数理解析研究所講究録
第695巻 1989年 125-134

多重記憶階層のもとでのデータの変更を考慮した最適ページング

九大工学部　掛下哲郎　(Tetsuro Kakeshita)
九大工学部　上林彌彦　(Yahiko Kambayashi)

mレベルからなる一般の記憶階層のもとで、データの変更を許すページ参照列について、レベル間の最適ページ配置アルゴリズムと各レベルが複数のデバイスから構成されている場合のレベル内の最適ページ配置アルゴリズムを求めた。

1. まえがき

mレベル記憶階層のもとでの最適ページングアルゴリズムを提案する。ページングのコスト関数としては、平均主記憶(最上位レベル)サイズを同一とした条件のもとでの下位レベルへのアクセス回数を考える。このコスト関数は、通常の固定サイズバッファ[COFF73]を仮定するものとは異なっているが、計算機システムにおいては種々のデータが主記憶上に置かれるので、それらの間の記憶領域割り当ては動的に行うべきである。従って、本稿の仮定の方がより実際的である。従来のOSページングではデータの変更を考慮してはいなかったが、本稿の方式は読み出し·書き込み列をページ参照列とみなすことによって、対応する最適なOSページングアルゴリズム[PRIE76]と比較したとき、最良の場合には1/2のコストで処理を行うことができる。さらに、各レベルが性能の異なる複数のデバイスによって構成されている場合のレベル内最適ページ配置問題について考察する。ページ配置問題には、(1)特定のデバイスへの負荷を最大にする問題と、(2)できるだけ多くのデバイスを並列動作させる問題の2つが含まれる。

2節ではmレベル記憶階層の定義と仮定を行った後、レベル間の最適ページ配置問題について議論する。その後、3節ではレベル内での最適ページ配置問題に関するアルゴリズムを提案する。

2. mレベル記憶階層における最適ページ配置

本稿の記憶階層はmレベルからなる。各レベル$i(1\leqq i\leqq m)$はk_i個の物理デバイスからなっており、各デバイスは$C_{ij}(1\leqq j\leqq k_i)$ページの記憶容量をもつ。しかし、本節の議論ではレベル間のページ配置のみに注意を払うので、各レベルは1つの論理デバイスからなるとして議論を行う。

各レベルiはレベルi+1しかアクセスすることができず、利用者はレベル1しかアクセスできない。そこで、レベルiに置かれたページを読み出すときにはレ

ベル2,…,レベルi−1を順にアクセスしなければならない。各レベルiに対して、そのレベルに時間1だけページを保持するために必要なコストをU_iとし、レベルi+1を1回アクセスするために必要なコストをR_iとする。レベルmは最下位レベルなので、$R_m=\infty$と定義する。以上をまとめたのが図1である。

利用者の要求(ページ参照列)は読み出し·書き込み列$O_1(x_1)\cdots O_n(x_n)$で与えられる。ここにO_iは読み出し(r)または書き込み(w)要求であり、時刻iに到着する。x_iはページであり、w(x)はレベル1上にページxを生成する。

レベル間最適ページ配置問題を解くために、次のような仮定をおく。

【仮定1】　$0<R_1 \ll R_2 \ll \cdots \ll R_m=\infty$　かつ　$0<U_m \ll U_{m-1} \ll \cdots \ll U_1$　□

下位レベルになるほどアクセス速度は遅くなり、ビットコストが下がることを考慮すると、この仮定によって議論の一般性は失われない。また、この仮定により、関係

$$R_1/U_1<R_2/U_2<\cdots<R_m/U_m=\infty$$

が成立する。

ページ参照列に対するページ追い出し方式は、各時刻tと各レベルiについて、レベルi中のページ数$P_{i,t}$とレベルi+1へのアクセス回数$Q_{i,t}$によって定義される。このとき、そのページ追い出し方式のコストは次式で定義される。

$$\sum_{t=1}^{n}\sum_{i=1}^{m}\{U_iP_{i,t}+R_iQ_{i,t}\}.$$

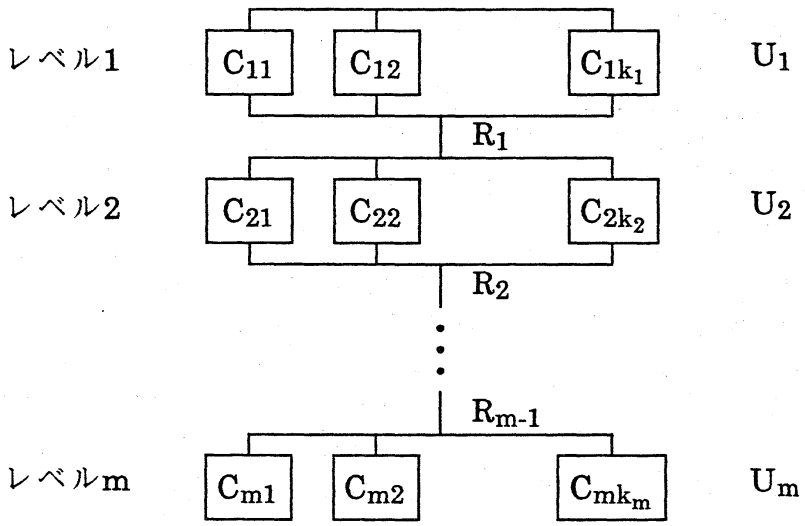

図1　一般記憶階層

ページ参照列に対する最適ページングアルゴリズムは、このコスト関数を最小にするページ追い出し方式である。これは通常の仮定である固定サイズのバッファを仮定していない。その代わりに、可変サイズのバッファにおいて平均バッファサイズ一定のもとでの下位レベルアクセスの最小化を行っている。実際のシステムにおいては記憶階層上にデータ以外のものが置かれることが多いので、それらとの間の記憶領域割り当ては負荷に応じて動的に行うべきである。本稿のコスト関数はこのような状況に対応している。

バッファサイズに上限がないことを考慮すると、上記のコスト関数は各ページ毎に独立して最適化できることが分かる。これは、ページyの転送によって別のページxが追い出されないためである。そこで、ページxに対する操作列$o_1(x)\cdots o_p(x)$とそれらの間の時間差(距離)の列$d_1,\cdots,d_{p-1}$についてxに対するコスト関数

$$\sum_{i=1}^{p-1} \left[o_i(x)\text{を処理後、}o_{i+1}(x)\text{を処理するまでに要するコスト}\right]$$

を最小化する。ここで、o_1のみがwになる可能性を持ち、残りのo_iはrであることに注意されたい。1つのデータが2回以上書き込まれる場合は存在しうるが、その場合は書き込まれたデータの各々をページに対応させることによって本稿のモデルに対応させることができる。また、ページxが複数回の変更を受ける場合への一般化は容易なので、本稿では省略する。o_1がrならばxに対する最適ページングが従来のものと同一になるので、本稿ではo_1がwの場合に限って議論する。

$o_i(x)$実行後のxの処理について考える。このときxはレベル1上にあるので、$o_{i+1}(x)$が到着するまで適当なレベルに追い出し、$o_{i+1}(x)$実行時にレベル1に再ロードする方法のうちの何れかが最適になる。ここで注意しなければならないのは、xを追い出すためにもコストがかかることである。読み出し･書き込みを考慮しないページングにおいては追い出しのためのコストは不要であったが、ページの生成を考慮するとこれが必要になるので、解析が複雑になり、また最適ページングアルゴリズムも異なったものになる。

そこで、先ずxをレベルj($1\leqq j\leqq m$)まで追い出すという条件のもとでの最適ページ追い出し方式を求める。このとき、xは$o_i(x)$実行後、直ちにレベルj_i($1\leqq j_i\leqq j$)まで追い出され、$o_{i+1}(x)$実行時にレベル1に再ロードする方法でなければ最適にはならないことが容易に示せる。また、$o_1(x)$実行直後にxをレベルjまで追い出しておくことによって、後続のステップについて追い出しコストを考慮しなくてもよくなるので、以後、この仮定のもとで議論を行う。仮定1によって、追い出されたレベルよりも下のレベルにxが置かれていてもページ参照列のコストは変化しないため、こう仮定することによって議論の一般性は失われない。このとき、xを追い出すためのコストは

$$R_1+R_2+\cdots+R_{j-1}=R_{j-1} \quad (仮定1により)$$

で与えられ、$o_i(x)$実行後$o_{i+1}(x)$を処理するまでのコストは

$$R_1+\cdots+R_{j_i-1}+d_i(U_{j_i}+\cdots+U_j)=R_{j_i-1}+d_iU_{j_i} \quad (仮定1により)$$

で与えられる(ただし、$j_i=1$の場合にはd_iU_1)。jを固定した場合にはxを追い出すコストは定数であり、$o_i(x)$実行後$o_{i+1}(x)$を処理するまでのコストは各i毎に独立して最適化することができるので、第2の式を各j_iについて比較することによって次の定理を得ることができる。

【定理2】　ページxをレベルjまで追い出す場合のxに対する最適な追い出しは、$o_i(x)$実行後、xをレベルj_iまで追い出す方策である。ここにj_iは$d_iU_{j_i}<R_{j_i}$かつ$j_i<j$を満たす最小の整数である。もし、そのようなj_iが存在しない場合にはxをレベルjまで追い出す方策が最適である。　□

【証明】　$o_i(x)$実行後$o_{i+1}(x)$を処理するまでの最小コストは

$$\min\{d_iU_1, R_1+d_iU_2,\cdots, R_{j-1}+d_iU_j\}$$

で与えられる。xをレベルj_iまで追い出す方策が最適ならば$R_{j_i-2}+d_iU_{j_i-1}>R_{j_i-1}+d_iU_{j_i}$が成立し、仮定1により$d_iU_{j_i-1}>R_{j_i-1}$となる。また$d_iU_{j_i}<R_{j_i}$が成立することも同様に示せる。仮定1から導かれる関係$R_1/U_1<R_2/U_2<\cdots<R_{j_i-1}/U_{j_i-1}<d_i$によって、$j_i$は$d_iU_{j_i}<R_{j_i}$かつ$j_i<j$を満たす最小の整数であることが分かる。また、そのようなj_iが存在しない場合は最小コストが$R_{j-1}+d_iU_j$で与えられる場合であることにより、定理の後半が示される。　□

これによって、ページxをレベルjまで追い出す場合の最適追い出しコストは

$$R_{j-1}+\sum_{i=1}^{p-1}\min\{d_iU_1, R_1+d_iU_2,\cdots, R_{j-1}+d_iU_j\}$$

になる。これを各$j(1\leqq j\leqq m)$について最小化することによって目的とするxの追い出しコストを求めることができる。そのための計算の複雑さは$O(mp)$時間であることが容易に分かるので、与えられたページ参照列に関する最適追い出しのコストは$O(nm)$時間で求められる。

従来の最適ページングアルゴリズムはページの変更を考慮してはいなかった。その場合の最適追い出し方式は次のようなものである。

> $o_i(x)$実行後、xをレベルj_iまで追い出す。ここにj_iは$d_iU_{j_i}<R_{j_i}$を満たす最小の整数である。そのようなj_iが存在しない場合にはxをレベルmまで追い出す。

しかし、本稿ではページを追い出すコストを考慮するため、次の定理を証明することができる。

【定理3】　従来の最適ページングアルゴリズムは本稿の最適ページングアルゴリズムと比較して、必ず処理コストが高くなり、最悪の場合には2倍のコストを必要とする。　□

【証明】　従来の最適ページングアルゴリズムが本稿のアルゴリズムよりも高い処理コストを必要とすることは自明である。$o_i(x)$を実行してから、$o_{i+1}(x)$を処理するまでに要するコストは、従来のページングアルゴリズムにおいては

$$\min\{d_iU_1, R_1+d_iU_2,\cdots, R_{m-1}+d_iU_m\}$$

であり、本稿のページングアルゴリズムでは最適な追い出しレベルをjとして、

$$\min\{d_iU_1, R_1+d_iU_2,\cdots, R_{j-1}+d_iU_j\}$$

である。従って、このコストに関しては本稿のアルゴリズムのコストの方が必ず高く(あるいは等しく)なる。次に、追い出しに要するコストについて比較する。従来のアルゴリズムでxを追い出すレベルをj'とする。そうすると、次のような条件を満たすiが存在する。

$$\begin{aligned} R_{j'-1} &= \min\{d_iU_1, R_1+d_iU_2,\cdots, R_{m-1}+d_iU_m\} \\ &\leqq \min\{d_iU_1, R_1+d_iU_2,\cdots, R_{j-1}+d_iU_j\} \end{aligned}$$

この式で表されるように、従来のアルゴリズムで追い出しに要するコストは本稿のアルゴリズムの処理コストを超えない。従って、定理の後半が示された。

□

従来のアルゴリズムが本稿のアルゴリズムの2倍のコストを必要とする場合も示すことができるが、紙面の都合で省略する。

3. レベル内での最適なページ割り当て方式

本節ではレベル内での最適なページ割り当てについて考察する。各レベルは複数の物理デバイスから構成されている。これらのデバイスの性能は同一であるとは限らないので、システム管理者は最大の効率を引き出すために、最適なページ割り当てを行う必要がある。本節では2つの場合についてこの問題を考える。1つは特定のデバイスに対する負荷を最大にする問題であり、デバイス間の性能差が大きい場合に有効である。もう1つの問題は各デバイスへの負荷を均等にしてできるだけ並列処理を可能にする問題であり、デバイス間の性能差がそれほど大きくない場合に有用である。

3.1　特定のデバイスへの負荷の極大化

議論に先だって、インターバルの定義を行う。レベルjにおけるデータxのインターバルはxがレベルjにロードされた時刻tとそれが追い出された時刻t'の組(t, t')である。xはレベルjに複数回ロードされる可能性があるので、レベルjにおけ

るデータxのインターバルは一般に複数個ある。また、各インターバルとレベルjへのアクセスが1対1に対応していることに注意されたい。インターバル(t_i, t'_i)は$t_i \leqq t$かつ$t \leqq t'_i$を満たすときに時刻tと重複するという。インターバルの集合はそれらのうちk個が重複する時刻tが存在するときk重の重複を持つという。

計算機システムのバージョンアップなどに伴って、特定のレベル中に性能のかなり異なったデバイスが入ってしまう場合がある。このような場合には、高性能のデバイスに対してより多くの負荷をかけることによってシステム全体の性能を上げることができる。しかし、この場合にも各デバイスの容量は一定なので、これを考慮しながら負荷の最大化を行う必要がある。本節ではこの問題について考察し、それを解くアルゴリズムを提案する。負荷極大化問題は次のように定式化することができる。

【問題4】　n個のインターバル(t_i, t'_i)からなる集合とデバイスの容量kを与える。このとき、$k+1$重以上の重複を持たないインターバルの部分集合で要素数が最大のものを求めよ。　□

これによって特定のデバイスに対する負荷を最大にすることができる。レベル内のデバイスの性能の順序が既知の場合には、最高の性能を持つデバイスに対してページ割り当てを行い、残ったインターバルについて2番目の性能を持つデバイスに割り当てるインターバルを決定するというように順次インターバルの割り当てを行うことができる。ここで、次のようなアルゴリズムを考える。このアルゴリズムが問題を解くことが以下で示される。

【アルゴリズム5】

ステップ1:　インターバルをt'_iの昇順にソートして、改めて$(t_1, t'_1), \cdots, (t_n, t'_n)$とする。

ステップ2:　$i=1, \cdots, n$について次のステップを繰り返す。

(a) インターバル(t_i, t'_i)と既に出力したインターバルからなる集合がk重の重複を持つならば、そのインターバルを捨てる。

(b) そうでなければ(t_i, t'_i)を出力する。　□

このアルゴリズムによって出力されたインターバルの集合が目的とする要素数最大の部分集合である。この事実は次の定理によって証明される。

【定理6】　アルゴリズム5は問題4を解く。　□

【証明】　インターバルの数nに対する帰納法による。$n=1$の場合には明らかに定理が成立するので、$n=N$で定理が成立すると仮定しよう。このとき出力されたインターバルの集合をS、$|S|=s$とする。定理を証明するために先ず、次の主張を示す。

【主張7】　要素数Nのインターバル集合に対してインターバルLを加えた集合を考える。ただし、Lは集合中で最大のt'値を持つ。{L}∪Sがk+1重の重複を持つならば、N+1個のインターバルからなる集合はk重の重複を持つs個のインターバルからなる極大部分集合を持つ。そのような重複を持たないならば、この集合はs+1個のインターバルからなる重複度kの極大部分集合を持つ。　□

【証明】　第2の場合には明らかに主張が成立する。第1の場合については、s個のインターバルからなる部分集合が存在することは明らかである。その極大性を示すために、s+1個のインターバルからなる部分集合が存在すると仮定する。s+1個のインターバルのうち1つは帰納法の仮定よりLである。{L}∪Sがk+1重の重複を持つので、L以外のs個のインターバルはSよりも短くなければならない。しかし、そのような集合が存在すればアルゴリズム5はそれをSとして出力する。これは矛盾であるから、主張が証明された。　□

N+1個のインターバルからなる任意の集合はNインターバルの集合に1つのインターバルを追加することによって構成できるので、各nについて定理が成り立つ。　□

次にアルゴリズム5の計算複雑度について考察する。このアルゴリズムの自明でない部分はインターバル(t_i, t'_i)がそれまでに出力したインターバル集合とk重の重複を持つ時刻が存在するかどうかの判定である。以下、我々は$O(k)$時間でこのテストを行う手続きを求める。従って、アルゴリズム5の全実行時間は$O(nk+n\log n)$になる。先ず、次のデータ構造を考える。

【データ構造8】　長さkの一次元配列a[u]。各a[u]の初期値は−1。　□

a[u]は次に処理するインターバル(t, t')をそれまでに出力したインターバル集合に追加した集合がu重の重複を持たないための時刻を示す。即ち、a[u]<tならば、そのようなことが起こらない。従って、a[k]とtを比較することによって(t, t')を出力してよいかどうかの判定を行うことができる。各t_iは正であるから、最初に処理されたインターバルは常に出力される。次の補題はこの配列の更新が$O(k)$時間でできることを示す。

【補題9】　(t_j, t'_j)をアルゴリズム5によってテストされるj番目のインターバルとする。各jとuについて次の式が成立する。

$$a[u]|_j = \begin{cases} a[u-1]|_{j-1} & a[u-1]|_{j-1} > t_j \text{の場合} \\ a[u]|_{j-1} & \text{その他の場合} \end{cases}$$

ここに、$a[u]|_j$は最初のj個のインターバルを処理した後のa[u]の値である。　□

【証明】　最初のj-1個のインターバルを処理した後、出力されたインターバルのうちu−1個が定義により時刻$a[u-1]|_{j-1}$と重複している。もし、$a[u-1]|_{j-1} > t_j$ならば(t_j, t'_j)もこの時刻に重複するので、$a[u]|_j = a[u-1]|_{j-1}$となる。そうでない場合には$a[u]$の値は変化しないことが容易に示される。　□

この補題により、アルゴリズム5のステップ2は次のように実現できる。

【プログラム10】
```
For i=1 to n do
    If t_i > a[k] then
        Output (t_i, t'_i);
        For j=k down to 2 do
            If a[j-1] > t_i then a[j] := a[j-1];
        done;
        a[1] := t'_i;
    end if;
done.
```
□

3.2　各デバイスへの負荷の均等化

特定のレベル中のデバイスの性能が揃っている場合には、各デバイスへのアクセス回数をできるだけ均等化することによって、並列処理による効率向上を行うことができる。しかし、各デバイスの容量C_{jk}は必ずしも等しいとは限らないので、これを考慮する必要がある。本節ではこの問題について考察する。負荷均等化問題はインターバルの概念を使って次のように定式化される。

【問題11】　n個のインターバル(t_i, t'_i)からなる集合とデバイスの容量の集合$\{C_{j1}, \cdots, C_{jk_j}\}$を与える。このとき、次の条件を満たすインターバルのデバイスへの割り当てを求めよ。

(1) 容量C_{jk}のデバイスに割り当てられたインターバルの集合はC_{jk}重以上の重複を持たない。

(2) 各デバイスに割り当てられたインターバルの数の最大数と最小数の差が最小になる。　□

この問題は次のアルゴリズムによって解くことができる。

【アルゴリズム12】

ステップ1:　インターバルをt'_iの昇順にソートして、改めて$(t_1, t'_1), \cdots, (t_n, t'_n)$とする。

ステップ2: 各デバイスに割り当てられたインターバルの数を表すカウンタを0に初期化する。

ステップ3: $i=1,\cdots,n$について次のステップを繰り返す。

(a) カウンタの値がもっとも小さいデバイスをkとする。

(b) インターバル(t_i, t'_i)とそのデバイスに既に割り当てられたインターバルからなる集合がC_{jk}重の重複を持たないならば、インターバルをデバイスkに割り当てて、カウンタの値を更新する。

(c) そうでなければカウンタの値がkの次に小さいデバイスをkとして(b)に戻る。 □

ステップ3では、割り当てられたインターバルの数が少ないデバイスから順に次のインターバルを割り当てられるかどうかのテストを行っている。この数が等しい場合には、容量の小さいデバイスを優先する。次の定理によってアルゴリズム12の最適性が示される。

【定理13】 アルゴリズム12は問題11を解く。 □

【略証】 インターバルの数nに関する帰納法による。$n=1$のときは自明に定理が成立するので、$n=N$でも成立すると仮定する。このとき、N個のインターバルからなる集合にインターバルLを追加した集合を考える。ただし、Lはこの集合中で最大のt'値をもつものとする。元のN個のインターバルを最適に配置した状態で、最も多くのインターバルを割り当てられているデバイス(複数のデバイスが存在する場合には容量が最も大きなもの)をKとする。LがK以外のデバイスに割り当てられるならば、明らかにそれは最適配置になる。LがKに割り当てられた場合でもアルゴリズム12は最小長さでの詰め合わせを行うので、最適配置になることが示せる。 □

次にアルゴリズム12の計算複雑度について考察する。$n \leqq k_j$の場合には問題11は自明な解を持つので、一般性を失うことなく$n > k_j$と仮定できる。アルゴリズムの初期設定にはインターバルとデバイスのソートが必要になるが、上記の仮定によりそのオーダーは$O(n \log n)$時間である。また、ステップ2は$O(k_j)$で処理できるので、この仮定によりアルゴリズム全体のオーダーには影響を与えない。ステップ3については、3.1節で示したように重複度の判定がデバイスの容量に比例する時間で行えることと、カウンタの値を更新した後のデバイスの並べかえが$O(k_j)$時間で行えることを考慮すると$O(n \sum_{k\in\{1,\cdots,k_j\}} C_{jk})$時間で処理できることが分かる。従って、アルゴリズム12の計算複雑度は$O(n \log n + n \sum_{k\in\{1,\cdots,k_j\}} C_{jk})$時間となる。

4. むすび

通常のページ追い出しアルゴリズムは固定サイズのバッファを仮定してページングを行うが、本稿ではより実際的な可変サイズのバッファを仮定し、平均バッファサイズ一定のもとでの下位レベルアクセス回数の最小化を行った。本稿では、ページ参照列が読み出し･書き込み列からなるとしたので、これまでに知られている最適な可変サイズページングと比較しても、最良の場合には1/2のコストで処理を行うことができる。

本稿のアルゴリズムは全て、ページ参照列があらかじめ与えられているというオフラインスケジューリング問題に対応している。そのため、今後、オンラインの場合への対応が必要になる。ただ、レベル間の最適配置アルゴリズムは従来のページングに対応するものなので、OSページングの理論的成果を見ても分かるように、実際的なオンラインページングアルゴリズムの性能をよく代表していると考えられる。しかし、3節で述べたレベル内の最適配置問題については従来考察されていないので、その点について拡張が必要である。

もう1つの課題は、ページの一括転送である。多重レベルの記憶階層においては、下位レベルのページサイズは上位レベルのページサイズよりも大きく、複数のページを同時に転送することによってアクセス速度の遅さを一部カバーしている。このような場合には、一括転送するページ集合の選択問題等の困難な問題が生じる。

謝辞 本稿の負荷極大化アルゴリズムの原型を考案した本学大学院生の加藤研児氏に感謝します。なお本研究の一部は文部省科学研究費試験研究の援助を受けている。

参考文献

[COFF73] Coffman, E.G. and Denning, P.J., *Operating Systems Theory*, Prentice Hall, 1973 (邦訳:川口, 藤井, オペレーティングシステムの理論, 日本コンピュータ協会, 1987.).

[PRIE76] Prieve, B.G. and Fabry, R.S., "VMIN - An optimal variable-space page replacement algorithm", Comm. ACM, Vol.19, No.5, May 1976, pp.295-297.