計算機システムの最適化制御*

特に TSS の制御について

大須賀　節　雄**

Abstract

This paper discusses on the optimization controls of computer systems. The optimization control is a programmed control of the computer system for itself under specified optimality criterion and so, as the cost of the control (overhead) and the rewards of it (e.g. decrease of idle time of CPU as the results of the control) have equivalent but opposite effects on the evaluation function, control program must be powerful enough to exercise effective control on the one hand and at the same time as simple as possible on the other hand.

In this paper, general concept of optimization control and a procedure for developing control program are first developed and then few examples, particularly the case of a time sharing system with segmentation and paging control more or less precisely, are described.

The auther thinks the optimization concept will be progressively important with the wide spread use of on-line computer systems.

ま え が き

この論文の目的は計算機システムを最適化する制御プログラムについて述べることである．制御プログラムは計算機のオン・ライン利用のもとでより重要なもので，計算機システム自体を動的な制御対象とみなして積極的な最適制御を行なうための，プログラムである．従来のオペレーティング・システムにこれに相当するプログラムが含まれている場合もあるが，ここでは上述の目的をもっと明確な形で捕えるために制御という面を強調し，これを独立なプログラム体系として扱かう．制御プログラムはシステムを最適化すると同時に極力単純化がなされねばならないので，実際の問題はプログラム作成以前にあり，プログラム自体は短かい単純なものとなる．したがって，実際にはシステム・プログラムの中の，ごく一部を占めるのみであろう．

以下 **1.** で計算機システムの制御について一般的概念を，**2**，**3** で実例として実時間専用処理装置および2次元アドレス方式を用いた TSS について述べる．

1.　計算機システムの最適化制御

1.1　最適化制御の意義

システム最適化のためには評価函数が存在せねばならない．一般にQなる評価函数について $I=\int Qdt$ を最小（大）にするという形で最適化が遂行される．最適化の基本概念は計算機システムでもアナログ制御系でも大差ない．計算機システムの場合の特徴は制御を行なう主体と対象が同一のものであり，制御の結果得られる利得と制御のコストが同じ価値を有しいることである．これは評価函数の中で制御コスト（具体的には制御のためのオーバーヘッド）が大きな割合を占めることであり，制御プログラムが過大になることは許されない．

仮りに評価函数が制御コストに依存する項 (I_1) と，これを0とした時残る項 (I_2) の和として表わされるものとする．制御の程度を示す何等かの量（たとえば

* Optimization Controls of Computer Systems, -Particularly on the Control of a TSS, by Setsuo Osuga (Institute of Space and Aeronautical Science, University of Tokyo)

** 東京大学宇宙航空研究所

制御に要する時間，プログラムの長さなど）を横軸に，評価量を縦軸にとって示すと **Fig. 1** のような2種類

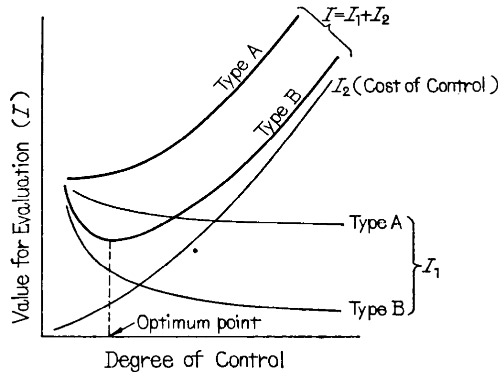


Fig. 1. General tendency of the effectiveness of control

の傾向をとる場合が考えられる．最適制御とは制御対象に固有な情報を利用して I を最適化するように制御手段をきめることである．このことは制御主体がシステムの将来の状態を予想し得ることであり，前記 I_1 は制御が正しく行なわれる限り増大はしない．**Fig. 1** において A のような傾向をとるのは I_1 の減少の程度が I_2 の増大の割合に達しないからで，制御コストが減少すれば B の型に移行する．これができない場合，たとえば利用すべき情報が存在しないか，または極めて利用しがたい場合は積極的な最適制御の効果が期待できない場合で，すべての操作を On-demand 方式で行なうのが最良であることを意味する．実際の場合多くは B の型に属することが予想され，したがってシステムの最適化が可能であるという前提のもとでシステム設計が進められるべきである．

1.2 システムのモデル化—利益付き確率過程における手段決定の問題

計算機システムは多数の独立な装置から構成され，システム内に多数のユーザー・プロセスが様々の処理段階で存在している．これら各ユーザー・プロセスの状態によってシステムの動的な状態が表わされていると考えることができる．ユーザー・プロセスは各ユーザーにより随意に作成されるものであり，またシステム内にドラム，ディスクなど確率的な要素を含んでいるから，システムの状態の変化も，確率的なものである．システム内に無限に成長する要素が存在しなければシステムのとる状態の数は有限のものであり，その間を確率的に遷移する有限状態の確率過程としてモデル化される．

このモデルは状態 i のとき dt 時間内に $Q_i dt$ なる評価量への寄与分を生ずるものとする．すると状態の遷移に応じて一連の評価量増分を生ずるが，状態間遷移確率が求まっていれば長時間の平均的な評価函数 Q が求まり，さらに評価量が $I=\int Q dt$ として得られることになる．この価は遷移確率が変わると変化する．

他方システムの各状態に対応していくつかの選び得る手段があり，そのいずれを選ぶかによって，次の状態への遷移確率が異なるものとする．それに応じて上記評価量も変化する．状態 i には選び得る手段が m_i 個あるとし，状態数が N とするとこのシステムには $\prod_i^N m_i$ とおりの遷移確率の異なる確率過程があり，したがって I の価にもこれだけの種類がある．この中から I が最小（大）となるような状態－手段の組み合わせを選ぶことができる．計算機システムにおける制御とは“I が最小（大）となるように，制御プログラムが各状態に対応する手段を選んで実行すること”である．

このように計算機システムの制御問題は利得付きの確率過程における最適手段決定の問題であり，最適手段はシステムの各時点での状態（またはその時点にいたるまでの経過）に対応してシミュレーションなどの方法により唯一のものに決定される．とくに単純マルコフ過程の場合には解析的な方法が用いられる[1]．このようにして最適手段は予かじめ求めておいて制御プログラムに教えてやることができる．制御プログラムは各時点においてシステムの状態を検知し，それに対応する手段を教えられたとおりに選んで実行すれば良いのであって，制御プログラム自身が最適手段決定のプロセスをその都度繰り返えす必要はない．

1.3 制御プログラム作成上の問題

制御プログラムをいかに作るかについて一般的な方法は未だ存在しない．今のところシステム・デザイナーの経験，分析力，勘などに頼らざるを得ない．制御プログラムの作成に到るまでの手順のうちで大きな問題として，（1）システムの状態の決定，（2）状態―手段の対応関係の表現方法，（3）プログラムの作成法がある．

（1） システムの状態の決定：

システムの状態はシステムによっては最初からきま

ってしまっているものもあるが，効果的な制御を行ない得るように，決定することのできる場合も少なくない．上述のように最適手段は，状態がきまれば比較的一般的な手順で求まるから，状態の決定がまず第一の問題である．システムの状態をきめるということは，待合わせ列の構成，その処理，待合わせ列間の関連などトラフィック制御の基本方式をきめることを意味し，その重要性は明らかである．また，それと同時にシステムの状態は個々のシステムによって異なり，一般的な方式も存在しないことも明らかである．ただ次のような手順をとることは効果的である．

（i） すべて On-demand の方式でシステムのトラフィック制御を行なうための概略の方式を求める．

（ii） システムに固有の情報を十分に活用し，制御に利用するにはさらにどの程度の機能を付加したら良いかを検討する．

（i）によって最適化制御が行なわれない場合でもシステム内のトラフィック制御のために必要な条件が求まる．これだけのものは（ii）にも含まれねばならない．このような手順をとることにより，最適化制御のために特に配慮されねばならないことと，基本的なトラフィック制御の問題を切り離して考えることができる．

（2） 状態と手段の対応の表現：

システムの状態と最適手段の対応をどのような形で記憶するかという問題も無視できない問題である．状態と手段とを羅列した対応表を持つというような，一般的ではあるが所要メモリーや時間が状態数に比例する方法は実用的ではない．状態の数はシステムの複雑さに応じてほぼ指数的に増大するから，制御プログラムの所要メモリーや時間もたかだか状態数の対数値に比例する程度の増大で済むような方法でなくてはならない．

状態と手段との対応もシステムごとに異なるから，細かな点はともかく，一般的には制御プログラムがその都度判断を下すことを減らすような方法を考える．先に最適手段の決定は制御プログラムと全く独立に行なわれ，プログラムは状態との対応を機械的に行なえば良いとしたと同様に，状態と手段との対応自体もできるだけ簡単化する方法を別途見出す必要がある．たとえば状態 S_i と選ぶべき手段 P_i との間に比較的簡単な函数関係 $P_i=f(S_i)$ を見出すことができるなら，プログラムは，この演算を機械的に遂行するのみで済む．

このように計算機システムの制御では，問題の大半はプログラム作成以前にシステム解析という形で行なわれるもので，プログラム自身は短かい，単純なものとしなくてはならない．

（3） プログラム作成上の問題：

上述の問題に加えさらにプログラム自体にも制御内容にできるかぎり無関係となるように作られねばならないという問題がある．それは（i）最適手段の決定は実際には面倒な問題で，結果を待ってからプログラム作成にかかるのではシステム開発が遅れる．（ii）システム構成や性能，使用条件などが変化すると最適手段も変わるから，それに適応できなくてはならないからである．この要求は制御プログラム簡単化の要求と相反する．なぜなら，プログラムを簡単化するには一般性を犠牲にして，システム固有の性質を利用せねばならないからである．この相反する要求を同時に満足する方法が見出されねばならない．たとえば前述のように $P_i=f(Si)$ の関係を導入することは，システム固有の性質を利用することであるが，システムの性質までが変わるような大変更でない限り，多少の変更によって函数形が変化せず，しかも，この中に含まれるパラメータの調整によって，これに適応できるというものであればより好ましい．

以上のことから理想的な制御プログラムは（1）少数のデータと短かい単純なプログラムからなり，（2）このデータのみに基づいて，最適手段の選択が行なわれ，（3）データを変更するのみで制御の特性が広範囲に，かつ連続的に変わり得るものということになろう．

2. 実時間専用処理装置（MARS 101）の例

1. で述べたようなシステムの最適制御を開発当初から明確な形で意図し，ほぼ上述の線にそって制御プログラムが開発された例として，やや旧聞に属するが国鉄の座席予約装置（MARS 101）がある．MARS 101 について述べるのが目的ではないから，**1.** の一般論の一例と見なしてこれを多少変形して述べることにする．したがって，以下の記述は MARS 101 の方式そのものではない．

このシステムでは多数端局から入力（呼と呼ぶ）が発生し，これらは，システム内では CPU を介していくつかの独立な処理装置を一定の順序で通りながら処理される．CPU 内では，これら各処理装置について

待合わせ列が形成される．CPU および各専用の処理装置の実効率を上げることが criterion となり，このために各処理装置から CPU への割込みの可否および優先順位を制御することが必要であった．この制御のための情報，すなわちシステムの状態は，各待合わせ列内の呼の分布によって表わされた．以後簡単のため制御の目的を，“システムの状態に基づいて最優先処理する呼をきめる”ものとする．仮りに4本の待合わせ列があり，各待合わせ列には最大4個までの待合わせが許されるものとすると，各列に存在する呼の数は0から4までの5通りであり，システムの状態は，$5^4=625$ 通りとなる．選ぶべき手段は各状態について最大16個の呼のうち最優先するものを選ぶという16通りである．

制御プログラムはシステムの状態から，この16種の手段の一つを選べば良い．これは逆に625通りの状態を16通りの手段ごとに分類することを意味する．状態の表示を各待合わせ列中の呼数の並置によって行なうとすると，たとえば（0134）と（4310）とでは選ぶ手段は非常に異なるが，（3222）と（3322）では大きな相違のないことが予想されたので，このような性質を利用して準最適分類ともいえる分類法が用いられた．各待合わせを一連の文字 A～P で表示する．A～P の並べ方には 16！通りがあるが，これを **Fig. 2** のように並べたとする．待合わせ列中 A～P の各場所が占められていたら，配列のその文字に対応する位置を 1，そうでなければ0とする．このように呼の分布による16ビットの 1，0 の配列ができる．制御プログラムは，この配列を左端より見て最初の1を取り出し，その位置に対応する呼を最優先にする（実際にはその呼の属する待合わせ列に関する各種の優先順位を高くする）という簡単なものである．**Fig. 2** の例ではまずGが見出される．すなわち，この方法では図の斜線を施こした部分の呼の分布がいかなるものであってもGが優先されるわけで，この手段には24の状態が属している．このような方法は分類を簡単化するための制約がついており，必ずしも最適手段が選ばれるとは限らない．しかし，この方法は A～P の配列を変えることにより分類の仕方を変えることが可能で，その中には最適分類に近いものを含む可能性は大きい．さらにこの配列を変更することによって制御の特性も大幅に変化する．この方法を **1.4** の理想的な制御プログラムの基準と照らし合わせてみると，データ語の少ないこと（この例では16ビット），プログラムが単純なこと，データを変えることにより制御の方式が大幅に変わり得ることなどで，その条件を満たしているが，最適手段と状態との対応がある程度制約された条件内でしかできないという欠点もある．このシステムでは，この欠点が実際上の制約にはならず，また，そのような予想に基づいて，この方式を用いたのであるが，これが一般的な方式でないことは，いうまでもない．

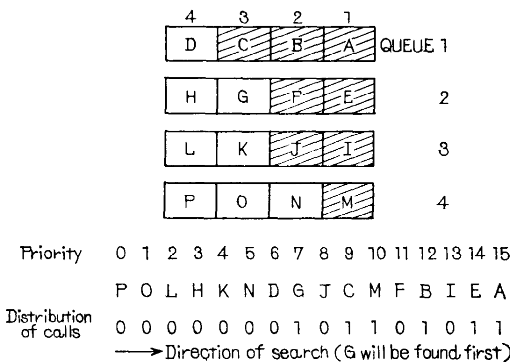


Fig. 2. A method to distribute every possible states among finite number of policies

制御プログラムがここまで単純化された結果，生じ得るすべての状態が16ビットの 1，0 の配列によって表示され，プログラムはこれを単なる配列として処理するから，いかなる状態にたいしてもプログラムの動作は同じである．これはプログラム作成のみならずデバッグにおいても重要な点で，理論的にはある一つの状態についてプログラムが正しく動作するなら，他のすべての状態にも正しいことが保証されるのである．

3. 2次元アドレス方式を用いた TSS の最適制御

システムの最適制御のもう一つの例として，2次元アドレス方式を用いた TSS の制御プログラムについて述べる．この方式も **1.** の一般手順にほぼ忠実に従って求められたもので，実際に HITAC 5020 TSS に用いられている．

3.1 システムの概要

以下に述べるシステムにはプログラムの動的なリロケーションを可能にするために，ユーザーの使用する name space と physical な location との対応をとるハードウェアとしての機能が付加されている．この装置を DAT（Dynamic Address Translator）と呼

ぶ．ユーザーはセグメント名およびセグメント内のアドレスを指定するが，これは DAT により Mapping Table (MT) という変換表を介して physical な location に変換される．プログラムおよびデータはページと呼ばれる一定の長さのブロックに分けられ，これが1単位となって relocation が行なわれる．MT 自身も一つのセグメントでありページングされる．このシステムでは1ページは256語になっている．

MT は Descriptor と呼ばれる語の集まりで，これはセグメントまたはセグメント内の各ページを定義して，それらの physical な location を指定するもので， segment descriptor (SD) と page descriptor (PD) とがあり，したがって MT も Segment Descriptor Table (SDT) と，Location Page Descriptor Table (LPDT) とに分けられる．PD はページングされている各プログラムの各ブロックの先頭の physical location を指し，SD はそのセグメントの LPDT の先頭の physical location を指している．

MT 自身もページングされるから，SDT が1ページ以上になる場合 SDT のページの先頭を指す segment page descriptor があり，これが集まったものを Segment Page Descriptor Table (SPDT) と呼ぶ．

この SPDT の先頭アドレスを指すレジスターがあり，これを Descriptor Base Register (DBR) と呼ぶ．DBR によって，各ユーザー・プロセスの name space の基点が定められ，これを入れかえることによってプロセス間のスイッチが行なわれる．以上を簡単化して示したのが **Fig. 3** である．

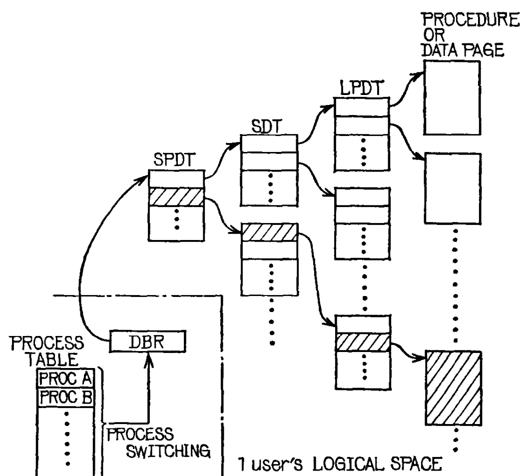


Fig. 3. Structure of a user's name space

同一のページが1回以上 refer されるときには，上記のような変換をその都度行なわなくても済むように数個の Associative Register が用いられている．また refer しようとしたページが主メモリーにない場合には missing page fault の trap が生ずる．これは各 descriptor 内の特定のビットが on になっている場合に生ずるもので， Supervisor によってこのビットが管理されている．また各種のメモリー保護機能を有しているが，紙面の都合もあり，また以下述べることとは直接関連がないので省略する．

3.2 TSS の最適制御

3.2.1 最適制御の可能性

TSS の最適化制御について述べる前に，マルチ・プログラム方式のスケジューリングについて一考しておく．以下システムの最適化の criterion は， CPU の利用度最大ということとする．マルチ・プログラム方式では多数ユーザーのプロセスが共存し，適当に定められた slice time ごとにプロセス間のスイッチが行なわれるとする．全ユーザーは，(1) CPU のサービスをいつでも受けられるものと (2) 他のプロセスによって wake-up されなければ CPU のサービスを受けられないものの2種に分けられる．通常 (1) を Ready (2) を Blocked と呼んでいる．(1) のうち現に CPU のサービスを受けているものを Running と呼ぶ．

主メモリーの中には全プロセスの必要とするプログラムのごく一部がはいり得るのみであるから，プロセス間のスイッチを行なう場合プログラムの swap が必要となってくる．従来の方式では (1) ランにはいるプロセスの全プログラムを reload する (そのためにこれまではいっていたものを purge する) ものと，(2) ランになったプロセスが missing page fault を介して，動的に必要なページの swap を要求するものがあった．

最適制御がシステムに固有の情報を有効に利用することによって可能になるという点から推論すれば，このいずれの方式も最適な制御とはいえない． (1) の場合制御プログラムは reload されるプログラムが実際にどの程度利用されるかについては何も知らない．また，すべての動作があらかじめきめられた時間ごとに割込みによって開始され，ドラムやディスクの特性はとくに考慮されてはいない． このような場合 CPU

が入出力の終了を待たねばならない場合がしばしば生ずる．他方（2）の場合は実質的にはいかなる制御も行なわない On-demand の方式であり，ランにはいるプロセスがどれだけのプログラムまたはデータを要求するかという情報は全然用いていない．ただランになったのち必要になったページのみを swap する．(1) の場合に比し不必要なプログラムを読み出す無駄はないが，逆に必要なことが明らかなものを読み出すにも割込みによって行なうという無駄な手順を繰り返えすことになる．

他方 swap out に関しては，その時点までにメモリー・ページが使われたか否かという情報に基づいて行なわれる場合が多いが，これだけの情報をとるために特別な機能を必要とするばかりでなく，この情報が真に役に立つという保証がない．過去に使われたか否かという情報はこれから使われるという確率の推定にどの程度役立ち得るか明確でない．その上この情報は利用し難いという欠点もある．ページ利用の可能性は（1）各ページがどのプロセスに属すか，（2）それらのプロセスがランにはいる確率はどのくらいか，（3）ランにはいったとき，そのプロセスによってどの程度使われるか，によってきめられるべきものである．

このように（1）と（2）の両方式は，トラフィックまたはプロセスの制御の傾向が反対であるが，いずれも最適なものとはいえず，最適化の余地が残されている．

3.2.2 制御のための情報および Controlled Swap

システム最適化の手順としてまず制御に利用できる情報があるか否か，それはどのような情報かを知らねばならない．ドラムやディスクの特性，システム内のプロセスの分布状況，入出力待ちのプロセスの数などが制御情報として一般的なものである．その他 TSS では非常に重要な swap に関して次の情報が利用できる．

（1） ページ使用確率：

プロセスは通常多数のページを用いるが，これらページの中にはランにはいったとき，使用確率が他に比して高いものがある．このシステムのようにセグメントおよびページ化が行なわれる場合には Mapping Table (MT) を始めこのプロセスをランさせるために必要な数種類の制御用テーブル類があり，それらが使用される確率もそれぞれに異なる．したがって一つのプロセスに属す各ページを使用確率の大きさにより数段階に分類し，それにランク番号を付けておく．使用確率の高いものには，たとえばランにはいるとき直ちに CPU の control が移るページと，それに関する LPDT, AUT (Active User's Table－ユーザーに関する固有の情報を記憶するテーブル)，SNT (Segment Name Table－そのユーザーが使用中のセグメント名が記録されている)，MT その他がある．また多くのユーザーに共通に用いられているセグメントも一般のページと別のランク番号を与える．

（2） プロセスの状態と優先順位：

swap の対象となるプロセスは次にランにはいる確率によってきめられるのが合理的である．プロセスの状態については次節で述べる．優先順位は最適制御プログラムとは別の Job Scheduler (JS) できめられる．

これより swap in は最も優先順位の高いプロセスのうち使用確率の高いページ順に行なわれるのが合理的であり，swap out はこの逆となる．

swap に関してこのような情報が利用されるためには制御プログラムが swap を司さどらねばならない．このように制御プログラムの権限において行なわれる swap を Scheduled Swap または Controlled Swap と呼ぶ（以下 CS と略記する．In または Out の区別をする時は CSI または CSO と書く）．CS が行なわれても，同時に missing page fault による swap 要求が併用され，これを Requested Swap (RS) と呼ぶ．使用確率の不明なページまたは確率の小さなページはすべて RS により swap される．CS はプログラムにより制御されているから swap 用2次メモリーへの入出力命令の調整が可能であり，また CSI は主メモリー内に available page (physical なページで以下 AP と略記する）が存在する場合にのみ行なわれる．制御プログラムは実際にはさらに積極的に CS を用いてシステムの状態を最適化の目的により好ましい方向に変化させ，システム全体を制御することができる．その制御目的の一つが AP を常に適正値に保つという主メモリーの管理であり，これは RS が生じた場合 AP がないと swap in が行なわれず，その始末のために余分な処理時間を費やすので，これを防ぐためである．

3.2.3 プロセスの状態

制御のさいに時として制御プログラムが無駄な動作を行なうことがある．たとえば3.2.1の（1）の方式では次のランにはいっても，一度も使われないプログラムを多数読み出して来る可能性がある．前述の CS にさいして，すでに CSI が行なわれたページを再び

swap out することはこの種の無駄を意味し，これをできるだけ少なくしたい．そのために，ある程度まで CSI が進んだプロセスは CSO の対象から除外することにする．CSI はあるプロセスがランにはいる前に，使用される確率の高いページをあらかじめ読み込んでおき，それによって不必要な missing page fault とそれに伴うプロセス間のスイッチを少なくするほか，ドラムの負荷の均一化をねらっているので，CSI がある程度進んだ状態でランにはいるのが理想的である．そこでプロセスの Ready の状態の定義を次のようにきめる．

- CPU のサービスを受けられるもので，そのプロセスに関するランク R までのすべてのページが主メモリー内に準備されているもの（R は制御パラメータでこれを変えることにより制御特性を大幅に変更することができる）．

Ready の状態のプロセスは CSO の対象から除外する．

制御プログラムによる AP の管理は前述のように CS を介して行なわれるものであるが，そのためには CS の対象となるプロセスが存在せねばならない．AP の管理はプロセスの管理に結びついている．上述のように，Ready のプロセスは CSO の対象とならず，また Blocked のプロセスは CSI の対象とならないから，これだけでは AP の管理を満足に行なえない．実際には，もう一つの状態が存在している．すなわち，Ready の定義を上述のように従来のものより多少限定したため，これにはいり得ないものがあるので，これを Pending と名づけ別の状態として扱かう．この状態にあるプロセスは CSI および CSO のいずれの対象にもなるものとする．このようにプロセスの状態は次のようになる（**Fig. 4** 参照）．

- Ready（Running を含む） ● Pending
- Blocked

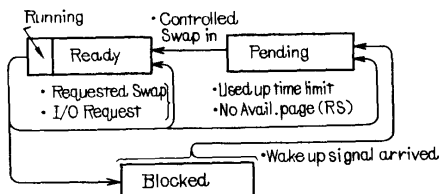


Fig. 4. States of the processes and their transitions

これら各状態とその間の状態変化は次のとおりである．

（1） Running から Pending への移行

（i） Slice time over

（ii） RS が生じたが AP がない—AP は，プログラムによって管理されているが，不確定要素のために AP が 0 となる場合もまれには生ずる．この場合は（i）と同様にみなされる．これはプロセスが AP の需要側から供給側に移されることを意味し，AP とプロセスの状態との関係が調整される．

（2） Running から Blocked への移行

（i） Block ルーチンを call する．または他の Process が Quit を call する．

（ii） Terminal への入出力要求—Terminal への入出力要求は一般に応答が遅く，しかも短時間に必ず終了するという保証がない（オペレータが席をはずすなど）ので，これを Blocked にする．

（3） Running から Ready への移行

（i） RS—ドラムやディスクへの入出力要求は不確定ではあるが，短時間内に必ずその処理が済むものであるから，状態をかえずに Ready の最後にもどす．これは前述の無駄の動作をなくす上で重要なことで，もしこのプロセスが swap out 可能な状態に移ると，入出力要求を出したプログラム・ページが swap out されてしまうことになる．

（4） Blocked から Pending への移行

（i） Wake-up—あるプロセスが Blocked 中のプロセスを wake up すると，wake されたプロセスは pending に移る．

（5） Pending のプロセスの制御および Ready への移行

Pending のプロセスは制御プログラムにより CSI または CSO がなされる．制御プログラムは Pending 待合わせの優先順位を見て，CSI の場合この順位にしたがって，CSO の場合にはこの逆に，対象となるプロセスを探す．各プロセスは CSI が行なわれると，主メモリー内にあるページのランクが一段上がり，CSO を受けると一段下がる．各プロセスごとに主メモリー内にあるページの最高ランクをそのプロセスのランクと呼ぶ．これは制御上重要な情報である．Pending の先頭プロセスのランクが CSI により高くなって R に達すると，そのプロセスは Ready に移行す

る．Ready になるともはや CSO は作用されない．Pending の待合わせ内プロセスのランクは待合わせの先頭が高く，後方に向って低くなる．

このように最適制御という目的のために，On-demand 処理では不要であった Pending という新しい状態を必要とすることになった．一般に最適制御はその一つの場合として On-demand の場合，またはそれに等価な場合を含むから多少複雑化するのはやむを得ない．

3.2.4　制御の方式

TSS 最適化のための criterion としては CPU の実効率が選ばれている．これによって求まる制御アルゴリズムと，あとにふれる Job Scheduler との共同により最終的には CPU 実効率をまし，かつ応答時間を小にすることを目的としている．最初から応答時間が criterion に選ばれなかったのはこれら 2 種の制御アルゴリズムが性質の異なるものであるから，オーバーヘッド，保守性，成長性などの点から考えて，これらを分離した方がより実用的と考えられたからである．応答時間に直接影響するのは Job Scheduler であるが，多数ユーザーのもとでは CPU 実効率を上げることは間接的に応答時間に寄与することになる．

CPU の実効率を悪化させる大きな原因として次の三つの場合が考えられる．

（a）　Ready の状態のプロセスが存在するにもかかわらず，CPU が動作できない場合—Available page (A.P.) が存在せず Swap out の終了を待つ．

（b）　ユーザー・プロセスは存在するのに Ready がない場合．

（c）　制御のためのオーバーヘッド．

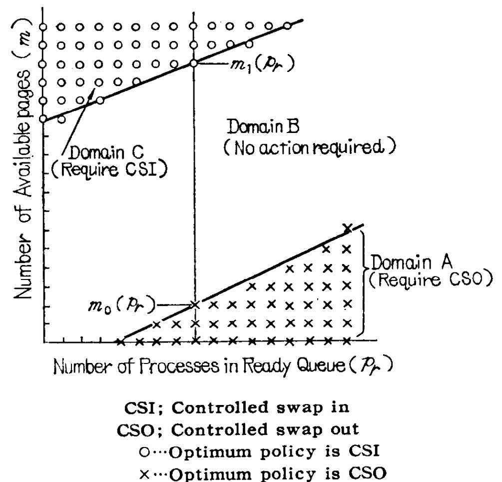
以上のうち（c）に関しては前述のように制御プログラム自身の問題である．(a) (b) に関しては，これらをできるだけ減らすためにシステムを適正状態に保つことが必要である．(a) は AP が十分にあること，(b) は Ready のプロセスが十分に存在することを要求しており，これらは相反するものであるからプログラムにより制御しなくてはならない．(a) (b) はともに CS によって制御されるのであるから，制御手段は（i）CSI を行なう．（ii）CSO を行なう．（iii）何もしない　の 3 通りである．どのページを対象とするかという細部に関する問題は，最初の選択がなされたのち行なえば良い．

最適制御に関する一般方式に従えば，この選択はシステムの状態に基づいて決定されることになるので，システムの状態を明確にする必要がある．上記手段に多少とも関係すると思われる情報には

（1）　Available page の数 (m)
（2）　Ready Queue 内のプロセス数 (p_r)
（3）　Pending Queue 内のプロセス数 (p_p)
（4）　Blocked Queue 内のプロセス数 (p_b)
（5）　Pending Queue 内のプロセスのランク (r)
（6）　ドラム・ディスク入出力待ちのプロセス数 (p_c)

などがある．上述のことから（1），（2）の重要性は明らかである．（3），（4）は CS を行ない得るか否か（対象とするプロセスが存在するか否か）を示す．（5）は実際には Pending Queue の先頭プロセスのランク番号によって代表され，Pending から Ready へ移行する距離を示している（ランク番号が r のときは（2）のプロセス数が r/R 個余計に存在すると考えられる）．(6) は CS が行なわれたが，まだ完了していない状態を示し，少時ののちには（1)～(5）の変化として現われる．

これら個々の状態を組合わせてシステムの状態とし，それに対応した最適手段を選ぶようにすることも可能だが，選ぶべき手段の数に比し状態の数が多いこと，各情報の利用価値が異なることなどを考慮してさらに単純化を行なう．



CSI; Controlled swap in
CSO; Controlled swap out
○…Optimum policy is CSI
×…Optimum policy is CSO

Fig. 5.　Optimum policy map in n-dimensional state space
(The case of $n=2$ is shown where system's states are represented by the nodes. An optimum policy corresponds to each of them)

仮りに用い得る情報が（1）と（2）のみであったとする．それらから生ずる状態は両方の数の組合わせで，これを **Fig. 5** のようにマトリックス表示をすると，各節点が状態を表わす．これら各節点に最適手段決定の方法によって得られた手段が対応するから，それを表示すると，図のような分布図が得られるであろう．この例のようにシステムの状態を構成する個々の数値について3種の選択がなされ，その3種が単純に分離されている場合には，多次元の状態においても選ばれる3種の領域が単純に分離され，混じり合うことはない．**Fig. 5** において Ready のプロセス数を固定してみると，AP について3種の領域ができる．この境界を m_0, m_1 とすると，これは Ready のプロセス数の函数で

$$m_0=f_0(p_r),\ m_1=f_1(p_r)$$

である．前記（3）～（6）の情報についても次元は増すが，これと全く同様にして

$$m_i=f_i(p_r,\ p_p,\ p_b,\ p_c,\ r),\ (i=0 \text{ or } 1)$$

が得られる．これより手段決定の方法は非常に単純で，

（i） $m \geq m_1$ でかつ $p_p \fallingdotseq 0$; CSI を行なう．

（ii） $m \leq m_0$ でかつ $p_p+p_b \fallingdotseq 0$; CSO を行なう．

（iii） 上記以外の場合何もしない．

である．ここまではシステムに関する特徴を用いてはいるが最適手段選択の近似化は行なっていない．

函数 f_0, f_1 の形は最適手段が求まるまでは厳密には定まらないが，函数形そのものは比較的単純なものとなることが予想されるので，これを簡単な形で近似化する．すなわち

$$m_i=A_i p_r+B_i p_p+C_i p_b+D_i p_c+E_i r+F_i \quad (i=0 \text{ or } 1)$$

である．ここでランク情報 r は Ready のプロセスの r/R 個分に相当するものとすると $E_i=A_i/R$ である．また p_b は手段決定のさいに独立な条件として用いられ，m の大きさの判定には大きな影響はないとみて，$C_i \fallingdotseq 0$ とする．結局

$$m_0=A_0(p_r+r/R)+B_0 p_p+D_0 p_c+F_0$$

$$m_1=A_1(p_r+r/R)+B_1 p_p+D_1 p_c+F_1$$

が得られる（厳密には p_c そのものも CSI によるものと CSO によるものに分けねばならないが，このような方式では CSI と CSO とが同時に行なわれている確率は非常に小さいからあまり問題はない）．

この方式では制御のパラメータとして，$A_0, A_1, B_0, B_1, D_0, D_1, F_0, F_1$ R の計9個があり，これらは理論的にあるいは実験的に求められねばならない．これらパラメータの値を変化することにより，制御特性を大幅に変化させることができる．

m_0, m_1 は状態変化のたびに常に修正されねばならない量である．状態の変化は各要因が独立に生じ，かつ要因の変化は原則として1単位ずつであるから，m_0, m_1 について差分をとり

$$\Delta m_i=A_i(\Delta p_r+\Delta r/R)+B_i\Delta p_p+D_i\Delta p_c,\ (i=0 \text{ or } 1)$$

とすると，Δp_r, Δr, Δp_p, Δp_c は ± 1 or 0 であるから，変化の生ずるごとに A_i, A_i/R, B_i または D_i を増減すれば良い．制御プログラムは AP の変化するごとに，具体的にはメモリーの要求が生ずるごとに call される．

制御プログラムの役割は m を m_0 および m_1 と比較するのみであるから数ステップで済むが，これに続いて CS を行なうとき，どのページを対象とするかについて 3.2.2 で述べた swap 用の情報が用いられる．なお実際には一度に swap される最大ページ数も定められており，これが制御パラメータに含まれる．

以上の制御はいわばプロセスとページを対象にした工程・在庫管理であり，P-P 制御と名づけておく．TSS では P-P 制御のほかにすでにのべたように，応答時間に影響するものとして Job Scheduler がある．この役割はあるプロセスが Ready または Pending の待合わせにはいるとき，すでに待合わせているプロセスと順位比較を行なって優先順位に応じた位置に配置することで，このさい同時に Slice Time の決定がなされることもある．この順位や Slice Time 決定の方法が Job Scheduler にとって重要な問題であるが，この種の制御に関しては，これまでにも多くの方法が発表されており，本稿では特に扱かわない．ただ Job Scheduling は P-P 制御とは独立に行ない得ることは重要である．P-P 制御が Job Scheduler によって定められた待合わせ列内の順序を制御情報に用いても，他方 Job Scheduler が P-P 制御を受けているプロセスを対象としていても，両方の制御が相互に干渉し合う必要は必らずしもないので，それぞれ独立に制御方式をきめ得る．

む す び

計算機システムの最適化制御について一般的概念と実例を述べた．最適化制御はオン・ライン・システムにとってより重要となろうが，効果的な最適化の方式をすべて定まった手順に従って求められるような一般的な方式はまだ存在せず，少なくとも現在の段階では

個々のシステムごとに，システム・デザイナーの経験と，システムを分析しかつ全体的に把握する能力とが要求されている．

本稿はその手順をできるだけ定式化することを試みたものだが，制御方法の決定において

（1） システムの状態をいかにきめるか

（2） システムの状態と手段の対応をいかに簡単に表現するか

という点が，システムの特殊性を利用するという事情もあってまだ一般化されない．今後の研究が期待される．

最後に本研究にあたって御助言を頂いた東京大学穂坂　衛教授，HITAC 5020 TSS に本方式を適用するにさいして各種の御協力を戴いた日立中央研究所島田正三，高橋延匡両氏，ならびに筆者と共に細部に到る検討をして戴いた同，稲富　彬（現明治大学），勝枝嶺雄，本林　繁，桑原　裕，益田隆司の諸氏に厚く感謝の意を表す次第である．

参考文献

1) R.A. Howard: Dynamic Programming and Markov Process, Technology Press and Wiley, 1960

2) 大須賀節雄：情報処理システムの内部トラフィック・コントロール，情報処理学会第7回全国大会予稿集，pp. 29，1966

3) J.H. Saltzer: Traffic Control in a Multiplexed Computer System, MAC-TR-20, June 1966

4) F.J. Corbató and V.A. Vyssotsky: Introduction and Overview of the MULTICS System, Proc. FJCC, 1965 pp. 185～196

5) E.L. Glaser, J.F. Couleur and G.A. Oliver: System Design of a Computer for Time Sharing Application, Proc. FJCC. 1965 pp. 197～202

6) V.A. Vyssotsky, F.J. Corbató and R.M. Graham: Structure of the MULTICS Supervisor, Proc. FJCC, 1965, pp. 203～212

7) C.T. Gibson: Time-Sharing in the IBM System/360 Model 67, Proc. SJCC, 1966 pp. 61～78

8) J.I. Schwartz and C.Weissman: The SDC time-sharing system revisited, Proc. A.C.M. National Meeting 1967 pp. 263～271

（昭和42年11月4日受付）