解　説

仮想記憶の制御方式†

益　田　隆　司††

1. まえがき

1959年，英国マンチェスター大学の ATLAS システムで芽生えた仮想記憶方式は，1965年に発表された米国 MIT Project MAC の MULTICS システムで本格的に採用され，さらに，1970年代にはいって，IBM 370 シリーズをはじめとする数多くの商用システムで実用化されている．この中でも，仮想記憶方式の技術の発展に大きな影響を与えたのは MULTICS システムであり，同システムによって，セグメンテーション，ページングの機能を有する仮想記憶方式の利点が広く認識された．しかしながら，実際にシステムを開発してみると，仮想記憶方式の実現に伴うオーバヘッドが大きいこと，特に，ページング方式がシステムの性能に及ぼす影響がきわめて大きいことが明らかになり，1970年代にはいる頃までは，この技術を疑問視する意見もかなり多くあったように思われる．

この間，この問題に刺激されて，ハードウェアのアドレス変換方式，オペレーティング・システムの主記憶制御方式の研究が積極的に行われた．この研究の過程で，システムが性能的に効率よく動作しないのは，仮想記憶の融通性が大きいことを利用しシステムを過負荷の状態で動作させていたり，主記憶制御方式に問題があったりすることも次第に明らかになった．

1970年代にはいってからの仮想記憶の急速な普及の背景には，ハードウェアの記憶技術の発展が寄与するところが大きい．MULTICS 前後のシステムでは，主記憶容量は高々1メガ (10^6) バイト程度であり，数10端末を同時に利用するには主記憶量が絶対的に不足する傾向にあった．多重プログラミングの水準を上げると個々のプログラムへの割当て主記憶量が不足し，逆に，個々のプログラムへの割当て主記憶量を増加させると多重プログラミングの水準が下がってしまうことになり，主記憶容量の不足がシステムのスループット，応答性の低下の大きな要因となっていることが多かった．しかしながら，その後の記憶技術の発展により，最近のシステムでは，主記憶容量も大幅に増大し，仮想記憶の制御も余裕をもって行うことができるようになりつつある．

このような背景の中で，MULTICS システムの発表以後，仮想記憶の制御方式について数多くの研究結果が発表されており[91]，本報告ではその概要を述べてみたい．

2. 主記憶制御方式

仮想記憶システムでは，主記憶の制御方式がシステム全体の性能に大きな影響を及ぼす．主記憶制御の役割は，システムへの負荷に応じた最適な多重度の決定，各プログラムに割り当てる主記憶量の決定，および，主記憶内に保持すべきページの決定等を行うことである．これらの制御を適正に行うことができずに，多重度を上げすぎると，ページ・フォールトが異常に多く発生する，いわゆるスラッシング (thrashing) 現象を生じたり，逆に，多重度を下げすぎると，数少ないプログラムに不必要に多くの主記憶を割り当てたりする可能性が大きくなる．

仮想記憶の主記憶制御に関しては，これまでに数多くの方式が提案あるいは実現され，同時に，それらの理論的な解析，実システムでの有効性の検証等が行われている．これら制御方式に関する検討を行うことが本報告の目的であるが，そのためには，まず，仮想記憶システムのもとで動作するプログラムの基本的な動作特性を把握しておかねばならない．

プログラムの動作特性に関しても，制御方式同様，多くの研究の成果が発表されているが，ここでは以降の制御方式の検討に必要な範囲だけを簡単に述べておくに留める．

プログラムが使用するアドレス空間の大きさが割り当てられた主記憶量よりも大きい場合には，プログラムの実行中，主記憶内のページ枠がすべて利用されて

† Memory Management Strategies for Virtual Memory Systems by Takashi MASUDA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学電子・情報工学系

いるときに，さらに，主記憶内に存在しないページが参照されることがある．この場合には，そのページを読み込むために主記憶内のいずれかのページを補助記憶装置にページ・アウトする必要が生じる．いずれのページをページ・アウトするのがよいかについて，いくつものアルゴリズムがあるが，一般に実現可能なアルゴリズムとしては，LRU (Least Recently Used) 置換えアルゴリズムがすぐれていることが知られている[1),24)]．これは，プログラムが有する主記憶内のページを時間的に最近に参照したものから順々に並べたときに，最も遠い過去に参照したページをページ・アウトするものである．LRU アルゴリズムは，一般に，プログラムには，その実行中，時間的に近い二つの部分で参照しているページには互いに一致しているものが多いという性質が存在することを利用したアルゴリズムである．しかしながら，プログラムの実行に伴うアドレス参照特性をさらに解析してみると，多くのプログラムでは，上記の二つの部分が時間的にある程度離れると，その両者で参照しているページの集合の中に共通に含まれるページは少なくなるという性質が存在している．

プログラムのアドレス参照に関するこれらの性質は，プログラムの局所参照 (locality of references) の性質とよばれている[11)]．そして，プログラムがある時間内定常的に参照しているページの集合を局所参照集合 (locality set) とよび，プログラムの実行が一つの局所参照集合内に留まっている範囲をプログラムのフェイズとよぶ．プログラムの実行は，時間的に利用するフェイズを次々に変えていくという形をとる．

プログラムには，基本的に以上のような性質が存在すると考えられている．主記憶制御方式は，このようなプログラムの動作特性を考慮にいれた方式であることが望ましい．

仮想記憶システムの主記憶制御方式を主記憶割当ての観点から分類してみると[13)]，

（1） 各プログラムに割り当てる主記憶量を，プログラムが開始してから終了するまでの間一定に保つ固定割当て法，

（2） 各プログラムに割り当てる主記憶量を時間の経過と共に変化させる変動割当て法，

に分けられる．

固定割当て法が現実のシステムで用いられることはないようであるが，固定割当て法は，従来の実記憶システムでの主記憶割当て方式の延長として考えやすいこと，多重プログラミングの環境下でも理論的な解析が可能であること，あるいは，変動割当て法の効率を論ずる場合，その比較の対象として考えられること等の理由から，引用あるいは解析の対象とされる場合が多い．

変動割当て法には，個々のプログラムの局所参照特性を考慮した変動割当て法と，それを考慮しない変動割当て法がある．

プログラムがあるフェイズの中でページ・フォールトの頻度が少なく効率よく実行されるためには，そのフェイズの中で利用しているページを主記憶内に用意すればよい．したがって，各時刻でのプログラムの局所参照集合を推定することができれば，効率のよい主記憶制御が可能になる．これが，個々のプログラムの局所参照特性を考慮する変動割当て法の根拠である．

プログラムの局所参照集合の推定量として最もよく知られているものは，Denning が提案したワーキング・セット[9),10),12),17)]である．あるプログラムの時刻 t におけるワーキング・セット $W(t, T)$ とは，そのプログラムが時刻 t から過去 T 時間（そのプログラムが使用した CPU 時間だけを累積して）の間に参照したページの集合である．T をウィンド・サイズ (window size) とよんでいる．また，$W(t, T)$ の中の異なったページの数 $w(t, T)$ をワーキング・セット・サイズとよび，その時間平均を平均ワーキング・セット・サイズとよんでいる．

後述の主記憶制御方式としてのワーキング・セット方式は，アクティブ・プログラムにはすべて，そのワーキング・セットを主記憶内に保証しようとするものである．その場合，ウィンド・サイズ T をどのように設定するかが一つの問題になる．

ワーキング・セットの考え方は，仮想記憶の制御方式に関する数多くの研究の中で，常にその中心に位置しており，ワーキング・セットの理論的な性質，局所参照集合の推定量としての有効性，あるいは，それに基づいた制御方式等について，これまでに数多くの研究が発表されている．

本章では，仮想記憶の制御方式に関する基本的な考え方を述べた．次章以降に，固定割当て法，変動割当て法に属する個々の制御方式の特徴等を具体的に述べる．

3. 固定割当て法

固定割当て法の場合，各プログラムに等しい大きさの主記憶を割り当てる方法と，それぞれに異なった大きさの主記憶を割り当てる方法が考えられる．個々のプログラムの局所参照集合の大きさに合わせて主記憶を割り当てれば，それぞれに等しい量の主記憶を割り当てるよりも処理能力が向上することは想像できる．しかしながら，個々のプログラムのページ要求特性を全く考慮しなくても，あるいは，すべてのプログラムの特性が同一の場合にも，それぞれのプログラムに異なった大きさの主記憶を割り当てた方が処理能力が向上する場合が存在することが確かめられている．Belady & Kuehner[2] は次のような実験結果を報告している．

多重プログラミング・システムが動作中に，あるプログラムに注目し，システムが p 回のページ・フォールトを生ずるまでは，注目したプログラムのページはページ・アウトしないように制御する．その間は，そのプログラムに割り当てられる主記憶ページは，次第に増加する．システムが p 回のページ・フォールトを検出すると，別のプログラムを新たに注目するプログラムとして選択し，同様の手順を繰り返す．このときには，直前に注目していたプログラムのページはより頻度多くページ・アウトされることになる（ページ置換えアルゴリズムとしては，FIFO [First In First Out] を利用している）．このように個々のプログラムに割り当てる主記憶量を時間的に変動させることにより，ほぼ主記憶を等分割して割り当てた FIFO アルゴリズムに比較して，処理能力が 10～15 パーセント向上したと報告している．Belady 等は，この原因は，割当て主記憶量とページ・フォールト間の時間間隔の関係を示すライフタイム (lifetime) 関数が下に凸の性質を有している傾向があるためであり，割当て主記憶量に差を設けることにより，ライフタイムの平均が等分割の場合に比較して長くなるためであると説明している．

固定割当て法の場合に，等分割がよいのか，各分割の大きさに差を持たせるのがよいかという上記の問題を，Ghanem[20] は，待ち行列モデルを用いて，多重プログラミングの効果を考慮し理論的に取り扱っている．

最初に，主記憶内に二つのプログラムだけがはいっている場合を考える．ライフタイム関数の形を Rx^{α}

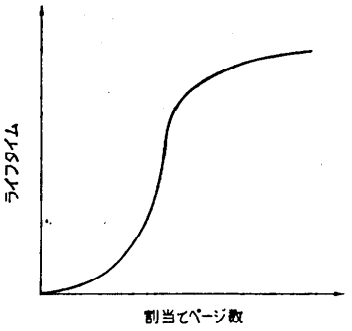

図 ライフタイム関数

(x：割り当てた主記憶量；$\alpha>1$) とすると，α がある α_0 より小さい場合には，主記憶を等分割して割り当てるのが CPU 利用率を最大にし，α が α_0 より大きい場合には，一方のプログラムに全主記憶を割り当てるのが最適であることを示している．Ghanem はこの方法を，プログラムの個数が N 個存在する場合に拡張して，すべてのプログラムのライフタイム関数が等しく，x^{α} ($\alpha>1$) に比例する場合の主記憶の最適な割当ては，N 個の内の q 個 (q：最適の多重度) のプログラムに等しい量の主記憶を割り当てることであることを示している．現実のプログラムのライフタイム関数は，図に示すように，ある程度の量の主記憶を割り当てるとそれから先は，主記憶量を増してもライフタイムの増加の割合が小さくなり，下に凸から上に凸の形をとる場合が多い．Ghanem はこれを近似して，ライフタイム関数が $x\geq m_0$ では一定の値 Rm_0^{α} をとるとして最適な主記憶の割当て法を求めている．この場合には，N 個のうちの主記憶を割り当てる q 個のプログラムには，ある一定の主記憶量 m または m_0 の主記憶を割り当てる方法が最適であることを示している．

これらの結果はいずれも，ライフタイム関数の下に凸の程度と，多重プログラミングによるオーバラップの効果の関係によって影響を受けることになる．ライフタイム関数の下に凸の性質を仮定し，オーバラップの影響を考慮しないと，二つのプログラムで考えれば一方のプログラムに主記憶をすべて割り当てた方がライフタイムは平均的に長くなるが，多重プログラミングによるオーバラップの効果はこれと逆の方向に働くことになる．

Ghanem は以上のような解析を，入出力装置のサービス特性として，窓口の数がプログラムの数 N よりも大きく入出力に待ちが生じないモデルを仮定して行

っている．Spirn[32]は同様の解析を，ライフタイム関数は下に凸の関数，入出力装置のサービス特性は独立な単一窓口の待ち行列を有限個集めたモデルを仮定して行い，ある条件のもとでは，等分割の場合がCPUの利用率が最も悪いという結果を得ている．さらに，Fin & 亀田は，ライフタイム関数が主記憶量の増加に伴い，下に凸から上に凸に変化し，また，入出力装置のサービス特性についてもより一般的なモデルを仮定して解析を行い，Ghanemと同様の結論が得られることを示している．

固定割当て法の利点は，それを実現したときのオーバヘッドが少ないことである．しかしながら，プログラムの実行に伴って生ずるフェイズの移動による局所参照集合の大きさの変化は，この利点を打ち消す方向に働く．時間の経過と共に，アクティブ・プログラムの集合 $(P_1, P_2, \cdots, P_q)$ のそれぞれの局所参照集合の大きさの変動が大きいと仮定する．P_i には m_i 個のページ枠が，P_j には m_j 個のページ枠が割り当てられているとし，ある時刻で P_i の局所参照集合の大きさが m_i より小さく，P_j の局所参照集合の大きさが m_j より大きいとする．このような場合にも，固定割当て法では，これら相互の間でページ枠を融通しあうことができない．

Coffman & Ryan[7]は，各プログラムに等しい大きさの主記憶量を割り当てる固定割当て法と，プログラムのワーキング・セットを主記憶内に保証する変動割当て法の比較を，ワーキング・セットの大きさが正規分布に従うとの仮定のもとに統計的に解析した結果を報告している．固定割当て法の場合には，各プログラムに割り当てられた主記憶量を相互に融通できないので，各々のプログラムのワーキング・セットの大きさが割り当てられた主記憶量よりある程度小さくても，すべてのプログラムのワーキング・セットの大きさが，それぞれに割り当てられた主記憶量を越えない確率は小さくなる．これに対して，変動割当て法であるワーキング・セット法では，アクティブ・プログラムのワーキング・セットの大きさの総和が全主記憶量を越えなければよい．個々のプログラムのワーキング・セットの平均の大きさが，それぞれに割り当てられる平均の主記憶量を越えなければ，ワーキング・セットの大きさの総和が全主記憶量を越える確率は小さくなる．Coffman 等はこれらの確率を計算し，固定割当て法が効率よく動作するのは，ワーキング・セットの大きさの変動が小さく，個々のプログラムに割り当てられる主記憶量がワーキング・セットの平均の大きさよりも 2σ（σ：標準偏差）程度大きい場合であると述べている．また，この場合でも，同一の処理能力を得るのに変動割当て法の主記憶量が固定割当て法に比較して，30パーセント程度少なくて済むことを示している．さらに，ワーキング・セットの大きさの変動がある程度以上大きい場合には，変動割当て法の主記憶利用効率が固定割当て法に比較して格段にすぐれているという結論を得ている．Oden & Shedler[26]もCoffman 等とは異なったアプローチで，これらとほぼ同様の結論を得ている．

4. 変動割当て法

先に述べた Belady & Kuehner の割当て法も変動割当て法の一つであるが，そこでは主記憶の割当てを個々のプログラムのページ要求特性とは独立に行っている．より望ましい方法は，プログラムのページ要求特性にあわせて主記憶の割当てを行う方法であり，いくつかの方法が提案あるいは実現されている．これらは，個々のプログラムの局所参照特性を検出し，それに基づいて主記憶の割当て，多重度の制御を行うものと，個々のプログラムの局所参照特性は特に意識せず，システム全体の負荷に基づいて多重度の制御等を行うものの2種類に分けられる．

4.1 プログラムの局所参照特性を考慮しない変動割当て法

この割当て法に属する最も典型的な方法は，グローバル LRU 法である．グローバル LRU 法では，主記憶全体を一つの LRU スタック（最近に参照したページほどスタックの上位に位置させる）で管理する．ページ置換えの必要が生じたときには，LRU スタックの最下位のページから置換えページを選択する．グローバル LRU 法は，後述のワーキング・セット法と並んで，現実のシステムで最も数多く採用されている主記憶制御方式である．LRU スタックを正確に実現するには，主記憶装置への参照がある度にスタックの内容を書き換えねばならず，実現が難しいので，現実には LRU 方式を近似した方法が用いられる．グローバル LRU 法を用いた場合の多重度の制御は，システム全体のページ・フォールト率に基づいて行われる場合が多い．

単一のプログラムのページ参照特性は，2章でも述べたように，LRU 的な性質を有している場合が多いことが確かめられているが，CPU のサービスを受ける

プログラム（アドレス空間）の実行順序は，スケジューリングの方法に依存するので，グローバル LRU 法が主記憶制御方式として必ずしも根拠のある方式とはいえない．この点に関して，実際のシステムでグローバル LRU 法が特に問題なく動作しているのは，

（a） ページング入出力，ファイル入出力等，高速の入出力装置に対する要求が生じた場合には，入出力が終了し，プログラムが再び CPU のサービスを受ける際にも，そのプログラムに属していたページは LRU スタック内に残っている可能性が強いこと，

（b） 端末入出力，タイム・スライスの終了のように，プログラムが再び CPU のサービスを受けるまでに平均的にかなりの長時間を要するものについては，それまでにそのプログラムに属していたページが，ページ・アウトされてしまっていても特に差し支えないこと，

のためと考えられる．

グローバル LRU 法に近い主記憶制御方式として，MULTICS[8]，OS 7[27]，VM/370 等で用いられているグローバル FINUFO (First In Not Used First Out) 法がある．時計の針のような「現在の位置」を示すポインタをつけて，主記憶内のページングの対象となるすべてのページを，一つの円周上に並べる．各ページは参照されるとハードウェアによって参照ビットがセットされる．ページ・フォールトが生じたとき，ページ置換えを行うためには，上記のポインタを順に回転させていき，最初に見つけた参照ビットが 0 のページをページ置換えの対象として選択する．新しく読み込んだページは同位置にセットする．ポインタを回転させていく過程で参照ビットが 1 にセットされているものについてはそれを 0 にしておく．

ページ置換えアルゴリズムとして，FIFO は，実現が容易であることが利点であるが，2 章で述べたようなプログラムの動作特性を考えるとすぐれたアルゴリズムではない．一方，LRU アルゴリズムは，前述のように実現が難しい．上記の FINUFO は，LRU に近いアルゴリズムを FIFO の容易さで実現したアルゴリズムである．

個々のプログラムの局所参照特性を考慮しない変動割当て法としては，上記以外に，Belady & Tsao[4] の提案した "AC/RT" 法がある．この方法は，各プログラムごとに，それぞれに割り当てられた主記憶をどの程度有効に利用しているか，および，近い過去にページ置換えの対象としたページのうち，再び参照されたものがどの程度存在したかを調べ，この両者の組み合わせにより，置換えの対象とするページを選択するものである．

以上，個々のプログラムの局所参照特性を考慮しない変動割当て法について述べたが，この制御方式の欠点は，主記憶の割当ての制御と多重プログラミングの水準の制御が別々に行われていることである．多重度の制御は，システム全体のページ・フォールト率等によって適当に行われることになる．

4.2 プログラムの局所参照特性を考慮した変動割当て法

変動割当て法の今一つは，個々のプログラムの局所参照特性を考慮したもので，Denning の提案したワーキング・セット法がその典型である．ワーキング・セット法では，個々のプログラムの局所参照集合の推定量として，2 章で述べたワーキング・セットを用いる．ウィンド・サイズ T が制御パラメータである．アクティブ・プログラムにはすべて，そのワーキング・セットを主記憶内に保証する．いいかえれば，ワーキング・セット法では，主記憶内のページは，アクティブ・プログラムの各時点での局所参照集合の推定量である．

ワーキング・セット法の今一つの大きな特徴は，その制御方式から，多重度の制御が同時に行われてしまう点である．アクティブ・プログラムのワーキング・セットは常に主記憶内に存在しなければならないので，その大きさの合計が全主記憶量を越えない範囲で多重度が決定される．ワーキング・セットの大きさの合計が主記憶量を越えると，いずれかのアクティブ・プログラムがインアクティベートされ多重度が下がる．

上記の 2 点が，ワーキング・セット法と，グローバル LRU 法等，個々のプログラムに割り当てる主記憶量を制御する考えに基づいていない制御方式とが基本的に異なる点である．

ワーキング・セット法は，このように概念的にはすぐれた制御方式であるが，ワーキング・セットをプログラムの局所参照集合の良い推定量とするには，ウィンド・サイズ T をどのように定めればよいかが問題である．プログラムの局所参照集合を含むウィンド・サイズは，個々のプログラムによって，また，プログラム実行中のフェイズによって異なるはずである．それにあわせてウィンド・サイズを決定しようとすると，オペレーティング・システムは，プログラム実行

中に，そのページ参照特性に関する情報を収集し解析を行わねばならない．各プログラムのライフタイム関数の形がわかっている場合には，それぞれのプログラムに割り当てる最適な主記憶量を決定する方法（後述の“knee”基準）が提案されている[15]．プログラムの実行中にライフタイム関数を求め，それに基づいて，各プログラムに割り当てる最適な主記憶量を決めようとする実験的な試みもある[25]．

Graham[16),21)]は，いくつかのプログラムについて，ウィンド・サイズと space time product（割当て主記憶量の時間積分）の関係を求めた結果，ワーキング・セット法におけるウィンド・サイズの許容範囲は広い，すなわち，ウィンド・サイズが space time product を最小にするウィンド・サイズの値からかなりずれても，space time product の増加が少ないことを報告している．このような実験的な裏付けもあって，現実のシステムでは，システム全体に共通のウィンド・サイズを設定しているものが多い．

2章で述べたように，プログラムはその実行中にいくつかのフェイズの間を移動する性質を有している場合が多く，フェイズの移動が生じた際には，それまでとは異なったページが数多く参照される傾向がある．したがって，ワーキング・セットの中には，一時的に移動前のフェイズのページ，移動後のフェイズのページが含まれてしまうために，新しいフェイズにはいってウィンド・サイズに対応する時間が経過するまでは，ワーキング・セットの大きさが大きくなる．移動前のフェイズのページは不用のものが多いはずであり，主記憶の利用効率を上げるためには，それらをできるだけ早くページ・アウトすることが望ましい．この問題を解決するために，Smith[30)]は，フェイズの移動が生じたところで一時的にウィンド・サイズの値を小さくすることを考え，以下のような方法を提案している．ウィンド・サイズとして，T のほかに，今一つ T' を設ける．$T'<T$ とする．ウィンド・サイズ T の外側のページは，通常のワーキング・セット法と同様に常にページ・アウトする．ページ・フォールトが生じたときに，主記憶内の LRU ページの参照が現在より T' 時間過去の時刻以前になされたものとすると，フェイズの移動が生じたと判断して，その LRU ページをページ置換えの対象とする．LRU ページの参照が現在より T' 時間過去の時刻以後になされていれば，新しい1ページを確保してそれをページ・フォールトが生じたページに割り当てる．Smith は，この方法によって，フェイズの移動に伴うワーキング・セット・サイズの一時的な増加の現象を防ぐことができることを実験によって確かめている．

個々のプログラムの局所参照特性を考慮した今一つの興味深い主記憶制御方式は，Chu & Opderbeck[6),28)]が提案した PFF (Page Fault Frequency) 法である．この方式では，ページ・フォールト率を制御パラメータとして，アクティブ・プログラムについてはすべてそのページ・フォールト率がある一定の値以下になるように，それぞれのプログラムに割り当てる主記憶量を制御する．制御パラメータとしてのページ・フォールト率の値を P とする．時刻 t で実行中のプログラムがページ・フォールトを生じたとする．このプログラムの前回のページ・フォールトの時刻を t' とする．$t-t'<1/P$ のときには，ページ・フォールト率が P よりも大きいと判断し，新たに1ページを割り当てる．また，$t-t'\geq 1/P$ のときには，ページ・フォールト率が P よりも小さいと判断し，時刻 t' と t の間で参照されなかったページをページ・アウトする．すなわち，この場合には，$t-t'$ をウィンド・サイズと考えることができる．

PFF 法は，ワーキング・セット法において，各プログラムのページ・フォールト率が一定の値以下になるように，それぞれのプログラムのウィンド・サイズを制御している方式であると考えられる．Chu 等は，いくつかのプログラムについて，その space time product を固定割当ての LRU 法（ローカル LRU 法），PFF 法，ワーキング・セット法について求め，PFF 法は LRU 法に比較してすぐれており，ワーキング・セット法にほぼ等しい効率を有する制御方式であると結論している．また，ワーキング・セット法と同様，PFF 法の制御パラメータの許容範囲が広いことを実験の結果として報告している．

一定の主記憶容量内でページ置換えアルゴリズムとして FIFO を利用した場合，ある種のページ参照列に対しては，主記憶容量が大きい場合の方がページ・フォールト率が大きくなる異常現象 (anomaly) が発生することが知られている[3)]が，Franklin 等[19)]は，PFF 法でも同様の異常現象が存在することを明らかにしている．

5. 多重度の制御

4章で代表的な変動割当て法について述べたが，各方式における制御パラメータの設定，いいかえれば，

多重度の設定，あるいは，システム負荷の設定に関する根拠が必ずしも明確ではない．現実のシステムにおいても，ワーキング・セット法におけるウィンド・サイズ，グローバル LRU 法の制御パラメータであるページ・フォールト率等，経験的に適当な値が設定されている場合が多い．それぞれの制御方式について，制御パラメータの設定の根拠づけを行うことは重要であり，この点に関していくつかの報告がある．特に，Denning 等[15]は，以下のような方法を提案している．

"knee 基準" では，プログラムを実行したときの space time product ができるだけ小さくなるように主記憶の割当てを行う．1回のページ・フォールトの処理の遅れを D とし，プログラムの開始から終了までに K 回のページ・フォールトが生じたとすると，このプログラムの処理に要する平均実時間は，$K(G(x)+D)$ である．ここで，$G(x)$ はライフタイム関数，すなわち，主記憶量が x のときのページ・フォールト間の時間間隔である．すると，単位処理時間当りの space time product は，

$$\frac{x(KG(x)+KD)}{KG(x)}=x+\frac{D}{g(x)}$$

となる．ここで，$g(x)=G(x)/x$ である．

先に述べたように，ライフタイム関数 $G(x)$ は，一般に，割当て主記憶量の増加に伴い，下に凸から上に凸の関数に変化する．$g(x)$ を最大にする点は，$G(x)$ の "knee" に対応する．knee 点に対応する主記憶量を割り当てると，上式の第2項，すなわち，ページングによる space time product が最小になる．space time product の中で第1項 x の影響が小さいとすると，knee 点で動作させれば全体としても space time product が最小に近くなる傾向があると考えられ，このことは実験によっても確かめられている[16),21)]．

knee 基準は，ワーキング・セット法等，個々のプログラムの局所参照特性を把握しようとする主記憶制御方式に対して，主記憶割当て量の決定に一つの根拠を与えるものである．しかしながら，それを実現するには，プログラムのライフタイム関数をその実行時に測定せねばならず，かなりの手間を要する．一方，前述のように，ワーキング・セット法等では，ウィンド・サイズの許容範囲は広い．そこで，Denning 等は，knee 点をプログラム実行時に厳密に求めることはせず，実行がはじまった点であらかじめ決めてあるウィンド・サイズから適当なものを選択するだけで十分であると述べている．

制御パラメータの設定に関する第2の提案は，"$L=S$ 基準" である．これは，システムのライフタイム L を，ページのスワッピング時間 S よりもやや大きめにとろうとするものである．

ページング装置のサービス完了率を b_p とし，CPU のサービス完了率を b_c，その内，プログラムが終了してシステムをでていく割合を a_0，ページ・フォールトの発生率を a_p とする．また，システムが多重度 n の平衡状態で動作中，CPU のスループットを $T_c(n)$，プログラムが終了する割合，すなわち，システムのスループットを $T(n)$，ページング装置のスループットを $T_p(n)$ とすると，

$$T(n)=T_c(n)\cdot\frac{a_0}{b_c}$$

$$T_p(n)=T_c(n)\cdot\frac{a_p}{b_c}$$

が成り立つ．システムのライフタイムを $L(n)$ とすると，

$$L(n)=1/a_p$$

であり，1ページのスワッピングに要する時間を S とすると，

$$S=1/b_p$$

である．また，$T_c(n)\leq b_c$，$T_p(n)\leq b_p$ であるので，結局，

$$T(n)\leq a_0\cdot\min\left\{1,\frac{L(n)}{S}\right\}$$

が成立することになる．$T(n)/a_0$ は CPU の利用率であり，ページング装置を CPU 利用率の隘路としないためには，

$$L(n)\geq S$$

とすることが必要である．

$L=S$ 基準は，先の knee 基準とは異なり，各プログラムへの主記憶の割当て量を定めるための基準を与えるものではなく，全体として CPU の利用率を高めることによりスループットを向上させることを狙ったものである．

Denning 等の提案している今一つの単純な基準は，"50% 基準" とよばれ，ページング装置の利用率を 50% 程度に保つように制御パラメータの値を設定しようとするものである．この方法の根拠は，ページング装置のサービス時間が指数分布に従うとすると，その利用率が 50 パーセントのときが，待ちができるかどうかの分岐点であり，利用率が 50 パーセントを越すとスラッシング現象を生ずることになり，その直前

に最適な負荷があると考えるところにある.

6. むすび

仮想記憶システムの主記憶制御方式について，これまでに提案あるいは実現されている方式を，主記憶割当ての観点から，固定割当て法，変動割当て法に分類し，それぞれの特徴，相互の関連等を述べた.

これら制御方式の検討を行うには，仮想記憶システムのもとで動作するプログラムの動作特性を把握することが重要であり，これまでに数多くの研究の成果が発表されているが，本報告では，この点に関しては，紙面の都合で必要最小限の範囲を述べるに留めた．プログラムの動作特性，および，そのモデルについては，機会があれば，稿を改めて報告したい.

仮想記憶システムの主記憶制御方式に関しては，これまでに数多くの研究がなされている．しかしながら，現実のシステムを眺めてみると，グローバルLRU法を採用しているシステムもあれば，ワーキング・セット法を基本にしているシステムもあり，そのいずれが，スループット，応答性等においてすぐれているか等，基本的な点も明確ではない．ワーキング・セットはすぐれた考え方であるが，プログラムに局所参照の性質がどの位存在しているのか，プログラムのフェイズの移動がどの程度生じ，それがワーキング・セット法にどの程度の影響を及ぼしているのか等，これまでに明確な結果は得られていないように思われる.

このような点を考えてみても，仮想記憶の制御方式に関する研究は，単一プログラムを対象とした場合にはいくつかのはっきりした結論が得られているが，多重プログラミング・システムのもとでの制御方式に関しては，今後さらに，モデルによる解析，実システムの実測データの解析等が必要であり，あわせて，これまでに解かれている範囲を明らかにする必要があるように考えられる.

今後の大きな課題としては，最近急速に普及しているデータベース・システムが，仮想記憶の制御方式にどのような影響を与えるかを明らかにすることがある．特に，プログラムの局所参照の性質等も，データベースのアドレス参照になると全く異なったものになる可能性が大きく，まず，これら基本的なデータの収集，解析が必要である.

最後に，本報告をまとめるために助言をいただいたTong-Haing Fin，吉沢康文，西垣通の各氏に感謝の意を表する.

参考文献

1) Belady, L. A.: A Study of Replacement Algorithms for a Virtual Storage Computer, IBM Sys. J., Vol. 5, No. 2, pp. 78-101 (1966).
2) Belady, L. A. and Kuehner, C. J.: Dynamic Space Sharing in Computer Systems, CACM, Vol. 12, No. 5, pp. 282-288 (1969).
3) Belady, L. A., Nelson, R. A. and Shedler, G. S.: An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Machine, CACM, Vol. 12, No. 6, pp. 349-353 (1969).
4) Belady, L. A. and Tsao, R. F.: Memory Allocation and Program Behavior under Multiprogramming, Proc. Computer Science and Statistics: 7th Annual Symp. Interface, pp. 72-78 (1973).
5) Chamberlin, D. D., Fuller, S. H. and Liu, L. Y.: An Analysis of Page Allocation Strategies for Multiprogramming Systems with Virtual Memory, IBM J. Res. and Devel., Vol. 17, No. 5, pp. 404-412 (1973).
6) Chu, W. W. and Opderbeck, H.: The Page Fault Frequency Replacement Algorithm, Proc. FJCC, pp. 597-609 (1972).
7) Coffman, E. G. and Ryan, T. A.: A Study of Storage Partitioning Using a Mathematical Model of Locality, CACM, Vol. 15, No. 3, pp. 185-190 (1972).
8) Corbato, F. J.: A Paging Experiment with the Multics System, Report MAC-M-384, Project MAC, MIT, Cambridge (July 1968).
9) Denning, P. J.: The Working Set Model for Program Behavior, CACM, Vol. 11, No. 5, pp. 323-333 (1968).
10) Denning, P. J.: Virtual Memory, Computing Surveys, Vol. 2, No. 3, pp. 153-189 (1970).
11) Denning, P. J.: On Modeling Program Behavior, Proc. SJCC, pp. 937-944 (1972).
12) Denning, P. J. and Schwartz, S. C.: Properties of the Working Set Model, CACM, Vol. 15, No. 3, pp. 191-198 (1972).
13) Denning, P. J. and Graham, G. S.: Multiprogrammed Memory Management, Proc. IEEE, Vol. 63, No. 6, pp. 924-939 (1975).
14) Denning, P. J. and Kahn, K. C.: An $L=S$ Criterion for Optimal Multiprogramming, Proc. Int. Symp. on Computer Performance Modeling, Measurement and Evaluation, pp. 219-229 (1976).
15) Denning, P. J., Kahn, K. C., Leroudier, J., Potier, D. and Suri, R.: Optimal Multiprogramming, Acta Informatica, Vol. 7, No. 2,

pp. 197-216 (1976).
16) Denning, P. J.: Optimal Multiprogrammed Memory Management, in Current Trends in Programming Methodology (Chandy, K. M. and Yeh, R. T., Eds.), Prentice-Hall, pp. 298-322 (1978).
17) Denning, P. J.: Working Sets Then and Now, in Operating Systems: Theory and Practices (D. Lanciaux, Ed.), North-Holland, pp. 115-148 (1979).
18) Fin, T. H. and Kameda, H.: An Extension of a Model for Memory Partitioning in Multiprogrammed Virtual Memory Computer Systems, JIP. Vol. 2, No. 2, pp. 89-96 (1978).
19) Franklin, M. A., Graham, G. S. and Gupta, R. K.: Anomalies with Variable Partition Paging Algorithms, CACM Vol. 21, No. 3, pp. 232-236 (1978).
20) Ghanem, M. Z.: Study of Memory Partitioning for Multiprogramming Systems with Virtual Memory, IBM J. Res. and Devel., Vol. 19, No. 5, pp. 451-457 (1975).
21) Graham, G. S.: A Study of Program and Memory Policy Behavior, Ph.D. thesis (1976), Dept. Computer Sciences, Purdue University.
22) Gupta, R. K. and Franklin, M. A.: Working Set and Page Fault Frequency Paging Algorithms: A Performance Comparison, IEEE Trans. Computers, Vol. C-27, No. 8, pp. 706-712 (1978).
23) Leroudier, J. and Potier, D.: Principles of Optimality for Multiprogramming, Proc. Int. Symp. on Computer Performance Modeling, Measurement and Evaluation, pp. 211-218 (1976).
24) 益田, 高橋, 吉沢: ページング・マシンにおけるスワッピング・アルゴリズムの比較とプログラムの動作解析, 情報処理, Vol. 13, No. 2, pp. 81-88 (1972).
25) 西垣, 池田: 仮想メモリ・システムのワーキング・セットの最適化に関する一実験, 投稿中.
26) Oden, P. H. and Shedler, G. S.: A Model of Memory Contention in a Paging Machine, CACM, Vol. 17, No. 8, pp. 761-771 (1972).
27) 大西, 秋田, 堂免, 金子: HITAC 8700/8800 オペレーティング・システム (OS 7), 情報処理, Vol. 14, No. 10, pp. 769-777 (1973).
28) Opderbeck, H. and Chu, W. W.: Performance of the Page Fault Frequency Algorithm in a Multiprogrammed Environment, Proc. IFIP congress 74, pp. 235-241 (1974).
29) Rodriguez-Rosell, J. and Dupuy, J. P.: The Design, Implementation and Evaluation of a Working Set Dispatcher, CACM, Vol. 16, No. 4, pp. 247-253 (1973).
30) Smith, A. J.: A Modified Working Set Paging Algorithm, IEEE Trans. Computers, Vol. C-25, No. 9, pp. 907-914 (1976).
31) Smith, A. J.: Bibliography on Paging and Related Topics, ACM Operating Systems Review, Vol. 12, No. 4, pp. 39-56 (1978).
32) Spirn, J. P.: A Model for Dynamic Allocation in a Paging Machine, Proc. 8th Princeton Conf., pp. 331-334 (1974).

(昭和55年1月29日受付)