 次記憶装置からローディングして数十 ms もの時間がかかってしまうからである．つまりページ置換アルゴリズムのシステム全体の処理速度に与える影響は重大なものである．

さらに，高速なネットワークの発達によって分散ファイルシステムが広く使われるようになった．分散ファイルシステムは，Sun の NFS に代表されるように，イーサーネットと TCP/IP を用いたネットワークによって接続されたマシン間でディスクを遠隔マウントすることによって，ファイルを共有できる機能をもち，ローカルなディスクと何ら変わりない機能を提供する．このような状況のなかで，大容量なディスク装置と高速なネットワーク・インタフェース，及び大容量のディスク・キャッシュを兼ね備えたファイル・サーバが普及し，それを中心として個人のワークステーションには最低限のファイルを保持するローカル・ディスクだけを持つという利用形態が広まっている．また，イーサネットよりも高速な光通信路を用いたネットワークなども実用化に向かっており，分散システムの重要性はますます高まっている．

また，2 次記憶媒体としてハードディスクの他に光磁気ディスクや CD-ROM，シリコンディスクなど，さまざまなものが利用されるようになった．これらは，単にデータ交換だけの目的ではなく，ワークステーションからファイルシステムとしても認識可能なものが多い．

以上のように，ハードディスク装置だけが唯一の 2 次記憶装置であった時代とは異なり，現在はファイル・システムとしてネットワークを通した分散ファイル・システムや光磁気ディスクなど数種類の装置が使用される状況になっている．つまり，システム内の仮想空間にマッピングされているページの保存されている場所が，ローカルなディスクの場合だけでなく，遠隔のマシンのディスクであったり，光磁気ディスク上であったり多様化してきた．このような場合，ページを退避するために必要なコストやページを再利用する場合に 2 次記憶装置からローディングしてくるためのコストは，そのページが属するオブジェクト毎に異なる．

本論文では，あるページを置換対象とした場合に，そのページを 2 次記憶装置に退避するために必要な処理コストと，そのページを再利用する場合に 2 次記憶装置からローディングするために必要な処理コストを考慮し，ページ置換時にページに優先度を持たせたページ置換アルゴリズムを提案する．さらに，実際に Mach 2.5 上に本方式を実装し，有効性を検証した．

2 ページ置換

2.1 ページ置換アルゴリズム

仮想記憶を持つオペレーティングシステムでは，フリーなページがない場合，システム内のページを 2 次記憶装置に退避することによって，プロセスに対して新たにページを割り当てる．この時，システム内のどのページを退避するかを決定するアルゴリズムをページ置換アルゴリズムと呼ぶ．現在までに，様々なページ置換アルゴリズムが研究されてきたが，UNIX のようなマルチプロセスのオペレーティングシステムでは，システム全体のページから選択を行なうグローバルなページ置換アルゴリズムが主に用いられている．ほとんどのページ置換アルゴリズムは，ページの過去の使用履歴を元にして，置換対象のページを選択するものである．代表的なものを以下に挙げる [1]．

- **最適アルゴリズム**
  将来最も遅く利用されるページを選択 (実現不可能)
- **Random アルゴリズム**
  乱数によってページを選択
- **FIFO アルゴリズム**
  主記憶に最も長くいるページを選択
- **LRU(Least-Recently-Used) アルゴリズム**
  最も長くアクセスされなかったページを選択

これらの中で，LRU アルゴリズムは，アプリケーションプログラムが時間的局所性を持つ場合に有効であり，実際に多くの OS で LRU を何らかの形で近似したアルゴリズムが使用されている．例として，4.3BSD UNIX では 2 本針のクロックアルゴリズム [2] と呼ばれるものが，Mach では FIFO with Second Chance アルゴリズム [3,4,5] と呼ばれるものが使用されている．これらについては後で述べる．

2.2 ページ退避方式

ページ置換アルゴリズムによって選択されてページを2次記憶に退避する方法として，ローディング時から修正が行なわれなかったページの処理によって，以下のように2つの方法が考えられる．

1. 修正されたページは2次記憶に退避して，修正されなかったページは即座に解放する．

2. 修正が行なわれたかどうかにかかわらずローディングしてきたページはすべて2次記憶に退避する．

単一の2次記憶装置からなるシステムの場合には，明らかに前者の方法のほうが効率が高いと考えられる．しかし，2次記憶装置として多くの装置が利用可能な場合には，後者の方法にはページを退避する2次記憶装置にアクセス速度の速いものが選択できるという利点が存在する．つまり，修正されなかったページを退避しない場合，そのページを再利用するためには元の2次記憶装置からローディングしてこなければならない．これは，元のページがアクセス速度の遅い2次記憶装置に存在した場合には，アクセス速度の速い2次記憶装置に退避した場合と比較して，どちらが効率が良いかという判断はできない．

例として，4.3BSD UNIX[2] では後者の方法が採られている．4.3BSD UNIX の場合には，あるディスクのパーティションの一つを，スワップデバイスとして，ページの退避用として割り当てる．これはスワップデバイスをファイルシステムとしてではなく，直接ディスクブロックからローディングすることにより，アクセスの速い退避用の2次記憶装置として使用できるからである．プロセスが生成された場合には，このスワップデバイス内に，そのプロセス全体をスワップ可能な領域が確保され，プロセス中のあるページが置換対象として選択された場合に，その領域に退避される．ただし，一度スワップデバイスに退避されたページで，その後に修正されていないページは退避しないという最適化は行なわれている．

逆に Mach[3,4,5] や Sprite[6] 等では前者の方法が採られている．これらの新しい OS では大容量のメモリを持った高速なプロセッサを対象としている．そのために，直接ディスクブロックを扱わずに，ファイルシステムとしてページをアクセスした場合のオーバーヘッドが無視でき，さらにバッファキャッシュの容量が大きいためにヒット率が高くなる．このようにファイルシステムを用いてページの退避を行ない，修正のないページは即座に解放し，再利用する場合は一様にファイルシステムからページを読み込む．

これら2つの方式の大きな差異は，分散ファイルシステム中のページの再ローディングについてである．4.3BSD UNIX では，上で述べたように一度ローディングしてきたページはローカルなディスクに退避するために，そのページを再利用する場合に再びネットワークを通してページを転送する必要はない．一方，Mach や Sprite では，一度解放したページを再利用する場合には，ネットワークを通して転送してくる必要がある．ただし，これらの OS では，ファイル・サーバとして大容量メモリと高速なネットワーク・インタフェース，高速なプロセッサを備えたものを前提としており，このページの再ローディングに関してはローカルなディスクと比較して遜色のないものであるという考えに基づいている．今後の分散システムでは，Mach や Sprite などと同様に，ファイルシステムにページを退避し，修正の無いページは即座に解放する方式をとることが主流になると考えられる．

3 処理コストに基づくページ置換アルゴリズム

今後の分散システムでは，ページ退避方式として修正の無いページは即座に解放し，そのページを再利用する場合には再びファイルシステムからローディングしてくる方法をとるものが主流であると考えられる．このような場合に，上で述べたように，再ローディングするページのあるファイルシステムの速度が，処理速度に影響を与える．例えば，低速なネットワークで接続されたホスト上にあるファイルよりも，ローカルな高速ディスク上のファイルからのローディングの方が速いであろう．逆に大容量のディスクキャッシュを備え高速のネットワークで接続されたホスト上のファイルよりも，低速なインタフェースで接続されたローカルなディスク上のファイルの方が遅いかも知れない．

3.1 基本的なアルゴリズムについて

このように，ページが存在するファイルシステムによって，そのページを再利用する場合のローディングに必要な分散システム全体の処理コスト (そのマシンにとっては処理時間) は異なる．また，修正されたページは2次記憶に退避しなければならないので，即座に解放の可能な修正されなかったページよりも処理コスト

は大きい．この処理コストの違いを利用して，以下のような方針に基づくページ置換アルゴリズムを考える．

1. そのページをページ置換の対象とした時に，ページを退避するための処理コストと，そのページを再利用する場合に再びローディングするために必要な処理コストを考慮して優先度を決定する．
2. 単純に優先度に従ってページ置換の対象を決定すると，あるファイルシステムに属するページばかりがページ置換されてしまうので，優先度の低いページも何らかの基準に従ってページ置換の対象となるようにする．
3. 同じ優先度のページに関しては，LRU に基づいてページ置換の対象を決定する．

最初の方針は，ページ退避処理と再ページイン処理の処理コストによって優先度を決定するという，このアルゴリズムの基礎となる方針である．例えば，現在，一般的に利用されている分散ファイルシステムを持つマシンに関して言うと，以下のような優先順位が考えられる．

1. ローカル・ファイルシステム (修正のないページ)
2. 遠隔ファイルシステム (修正のないページ)
3. ローカル・ファイルシステム (修正のあるページ)
4. 遠隔ファイルシステム (修正のあるページ)

しかし遠隔ファイルシステムでも高速な 2 次記憶装置や高速なネットワークを持つファイルサーバを利用できる場合は優先順位の逆転がおこる可能性がある．

2番目の方針は，上記のように優先度を決定した場合に，ローカル・ファイルシステム上の実行ファイル (ページ内容の修正は行なわれない) ばかりがページ置換の対象となってしまうのを防止するものである．また，3番目の方針は，優先度が同じページに関して，どのページをページ置換対象とするかを選択するかは LRU に基づいたものとし，プログラムの時間的な局所性を生かすようにするものである．この 2 つの方針を合わせて考えた場合，以下のような処理コストを考慮した LRU アルゴリズムが考えられる．

- LRU でページ置換の選択を行なうために用いる時間に関する尺度を優先度に応じて決める．
- 上で決定した尺度によって，最も長く主記憶にいるページをすべての優先度のページ中から探し，それをページ置換対象とする．

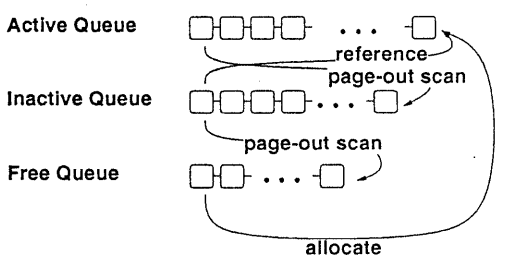

図 1: FIFO with Second Chance アルゴリズム

3.2 Multi-Level FIFO with Second Chance アルゴリズムについて

実際のハードウェアでは，そのページを最後にアクセスした時間が分からないので，上で述べたアルゴリズムを何らかの形で近似しなければならない．以下のように LRU を近似している方法に拡張を加えた形で実装するのが適当である．

3.2.1 FIFO with Second Chance アルゴリズム

Mach で採用されている FIFO with Second Chance アルゴリズムは，図1に示すように3つのキュー (active, inactive, free) を用いている．使用中のページは最初は active キューに入れられ，free キューに属するページが少なくなった時点でページ置換が起こり inactive キューからページ置換するページを探す．inactive キューは長さを一定に保たれ，キューにつながれているページが少なくなってきた場合には active キューの先頭からページを持ってくる．その時に参照ビットをクリアしておき，ページ置換アルゴリズムがページ置換対象を選択する場合には，inactive キューの先頭からページを外し，参照ビットがオンになっている場合には再び active キューに返し，オフの場合にはページ置換の対象とする．

3.2.2 Multi-Level FIFO with Second Chance アルゴリズム

FIFO with Second Chance アルゴリズムを拡張した Multi-Level FIFO with Second Chance アルゴリズムを提案する．上で述べた処理コストを考慮した LRU アルゴリズムを，この FIFO with Second Chance アルゴリズムに反映させるためには，inactive キューを優先度ごとに複数持ちその長さを変える方法が考えられるが，常にその優先度のページが多く存在するわけで

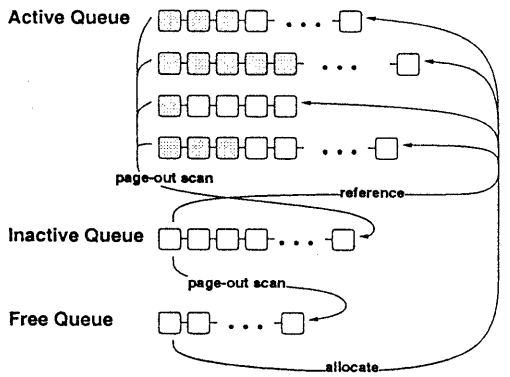

図 2: Multi-Level FIFO with Second Chance アルゴリズム

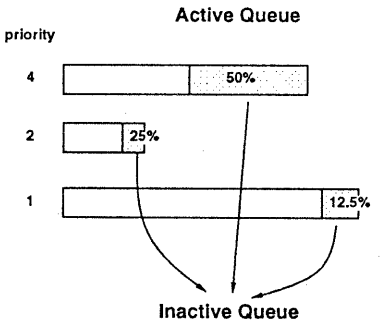

図 3: Multi-Level FIFO with Second Chance アルゴリズムの active キューからのページ取り出し操作

はないので inactive キューの長さの管理が困難となる．そのために図 2に示すように，active キューを優先度ごとに複数用意し，その中から inactive キューに入れるページ数を優先度に応じて変化させることによって，FIFO(active キュー) 中のページの移動速度を変化させ，上記のような処理コストを考慮した LRU アルゴリズムを近似する．この active キューからページをとり出す場合に，active キューの長さがそれぞれの優先度のキューによって異なるために，図 3のように active キューの長さと優先度毎に設定された倍率に従って，active キューから取り出すページの割合を優先度の高いものは大きくする．

4 実験と評価

Mach2.5 に上で述べた Multi-Level FIFO with Second Chance アルゴリズムを実装した．計測には，当研究所で試作したワークステーション Stonehigh[7] を使用した．Stonehigh は CPU にモトローラ MC68040 を用いたマルチプロセッサ・システムである．ページサイズは 8KB である．

4.1 ページの処理コストの測定

一般的なワークステーションではファイルシステムとして，ローカルディスク上の UFS とネットワークで接続されたマシン上のディスクをマウントする NFS を使用可能である．今回の計測では，UFS と NFS(サーバマシンは SUN3/80) という 2 種類のファイルシステムを使用した．さらに，Mach ではファイルシステム中のファイルを仮想空間にマッピングしたものに関しては書き換えが不可能であるため，NFS に対してページの退避は行なわれないし，実行ファイルのメモリオブジェクトに対しても書込みは行なわれない．よって，以下の 3 つの種類のメモリオブジェクトに対してディスクへの退避及びローディングの処理コストを測定する．

- UFS 上の修正不可能のメモリオブジェクト (text)
- UFS 上の修正可能のメモリオブジェクト (heap と stack)
- NFS 上の修正不可能のメモリオブジェクト (text)

ページのローディングに関しては，ローディング時間を測定した (図 4)．ページの退避に関しては，バッファキャッシュがあるために，処理コストがページのライト時間として簡単に測定できないので，参考としてバッファキャッシュを最小 (16 ページ) にした場合のページ単位の平均処理時間を示した．これによると，UFS からのローディングでは単一プロセスを実行した場合より，複数のプロセスを一度に実行した場合の方が時間がかかる．単一プロセスの場合はローディングの要求の頻度が少なく，前のローディングが終るまで他のローディング要求は発生しない．一方，複数プロセスの場合にはディスクアクセスの頻度が高いために，UFS ではデバイスドライバで要求がキューイングされてしまうために遅くなる．逆に NFS からのローディングでは，(a) と (b) のどちらも処理速度はほとんど同じである．NFS ではサーバ側のディスク速度が，UFS の場合と比較して速く (実測で約 18ms/page:ローカルディスクは

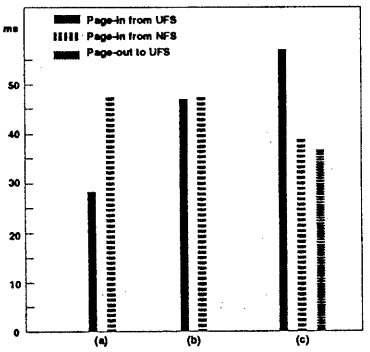

図 4: ローディング時間及びページ退避時間の測定結果: (a) 単一プロセス立ち上げ時 (ディスクアクセス頻度平均約 10 回/秒), (b) 複数プロセス立ち上げ時 (ディスクアクセス頻度平均約 20 回/秒), (c) 主記憶が不足した状態で複数プロセス立ち上げ時 (ディスクアクセス頻度 UFS 平均約 53 回/秒, NFS 平均約 23 回/秒)

約 22ms/page), かつサーバ側の負荷は UFS ほど増さないため, NFS からのローディングは遅くならないと考えられる.

また, 主記憶容量を小さくして, ページ置換が頻繁に行なわれるような環境で複数のプロセスを実行した場合には, UFS からのローディングはさらに遅くなるが, NFS からのローディングは逆に速くなった. 測定した X window の立ち上げでは, UFS 上に X window がある場合には UFS へのアクセス頻度は高く, さらにディスク要求が遅くなる. 逆に NFS 上に X window がある場合には, それほどページ置換が頻繁になっても NFS へのアクセス頻度は上がらない. UFS に対してページ退避のための書込み要求が発生してしまうためであり, また NFS 環境でも通常の使用状況であるとデーモンやシェルは UFS に存在し, その他のアプリケーションプログラムが NFS にあるために, それほど NFS に集中してローディング要求が発生するわけではないからである. また, NFS のファイルのページを解放してしまった場合に, 再度ローディングする場合にはサーバ内のバッファキャッシュにそのページが残っていることが多く, この場合のローディングは要求の約 40 %が 30ms 以内に戻ってきている (単一プロセスのローディングの場合は約 20 %). そのため, NFS では平均のローディング速度が速くなった.

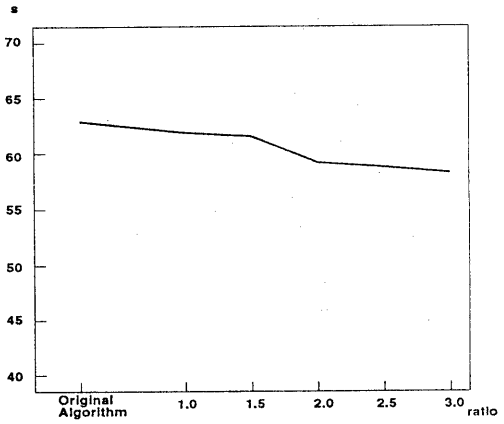

図 5: UFS に X window の実行ファイルを置いた場合の X window の立ち上げ時間の測定結果

4.2 UFS だけを用いた実験

UFS だけを用いて, X window(X server, twm, xload, oclock, xterm, exterm) を立ち上げる時間と, さらにその上で SPECmark92 内の eqntott を実行し X window を終了する時間を測定した. ページ置換の優先度は, 修正の行なわれるページ (heap 領域やスタック領域など) と修正の行なわれないページ (text 領域など) の 2 種類について倍率で設定する. 図 5と図 6が, 修正の行なわれないページを何倍の割合で inactive キューに入れるかを横軸にとった測定結果である. オリジナルのページ置換アルゴリズムを用いた場合に比較して, 図 6では優先度が 2 倍のところが最適となった. また, 図 5では修正のないページの優先度をあげていくにつれて実行時間が減少している. この計測の場合, ページ置換はほとんど (約 90 %) がこの X window の立ち上げ時に発生する. つまり, この X window の立ち上げ時間が, このページ置換を用いたことによる影響を最も受ける. ただし, 修正のないページの優先度を高くした場合, X window の立ち上げ速度は上がるが, 立ち上がった後に, 立ち上げ時に解放してしまった text ページなどのローディングのための処理をしなければならなくなる. このため X window 上でアプリケーションを実行させると, そのアプリケーションプログラムが影響を受け, 図 6のように実行速度が遅くなってしまい, X window 終了までのトータルの時間も遅くなってしまう.

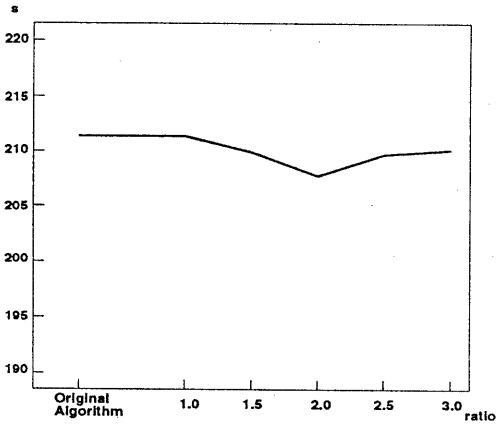

図 6: UFS に X window と eqntott の実行ファイルを置いた場合に，X window を立ち上げ eqntott を実行させて X window を終了させた時間の測定結果

4.3 NFS を用いた実験

同様に NFS も用いて 4.2 の UFS を用いた実験と同じ実験を行なった．今回は X window の実行ファイルすべてを NFS によってマウントされたファイルシステム上に置いた．この場合の，ページ置換の優先度は以下のように 2 種類に決定した．

1. UFS に退避しなければならないページや UFS 上にある実行ファイルのページ

2. NFS 上にある実行ファイルのページ

図 7と図 8が，(2) のページを (1) のページの何倍の割合で inactive キューに入れるかを横軸にとった測定結果である．

この実験では，オリジナルのページ置換アルゴリズムを用いた場合に比較して，図 8では優先度が 4 倍のところが最適となった．UFS だけを用いた実験と同様に図 7では NFS のページの優先度をあげていくにつれて実行時間が減少している．この場合も同様に text のページを多く解放している状態であるが，再ページインの処理コストが UFS の場合と比較して小さいために，その上で実行されるアプリケーションプログラムの実行速度の影響が少ない．

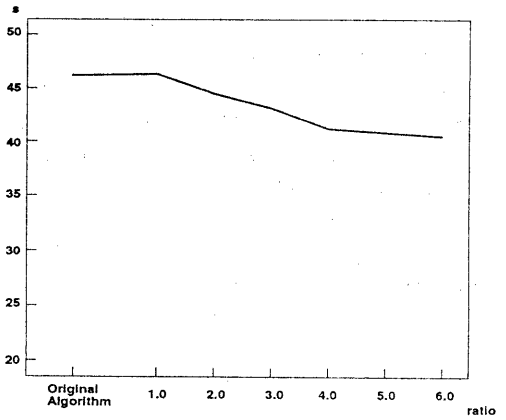

図 7: NFS に X window の実行ファイルを置いた場合の X window の立ち上げ時間の測定結果

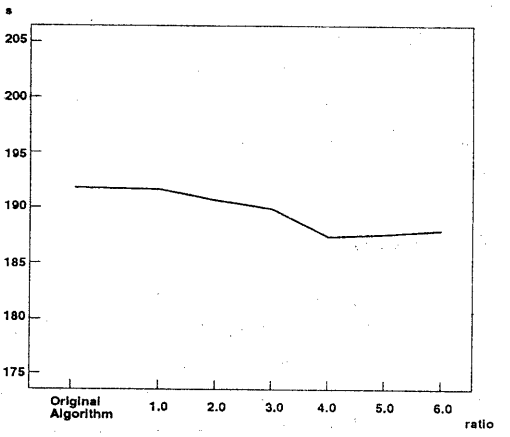

図 8: NFS に X window の実行ファイル，UFS に eqntott の実行ファイルを置き，X window を立ち上げ eqntott を実行させて X window が終了するまでの時間の測定結果

5 考察

行なった実験のうち，一つはローカルディスク上のUFSだけを用いて，修正され退避しなければいけないページと，修正がなく即座に解放可能なページとの優先度を変化させることによって，X window上でアプリケーションプログラムを実行する実験である．この場合に，ページ置換の頻繁に発生するX windowの立ち上げ時に処理時間が短縮できた．しかし修正がないページの優先度を2倍より大きな値にすると，逆にX window上で動作させるアプリケーションプログラムの実行速度が遅くなるが，優先度を2倍とすることで，アプリケーションプログラムの実行時間に影響がなく，X windowの立ち上げ速度を約8％速くできた．

さらに，NFSを用いてNFS上のページとUFS上のページ(UFS上に退避するページも含む)の優先度を変化させて，上記と同じようX window上でアプリケーションプログラムを実行する実験を行なった．最初にページのローディング速度を計測した結果，ページ置換が頻繁になりUFSに書込み要求が多く発生する場合には，UFSからのローディングはNFSからのローディングよりも遅くなるということが分かった．このためにNFSのページの優先度をUFSのページよりも高くして実験を行なった．X windowの立ち上げ時間が短縮でき，かつアプリケーションプログラムの実行に悪影響がないことを確認した．また，この実験の場合には，NFSのページの優先度をかなり高い倍率に上げた場合でも，X window上でアプリケーションプログラム動作させる全体の実行速度が遅くはならないことが分かった．

6 おわりに

従来のページ置換アルゴリズムは，全てのページについてページ退避と再ローディングの処理コストが等価であるとの仮定に基づいていた．しかし，ページ退避をファイルシステムに行ない，修正の行なわれていないページは即座に解放するような方式をとるOSにとっては，ページ退避と再ローディングの処理コストは大きく異なる．また，分散システムに於ては，ローカルディスク上のファイルシステム上のページと分散ファイルシステム上のページで，再ローディングの処理コストも異なる．

本稿では，これらの処理コストの差異を利用し，退避の処理と再ローディングの処理のコストが小さいページに対して，優先的にページ置換の対象となるようなページ置換アルゴリズムを提案した．さらにそのような処理コストを考慮したページ置換アルゴリズムをMach 2.5上に実装し，その有効性を示した．

今後の課題としては，さらに処理コストの差異が大きなファイルシステムに対して，このページ置換アルゴリズムがどのくらいの有効性を持っているかの検証を行なうことや，クロックアルゴリズムを拡張した実験などを行なっていく予定である．

参考文献

[1] A. S. Tanenbaum, *Modern Operating Systems*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992.

[2] S. J. Leffler, M. K. McKusick, M. H. Karels and J. S. Quaterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Reading MA: Addison-Wesley, 1989.

[3] A. Tevanian, "Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach," Ph.D. thesis, CMU, 1987.

[4] R. Rashid, A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. J. Bolosky and J. Chew, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," *IEEE Trans. Comput.*, C-37, pp. 896-907, Aug. 1988.

[5] R. P. Draves, "Page Replacement and Reference Bit Emulation in Mach," in *Proc. Usenix Mach Symposium*, 1991, pp. 201-212.

[6] M. N. Nelson, "Virtual Memory for the Sprite Operating System," CSD Tech. Report No. UCB/CSD 86/301, University of California, Dec. 1986.

[7] 伊達, 濱口, 出井, 柴山, "マルチプロセッサワークステーション "Stonehigh" – コンセプトとハードウェア概要 –," 情報処理学会第45回全国大会予行集, 1992.