仮想記憶システムのモデル化とシステム性能評価

大須賀節雄（東京大学　宇宙航空研究所）

1．序言

計算機システムの性能は扱われる情報（プログラムおよびデータ）の性質に強く影響される。したがってシステム・アーキテクチュアやオペレーティング・システムの設計は本来これらの情報の動的な性質を究明し、理解した上で行なわれるべきものである。しかしこれまで、このような情報の動特性分析が必ずしも十分でなく，システム設計にも推測に頼る部分が残されている。

この結果，今後予想される利用パターンの変化に伴って、現在のアーキテクチュアおよびOSのもとでは著しい性能の低下が生じ得る。この意味で現在の仮想メモリ方式は設計思想自体が必ずしも完全なものではない。本論文では、まずマルチ・プログラムのもとでの計算機システムの平衡条件を求め，その条件に関連する諸要因を明らかにする。次いで各プログラム動作を表わすモデルを作成し、これより特性指標を理論的に求める。情報のもつ本質的な性質から、この特性指標の値の一般的傾向が論じられる。このような解析を通して、以下では，(1)システムの特性はプログラムとデータのそれぞれの特性に依存して定まること，(2)この両情報の動的性質は非常に異なるものであること，(3)従来はプログラム特性に重点が置かれてシステム設計が行なわれてきたが，実際にはデータ特性が大きな影響を及ぼすため、今後、データベースなどデータ利用技術の発達によりシステム性能の大中な低下があり得ること，したがって (4)システム解析はこの両方の情報の性質を反映できるモデルにより行なわれるべきこと，などが導かれる。特に仮想メモリ方式については，これがユーザに大きな論理空間を与える点に意義があり、この効用は特に大型のデータ構造を扱う際に大きいことが期待される。しかし現実にはデータの増大に伴って、仮想メモリ方式は著しい性能低下をきたす。これは仮想メモリ方式の設計思想における矛盾ともいえるものである。これらの検討を通して、(5)システム性能維持の条件と可能な方式を採る。

2．システムの平衡条件と情報の特性指標

以下の議論では複数のユーザ・プロセスを擁する仮想メモリ・システムを想定する。仮想メモリ方式は二つの仮定のもとに成立っている。すなわち，

(1)プログラムの局所性：ページ参照の確率はプログラム全体にわたって平均的でなく、時間的に利用される部分が異なり，これがプログラムの実行に伴って移動する。

(2)利用ページの予測可能性：ある時点で必要とするページ集合を得る方法が存在する。

である。(1)についてはこれまでの多くの研究[2][4][7]からほぼ事実に近いものと考えられ、この前提のもとで(2)の方式が検討されてきた。しかし、(1)に関するこれまでの検討はプログラム・ページに偏重しており、データ・ページに関する考慮が不十分である。データ・ページの存在がシステム特性に影響することはすでに

多くの例から明らかであり、今後データ・ベースのような大型データが扱われるようになった場合、現用の仮想メモリ方式そのものに疑念が生ずる。以下ではモデルによる分析を行なってこれを示す。

システム内でアクティブ・ユーザの処理プロセス（以下プロセス）の集合を $P = \{P_1, P_2, \cdots, P_n\}$ とする。このうち、各プロセスごとに全ページのうち一部が主メモリに準備されており、いつでも実行可能な状態にあるものを"ready"状態にあると言う。主メモリには参照確率の高いページの集合（以下 Working Set または単にWSと略称）を準備するのが望ましい。この状態にあるプロセスの集合を $P_r = \{P_{i_1}, \cdots, P_{i_k}\}$ とする。

このようなプロセスの実行中，WS以外のページを参照した時，ページ割込みが生じ、このプロセスの処理は中断し、ready状態にある他のプロセスに制御が移されると同時に、要求したページの読込み要求が出される。この読込み完了までに、時間Tを必要とする。あるプロセス実行開始後，ページ割込みまで，すなわち割込み間時間を1パスと呼び，この時間長を"パス長"と言うことにする。時間Tが経過すると要求ページが主メモリ内に準備され，当該プロセスは再びreadyとなる。

この状況は図1(a)のように複数のI/O queueと1個のCPU queueが直列に配置され、かつ有限個の呼が循環する待合せ系のモデルにより表わされる。これはさらにI/O系の入出力は1個にしぼられているから、図1(a)の点線部分を一まとめにして表わすと、図1(b)のように単純化される。このモデルではI/Oと名付けられたサービス窓口の処理時間分布は図1(a)でA点において観察された呼間の時間によって測定される特定の分布をもっている。この分布形状は各入出力装置の性質や、装置の数により変化し、厳密に定めることは困難である。しかしこの分布が有限の平均値 $1/\mu$ を持つこと，この平均値 μ^{-1} は、各I/O装置の一件あたりの処理時間 μ_i^{-1} とし、このI/O装置の利用頻度を w_i ($\Sigma w_i = 1$) とした時，$\mu^{-1} \geq \Sigma w_i \mu_i^{-1}$ なることは明らかである。

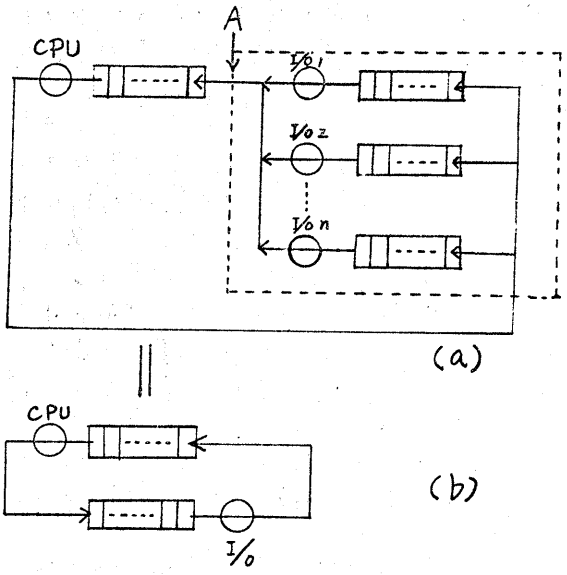


図1. システム待合せモデル

ここで粗い近似として、CPUおよびI/Oの処理時間分布をそれぞれパラメータ入および μ の指数分布としてみよう。この時、CPU queueが0となる確率すなわちCPUがidleとなる確率は $(1-\rho)/(1-\rho^{m+1})$ である。ここに $\rho = \mu/\lambda$ であり、mはこの循環系の中の総呼数であり，各プロセス P_i のWSのページ数を W_i，主メモリの容量（ページ数）をMとすると，$\sum_{i=1}^{m} |W_i| \leq M$ なる最大の整数mである。従ってCPUのThrough put η は

$$\eta = \begin{cases} \dfrac{\rho(1-\rho^m)}{1-\rho^{m+1}} & \rho < 1 \\ \dfrac{m}{m+1} & \rho = 1 \\ \dfrac{\rho(\rho^m-1)}{\rho^{m+1}-1} & \rho > 1 \end{cases}$$

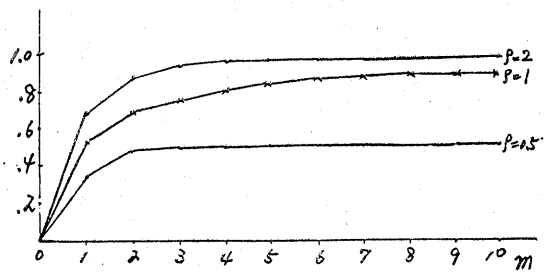


図2. プロセス数ースループット

で与えられる。図2はこの傾向を示したものである。$m \to \infty$ の極限値において

$$\eta_\infty = \begin{cases} \rho & \rho < 1 \\ 1 & \rho \geq 1 \end{cases}$$

最初に述べたように、これは極めて粗い近似である。しかし、この系の特性としてまず挙げられるのは分布形状によるηの変化が少ない点である。すなわち $m=\infty$ および $m=1$ における η の値は分布形状によらないし、mについて単調増加の性質も保存される。したがって図2において分布形状に影響されるのは $m \geq 2$ における漸近曲線の多少の変化のみである。以下で指摘するのはこれよりはるかに大きな影響を与える要因についてであり、したがってわれわれは上述のモデルによる近似で一般的な傾向を調べることにする。このモデルの解析結果から、次の結論が導かれる。

(1) $\eta < \rho$ であること，したがって η を増すには $\rho \gg 1$ とすることが最も重要であること。

(2) $\rho \gg 1$ とするには.

(2-1) $1/\mu$ を小にすること、これには $\Sigma W_i \mu_i^{-1}$ を小にすること。

(2-2) $1/\lambda$ を大にすること、これにはパス長を大にすること。

(3) $\rho \gg 1$ でも m が小の場合、η は小である。したがって十分大なるmを与えるだけの主メモリ容量Mを与えること。しかし、ある程度以上（$m \geq 5 \sim 6$）のMの増大は無意味である。$\rho < 1$ では m の増大は殆んど効果がないこと。

このうち (2-1) $\Sigma W_i \mu_i^{-1}$ を小にするには最も頻繁に使うI/O装置としては処理速度の大なるものを用いるということを意味するが，実際に頻繁に生ずるmapping faultに対してswap用の早いドラムを用い，一般I/Oには大容量ディスク等を用いるといった使い分けはすでに行なわれている。しかし根本的に μ^{-1} を小にするには、各I/O装置の物理的な処理速度 μ_i^{-1} を小にせねばならず、以下で示すようなシステム側の要求にI/O装置の改善で応えようとしたら、全く新しい、アクセスの早い（現在のものの2～3桁早い）装置を必要とすることになろう。現状ではこれは不可能である。したがって、システムの効率を改善するには (2-2) で $1/\lambda$ を出来る限り大とする努力が唯一のものとなる。このことを念頭において、以下ではプログラム動作をモデル化し、分析してみる。

3. システムのモデル

現在実行に入ったプロセスをP_aとし，このプロセスについて前節で定義した$1/\lambda$をt_aと表わす。$t_a(W_a)$はプログラムとデータの性質を表わす一つの特性指標関数としての性質をもっている。本節ではこれをシステムのモデル化により分析する。

3．1　マルコフ過程によるモデル化

プログラムの進行により主メモリ内の語が順次アクセスされるが、このアクセスの過程をマルコフ過程とみなし、モデル化を試みる。

nページから成るプロセスP_aでiからjへの遷移確率をp_{ij}と表わす。p_{ij}は$\sum_{j=1}^{n} p_{ij} = 1,\ (i=1, \cdots, n)$である。$p_{ii}$は同一ページ内の語に二回続けてアクセスする確率である。WSをW_aとする。次の遷移マトリックスT_aを定義する。$W_a = \{P_1, P_2, \cdots, P_k\}$とする。

N回の遷移後，状態iにある確率を$\pi_i(N)$，これをi要素とするベクトルを$\Pi(N) = (\pi_1(N), \pi_2(N), \cdots, \pi_n(N))$とする。同様に初期状態ベクトルを$\Pi(0)$で定義する。すると(3.2)式の通りである。

$$T_a = \begin{bmatrix} p_{11} & \cdots & p_{1k} & p_{1k+1} & \cdots & p_{1n} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{k1} & \cdots & p_{kk} & p_{kk+1} & \cdots & p_{kn} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{n1} & \cdots & p_{nk} & p_{nk+1} & \cdots & p_{nn} \end{bmatrix}, \qquad (3.1)$$

$$\Pi(N) = \Pi(0) \cdot T_a^N \qquad (3.2)$$

実行開始後，N回の遷移がすべてW_a内のページで行なわれる確率を求めるには、新しい遷移マトリックスT_a^*を定義する((3.3)式)。これに対応して、N回遷移後の状態確率ベクトルを$\Pi^*(N)$とすると(3.4)式が得られる。

$$T_a^* = \left[\begin{array}{ccc|ccc} p_{11} & \cdots & p_{1k} & p_{1k+1} & \cdots & p_{1n} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{k1} & \cdots & p_{kk} & p_{kk+1} & \cdots & p_{kn} \\ \hline \multicolumn{3}{c|}{0} & \multicolumn{3}{c}{I} \end{array}\right] \qquad (3.3)$$

0；全要素0のマトリックス
I；単位マトリックス

$$\Pi^*(N) = \Pi(0) \cdot T_a^{*N} \qquad (3.4)$$

さらに、N回の遷移後，W_a内のどれかのページにある確率を$\hat{P}_a(N)$とすると(3.5)式のようになる。ここでKは1～k要素が1，他は0なる縦ベクトルである。これより，1ステップの実行時間tを用いると$N t = \alpha$であるから(3.6)式が得られる。

$$\hat{P}_a(N) = \Pi^*(N) \cdot K = \Pi(0) \cdot T_a^{*N} \cdot K \qquad (3.5)$$

$$P_a(\alpha/W_a) = \hat{P}_a([\alpha/t]), \quad [x]；x\text{より小なる最大の整数} \qquad (3.6)$$

さらに、丁度N回目迄はW_a内で遷移を続け、$N+1$回目でW_aの外に出る確率は(3.7)式の通りである。

$$\bar{P}_a(N) = \hat{P}_a(N) - \hat{P}_a(N+1) \qquad (3.7)$$

以上の結果，$\bar{t}_a$が

$$\bar{t}_a = \bar{N} \cdot t = t \sum_{N=1}^{\infty} N \bar{P}_a(N) = t \sum_{N=1}^{\infty} \hat{P}_a(N) = t \sum_{N=1}^{\infty} \Pi^*(N) \cdot K = t \sum_{N=1}^{\infty} \Pi(0) \cdot \hat{T}_a^{*N} \cdot K = t\Pi(0) \cdot \hat{T}_a^* \cdot (I - \hat{T}_a^*)^{-1} \cdot K' \qquad \cdots (3.8)$$

と得られる。ここでIは単位マトリックス，K'はすべて1の要素のみから成るk次の列ベクトル，$\hat{T}_a^*$はT_a^*のうちWSに入る要素のみから成る小行列である。

3．2　プログラムとデータの特性

前節で定義したT_aは扱われている情報の動的な性質をよく反映している。これをプログラムの場合とデータの場合とに分けて調べてみよう。

3．2．1　プログラム特性値

プログラム・ページの場合，p_{ii}が一般に大なることはその逐次アクセスの性質から明らかである。p_{ij}を直接的に示すデータが十分でないので、以下では既存のデータを手がかりにp_{ii}の値を推定してみる。

一つは、少し古いが、Gibson Mix のもとになっている命令語の使用頻度からの推定である。このデータは全命令のうち無條件ジャンプ命令が 0.175，條件付きジャンプ命令が 0.065 の割合を占めていることを示している。條件付きジャンプのうち、実際にジャンプ條件が成立するのがこの半分とみると、ジャンプの生ずるのが全体の20%であり、残りが逐次的に実行される命令である。各ページで最後の命令はジャンプ命令以外でもページ遷移が生ずるが，頻度としては少ないので，ジャンプによる遷移に含めて扱う。

ジャンプ命令でも同一ページ内遷移の場合が多いのでこの割合を f とすると，$p_{ii}=0.8+0.2f$，$(0\le f\le 1)$ である。

もう一つの参考例としては現実のシステムにおいて、総ページ100前後のプログラムでWSの大きさとして20～40ページが用いられ，この時の1パス長として数百～数千ステップが実行されるという事実がある。各ページごとに p_{ii} は一定，$(=\alpha)$ であり，p_{ij} $(i\neq j)$ は均等な確率 $(\beta=(1-\alpha)/n-1)$ として式(3.8)より π_a を求め、この結果を具体例に適用してみると，

$$\pi_a=\frac{h}{1-\alpha}\left(\alpha+\frac{c-1}{n-c}\right),\quad c=|W_a| \qquad (3.9)$$

となる。これから、π_a が $\alpha=p_{ii}$ に大きく影響されることが判る。これはアクセスにおける局所依存性の重要性を示す。

典型的なプログラムの場合、遷移確率は現在位置から遠くなる程小さくなる。これを図3のように α，β，γ の3段階の矩形分布で近似する。ここで $\alpha=p_{ii}$，$(i=1,\cdots,n)$，$\beta=(1-\alpha)/(n-1)$，$\gamma=\beta/\delta$，$(\delta$：整数$)$ とし、WS内のページへの遷移を β，WS外のページへの遷移を γ とする。WS外に一度遷移すると再び戻ってくることはない。したがってWSは α，β を含むものとし，これ以外の状態（ページ）をいくつかずつまとめて1つの状態におきかえても結果に影響しない。たとえば δ 個ずつをまとめて一つの状態にすると，p_{ij}，$(i\neq j)$ はすべて同じ価となり、式(3.9)が適用できる。総ページ数が n，$|W_a|=c$ であるから、この結果は見かけ上のページ数 $n'=c+(n-c)/\delta$ となる。そこで $n=50$　$c=25$　$\delta=5$　$\pi_a=100h$ とすると、$n'=30$，$\pi_a=h(\alpha+4.8)/(1-\alpha)=100h$，から $\alpha=0.943$ となり、ネ1の推定において、$f=0.715$ すなわちジャンプ命令のうち 71.5% が同一ページ内にジャンプすることに相当する。

3.2.2. データ特性値　　一方，データについては配列に制約のある線形ファイル，ポインタ結合をもちいるリスト構造，ランダムアドレス配置のハッシュ・コード・データなど各種の形式があり、構造にプログラムほどの制約がない。特定の言語内で許されるデータ形式には一定の制約があるが，データの配列に関する制約とアクセス順序とは理論上無関係である。この意味でデータはアクセス順序という観点からは構造性が乏しいのが特徴である。したがって以下ではデータをランダムな配置として解析し、しかる後 構造化について考察してみる。

n ページのファイルを考えよう。遷移確率等に関する諸概念はプログラムの場合と同様とする。データの場合の最大の特徴は遷移確率 p_{ij} が上述のランダム性から状態 i には依らず一定値 $1/n$ となることである。WSの中でデータ参

$$\pi_a^{*N}=\frac{c^{N-1}}{n^N}\left[\begin{array}{c|c}1\cdots 1 & 1\cdots 1\\ \hline 1\cdots 1 & 1\cdots 1\\ 0 & I\end{array}\right] \qquad (3.10)$$

$$\hat{\pi}_a^{*N}\cdot K'=\left(\frac{c}{n}\right)^N\cdot K' \qquad (3.11)$$

$$\hat{p}_a(N)=\Pi(0)\cdot\hat{\pi}^{*N}\cdot K'=\left(\frac{c}{n}\right)^N\cdot\Pi(0)\cdot K'=\left(\frac{c}{n}\right)^N \qquad (3.12)$$

照が行なわれる確率は、したがって、$|W_a|=c$ のみに比例する。すべての i, j について $p_{ij}=1/n$ とすると (3.10)～(3.14) の式が得られる。このような例はデータ群中に特定のデータを無作為に探索する場合に生ずる。一例として、プログラムの場合と同じ条件すなわち、$n=50$ $c=25$ として比較すると $\bar{t}_a=c$ となり、右の値で 10^{-2}, p_{ij} で $1/0.8n = 1/40$ の程度の相違を示す。図4に $\bar{t}_a-|W_a|$ 曲線をプログラムについて式(3.9), データについて式(3.13)を用いて表わす。

$$\bar{t}_a = c\sum_{N=1}^{\infty} P_a(N) = c\sum_{N=1}^{\infty}\left(\frac{c}{n}\right)^N = c\,\frac{c}{n-c} \qquad (3.13)$$

$$k_a = \bar{t}_a/c = c/(n-c) \qquad (3.14)$$

以上はデータにおける一例であり、実際のデータは様々の構造を持ち、その構造によって p_{ij} の値も異なっている。ただし、データに与えられる構造は論理的な構造であり、パス長に必ずしも寄与するものではない。ここで問題となるのはアクセス順序と記憶配列との一致の程度である。プログラムの場合、アクセスの線形性と記憶配置の線形性とが良い一致をしているのにたいし、データの場合にはこれが保証されない。それはデータの配列方法は一通りであるのに、データを利用する側のデータの見方すなわちアクセス順序は多様であり、常にこの一致を図ることは困難なためである。これを前提として、以下でいくつかのデータ形式について特性を調べてみる。

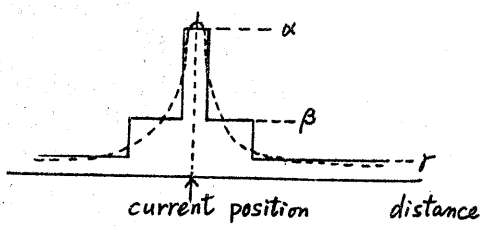


Fig. 3 Approximated distribution of transition probability of program

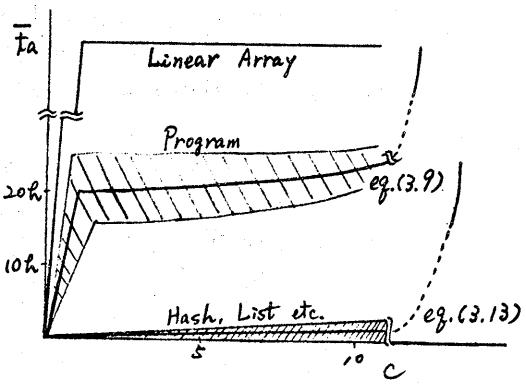


Fig. 4 Mean path length

〔1〕. 線形配列； 線形配列ではデータ要素のアドレスが論理的データ順序 i と線形関係 $a+bi$ で対応づけられる。一般にはデータ要素はメモリ内に密に詰められる。アクセスが i の順序で行なわれる場合には p_{ii} の値がプログラム以上に大となる。

多次元のアレイ、たとえば (i,j) 要素が $a+(g(i-1)+j-1)b$ の形で与えられる二次元アレイでは、b は1と定めても i に関してデータ要素を参照すると、g 間隔の離散的参照となり、p_{ii} は著しく低い。特に $g>c$ では p_{ii} はほとんど0にまで低下する。このことは多次元アレイ方式は本質的にシステム効率を低下させる作用を含むことを意味する。

〔2〕. ハッシング； ハッシュ関数も関数を用いてアドレスを算出するという点では〔1〕と同じタイプのアドレス法といえるが、この性質は線型配列と正反対で ラ

ンダム化を目的としており、p_{ij} が均一の典型的な場合である。主メモリ以外の記憶を用いる場合，ハッシング法はマルチ・ユーザのもとでシステム全体としての効率を著しく低下させることになる。*

[3]．ポインタ結合によるデータ構造；　ポインタによりデータ要素がリンクされるリスト形データ構造は論理的なデータ要素間の関係を物理的な記憶アドレスの関係から分離するものとして用いられている。したがってこの型のデータ構造も基本的には前述のランダム型に近い。ただしデータのアクセス範囲が限定されることは，関係の深いデータを近い位置におくという操作が限定的ではあるが可能であり，t_a を多少増加し得る。これにはクラスタリングなどの手法が有効である。

3．3　特性値による情報の型の分類

以上の分析結果は次のようにまとめられる。

(1)プログラムは t_a がページの全域にわたり高いレベルにある。以後これをP型特性と呼ぶ。

(2)データはデータ形式により分類され，

(i)応用プログラムの直接管理化にある単一目的の小量データは t_a が大である。

(ii)探索が行なわれる多目的データや、ハッシュ・コード型およびポインタ型のデータ構造は、データ量が大なる時，実効的な t_a の値がプログラムに比し著しく小さい。以下これをD型特性と呼ぶ。

図4にこれら各型の例が示されている。各型の分布中は同一ページへの関連データの集積度により t_a の値が変わることを示す。

実際上，今後問題となるのはデータベースなど，上記(2)の(ii)の型のデータである。したがって以下ではP型とD型の情報を扱う。

4．プログラムの動特性

4．1　プログラム・シーケンス・モデル

実際の各プロセスはプログラムとデータの混合である。従来は全般的にデータ比率が低く，全データ・ページを主メモリに読込むことができた。この時はデータ部分に関して $c/n=1$ とすることができ，データの性質にかかわらず，(3.13)，(3.14)式により，$t_a=\infty$ となるから、データ特性は無視できる。数値計算やコンパイリングなど，変換を主とした従来の計算機利用の主流がこのP型であり，この結果，$|w_a|$ の値を一定の範囲内に納めることができ，したがって安定な動作が維持された。

しかし大容量記憶装置が開発され，データベース技術が普及すると共に，D型情報が増えてきている。この傾向は2節の β を減少し、システムのバランスを崩すのでその対策が必要であるが，このためにはこれまでの単純なモデルでなく，

---

* ただしハッシングは内容によるアクセスであり，これと同じ結果を得る方法，たとえば線形ファイル探索と比較することが必要である。後者は局所依存性があり，t_a が大となるが、目的とするデータに達するまでの探索がハッシングではハッシュ関数の演算とユリジョンの処理のみで済むのであるから、線形探索も t_a が見かけ上増大しているのみである。したがってハッシングの際の t_a の低さは内容によるアクセスという問題の性質から来るものである。

プログラムとデータの両方の遷移を含む、より現実に近いモデルを作る必要がある。

命令の実行は(1)命令の Fetch とその解釈，(2)オペランドの Fetch とその実行 という順序で行なわれ、プログラムの遷移とデータの遷移がそれぞれこの(1)と(2)に対応して生ずる。すなわち 3.2 節のプログラムとデータの遷移が同一のプログラムの中で結合されており，この両方の関係で一つのプログラムの特性が定まるのである。

マルコフ過程において状態をプログラム・ページ i とデータ・ページ ξ の複合で (i,ξ) と表わし、$P_{(i,\xi)(j,\eta)}$ により，状態 (i,ξ) から (j,η) への遷移確率を表わす。プログラム・ページの遷移とデータ・ページの遷移間に関連がある場合には $P_{(i,\xi)(j,\eta)}$ を個別に定義する他ないが、この間に強い関連がなければこれをそれぞれの遷移確率の積で近似できる。以下ではこの近似を用いる。すなわちプログラム・ページのみの場合の遷移確率を p_{ij}，データ・ページのみのものを $g_{\xi\eta}$ とすると

$$P_{(i,\xi)(j,\eta)} = p_{ij} \cdot g_{\xi\eta} \tag{3.15}$$

と表わす。遷移マトリックスを T_a で表わし，要素の順序は各 i の中で ξ の順序を付ける辞書的順序とする。

一方、N回遷移後の状態確率を $\pi_{(i,\xi)}(N)$ とし、これを要素とするベクトルを $\Pi(N)$ とする。これにたいして，(3.2)式が成立する。

p_{ij} は3.2.1節で用いた特性のものを用いる。$g_{\xi\eta}$ は3.2.2節のものを基本とするが，より実際の状況に近いものとするため，データ参照の対象をファイルのみでなく，ユーザ作業領域やシステムのコモン領域など，主メモリ内に存在する確率の高いものと，ファイル領域に分ける。前者に属するページをCDP，後者に属するページをFDPと呼ぶ。この時CDP内部での遷移は一律に確率 a，CDPからFDPへは b，FDPからCDPへは c，FDP内遷移は d とすると，(3.16)式のようになる。b,d 等は(3.17)式である。

$$P_{(i,\xi)(j,\eta)} = \begin{cases} \alpha a, & i=j,\ \xi\in CDP,\ \eta\in CDP \\ \alpha b, & i=j,\ \xi\in CDP,\ \eta\in FDP \\ \alpha c, & i=j,\ \xi\in FDP,\ \eta\in CDP \\ \alpha d, & i=j,\ \xi\in FDP,\ \eta\in FDP \\ \beta a, & i\neq j,\ \xi\in CDP,\ \eta\in CDP \\ \beta b, & i\neq j,\ \xi\in CDP,\ \eta\in FDP \\ \beta c, & i\neq j,\ \xi\in FDP,\ \eta\in CDP \\ \beta d, & i\neq j,\ \xi\in FDP,\ \eta\in FDP \end{cases} \tag{3.16}$$

$$b=(1-ka)/l, \qquad d=(1-kc)/l \tag{3.17}$$

k ; |CDP| = 主メモリ内CDPページ数

l ; |FDP| = ファイル内データ・ページ数

全データのうち、FDPの一部とCDPがWSとなる。WS外のページ(状態)は吸収状態であり，(3.8)式より明らかなようにシステムの動特性に関与しない。したがって状態 (i,ξ) のうち、i,ξ 共にWSに入るもの以外を除去したマトリックス $P^*_{(i,\xi)(j,\eta)}$ を用いれば十分である。P^* はかなり大型のマトリックスとなるが，上記のように要素を定めた場合には解析解が得られる。結果のみ記すと，

$$\bar{h}_a = \Pi(0)P^*(I-P^*)^{-1}K\cdot h = \Pi(0)[(\ ),\cdots,(\overbrace{S_c,S_c,\cdots,S_c}^{k},\overbrace{S_F,S_F,\cdots,S_F}^{l}),\cdots(\ )]^T\cdot h \tag{3.18}$$

$$\underset{1,\ \cdots\ ,\ i\ \cdots\cdots\ c}{}$$

$$S_c = (k\lambda + l\mu)/\Delta_1 + C(k\bar{\lambda} + l\bar{\mu})/\Delta_1\cdot\Delta_2 \tag{3.19}$$

$$S_F = (k\nu + l\sigma)/\Delta_1 + C(k\bar{\nu} + l\bar{\sigma})/\Delta_1\cdot\Delta_2 \tag{3.20}$$

$$\begin{cases} \lambda = (\alpha-\beta)a - l(\alpha-\beta)^2(ad-bc) \\ \mu = (\alpha-\beta)b \\ \nu = (\alpha-\beta)c \\ \sigma = (\alpha-\beta)d - k(\alpha-\beta)^2(ad-bc) \end{cases} \quad (3.21)$$

$$\begin{cases} \bar{\lambda} = \beta[a - l(ad-bc)\{2\alpha+(C-2)\beta - l(\alpha-\beta)(\alpha+\overline{C-1}\,\beta)d\}] \\ \bar{\mu} = \beta b[1 - kl(ad-bc)(\alpha-\beta)(\alpha+\overline{C-1}\,\beta)] \\ \bar{\nu} = \beta c[1 - kl(ad-bc)(\alpha-\beta)(\alpha+\overline{C-1}\,\beta)] \\ \bar{\sigma} = \beta[d - k(ad-bc)\{2\alpha+(C-2)\beta - l(\alpha-\beta)(\alpha+\overline{C-1}\,\beta)a\}] \end{cases} \quad (3.22)$$

$$\Delta_1 = 1-(\alpha-\beta)(ka-ld)+kl(\alpha-\beta)^2(ad-bc) \quad (3.23)$$

$$\Delta_2 = 1-(\alpha+\overline{C-1}\,\beta)(ka-ld)+kl(\alpha+\overline{C-1}\,\beta)^2(ad-bc) \quad (3.24)$$

と得られる。すなわち，初期条件として，CDP内のページから開始した場合および FDP内のページから開始した場合の平均パス長がそれぞれ S_C および S_F で与えられる。

4 2 数値例

(3.18)～(3.24)式を用い，仮想的なシステムの特性を調べてみよう。なお，システムの特性はWSの大きさが一定でもどのページをWSに含めるかによう異なってくる。以下ではプログラム・ページを P_i，データ・ページを D_j と表わし，まず少数データの場合についてWSの構成による特性の相違を調べ，ついで大量のデータの場合を検討する。

(1) 少量データの場合　　次のようなプロセスを仮定する。

データ・ページ数　$m=7$,　$a=b=c=d=1/8$

プログラム・ページ;　$\alpha=0.95$,　$\beta=0.015$　　$\gamma=0.001$

なおコモン・ページは1とし，かつこれがプログラム・ページ P_1 と同一とする。これは非メモリ参照命令を，当該命令と同一ページへの参照によって表わすことに相当する。またプログラムの局所性から，WS以外の個々のページへの遷移確率はあまり重要でない。したがって総プログラム・ページ数もあまり意味のない数値であるが，上の例では　$n=23$　に相当する。

この時，$\alpha'=\alpha-\beta$,　$\bar{\alpha}=\alpha+(C-1)\beta=\alpha'+C\beta$　とすると

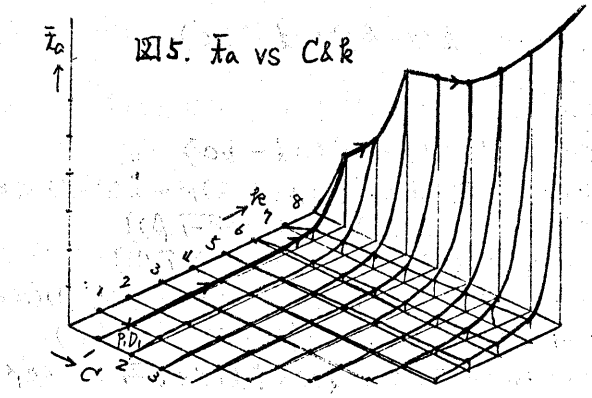
$$\bar{x}_a = \frac{(k+l)\bar{\alpha}}{(k+l)-(k-l)\bar{\alpha}} = \frac{m\{\alpha'+C\beta\}}{m-(k-l)\{\alpha'+C\beta\}} = \frac{\alpha'+C\beta}{1-\frac{k-l}{m}\{\alpha'+C\beta\}} = \frac{\alpha'+C\beta}{1-\frac{2k-m}{m}(\alpha'+C\beta)}$$

であり，k/m, C の値に応じて $\bar{x}_a$ が求まる。これを k, C を独立なパラメータとして図示したのが図5である。この結果から明らかなように，WS中のデータ・ページ少ない時は $\bar{x}_a$ は極端に低く，前述のデータ特性が直接的に反映されている。この傾向は全データ・ページがWSに含まれるまで続きその後にはじめてプログラム型特性が現われる。

このプロセスを，最初のプログラム・ページおよびデータ・ページのみ各1ページをWSとして与えてランさせた場合を想定しよう。図6で P_1 D_1 と印したX印が出発点である。このプロセスは次にデータ・ページもしくはプログラム・ページ参照の mapping fault を生じるが次にどのページを参照するかは参照確率に依存する。したがって局所性の乏しいデータ・ページ参照の場合が多い。そこでページ D_2 が参照されたものとする。この場合，Swap in が行なわれると図6の D_2 に移る。同様にして最も生ずる確率の高い Swap-in の Sequence が図中太線で示した P_1 D_1 D_2 D_3 ⋯ D_8 P_2

P_3 … である。もちろん確率的にはこれより低いが　これ以外の sequence, たとえば $P_1 D_1 D_2 D_3 P_2 D_4 D_5 P_3 D_6 D_7 D_8 P_4$ … のようなものも可能である。

このようにパス長 l_a はプログラム・ページ数 C と，データ・ページ数 k とがそれぞれ独立に異なった影響を与えるが，WS の大きさと l_a との関係となると，$l_a \sim C+k$ の関係が必要になる。上述の二つの例の場合を含め、異なった sequence にたいしてこの関係を図示したのが図6である。



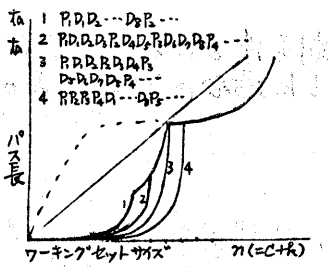


図6. システム特性の一例

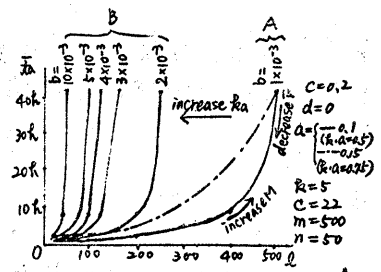


Fig.7 Mean path length vs. l and effect of increasing k.

以上は一つの計算例ではあるが，参照確率に関するプログラムおよびデータの特性から、かなり一般的なシステム特性を表わしていることは明らかである。

(2) データのうちの相当数のページが WS に入らない限り プログラムの局所性の利点が活きてこないことは D型プロセスが増加した場合を考えると極めて重要な事実である。これを調べるために $m=500$ とした例を図7に示す。

図7Aはプログラム・ページを一定にしてデータ・ページの比率を増した時の傾向を示す。この傾向はデータ量にかかわらない。

ここでシステム平衡条件からシステム設計の主要なパラメータは M および $\bar{T}$ である。図7より，$\bar{T}$ が小になると平衡条件を満たすために必要な |Wa| は減少するが，この効果は $\bar{T}$ の絶対値が著しく小にならない限り小さく，現在用いられているファイル装置へのアクセス時間程度では，$\bar{T}$ の減少による効果は小さい。同様にして，M の増大が l_a を増加する効果も，主メモリ内データ・ページの絶対値が少ない間は小さい。D型情報の特性はこのように現在のアーキテクチュアのもとではシステムの平衡条件を満たすことを困難にする。残された方法は l_a の減少を防ぐことのみであり，これには新しいアーキテクチュアが必要となる。これについては紙数の都合で省略する。

5. 結論

本論文はまずシステムの効率を保つための条件と，これに関与する要因を求めた。次いでシステム動作を表わすモデルを作り、分析した結果，データ量の多いプロセスではシステム性能が大中に低下することが示された。これらの結果はシステム設計の基本に関わるものであるが，本論文自体はプログラムやデータの特性に関する実測データが不十分のため、システムの一般的な傾向を示すにとどまっている。

参考文献；　(1) E.G.Coffman & L.C.Varian: CACM, Vol.11, No.7, pp.471-474(1968). (2) P.J.Denning & S.C.Schwartz: CACM, Vol.15, No.3, pp.191-198(1972). (3) P.J.Denning & G.S. Graham: Proc. of the IEEE, Vol.63, No.6, pp.924-939(1975). (4) 益田，高橋：情報処理,Vol.13, No.11, pp.765-770(1972). (5)益田,塩田: 情報処理,Vol.16, No.2, pp.1055-1063(1975). (6) J.H.Saltzer: CACM, Vol.17, No.4, pp.181-186(1974). (7) J.R.Spirn & P.J.Denning: AFIPS Conf. Proc.FJCC, Vol47, pp.611-621(1972). (8) W.C.Wesley & H.Opderbeck: Proc. FJCC, pp.597-609(1972).