計算機システムの解析と制御 3－3
（1978. 11. 2）

OSの省力化と自動化

佐薙 充
（富士通）

1. はじめに

現在の制御プログラム（OS）に求められている課題のひとつは，より多くのインテリジェンスをもつことである。さまざまな場面で人間のかわりにOSが適切な判断を下し，自動的なシステムの制御を実現することが求められている。この報告では，これらの課題の中から

- 計算機システムの運転操作の省力化
- 自動的な最適化制御
- 計算機システムの設置先の運用政策を反映した自動制御

をとりあげ，富士通のOSでの実例を中心に現状を報告し，あわせて今後の展望を述べたい。

2. 運転操作の省力化

(1) プリンタ出力の省力化

プリンタ出力に関する操作を減らすために，ジョブの切れ目でOSからの指令によって用紙を切断すること，用紙の種別にもとづいて出力の順序を変更し，用紙の交換の回数をできるだけ減らすことなどがOSの標準的な機能となっている。後者の機能は形式の違う用紙を多種類使用するシステム（事務処理中心の典型的なシステム）で効果的である。また京都大学大型計算センタのシステムでは用紙の自動切断のほかに，出力結果を利用者が待っているターンテーブルまでベルトコンベアで運ぶこと，すぐに利用者に渡せない出力結果を保管して，利用者から要求があった時にあらためてとりだし，ターンテーブルまで運ぶことなどが，すべて人手を介さず計算機の制御によって自動的に行なわれている。

(2) オペレータコマンドの省略形

オペレータからOSに対する指令を省略形（たとえば指令の先頭一文字のみ）で入力することを許し，操作の負担を軽くしている。ファンクションキーのいくつかに，事前に特定の意味づけをしておき，コマンドおよびパラメータ列を入力するかわりにファンクションキーを押すだけですませる機能も同様の目的をもつ。

(3) オペレータコマンドの自動発生

イニシャルロード後のシステムの初期化処理，オンライン制御などの各種サブシステムの起動処理には，いくつかのオペレータコマンドを入力することが必要である。その理由のひとつはOSがモジュラな構造になっており，コマンドの入力のしかたでさまざまなオプションを選択できるようになっているためである。しかし，日常の定常的なオペレーションでは入力するコマンドは決まっており，システムの初期設定のたびに毎回同じ操作をするのはむだである。この操作を省くために，システムの初期設定時にOSが自動的に一連のコマンドを内部的に発生させる。オンラインの業務中にシステムがダウンし，そのシステムまたは予備のシステムを初期化して業務を再開する時には，この機能は単に操作を減らす点だけでなく，業務再開までの時間を短縮する点で重要である。

(4) 無人運転

運転操作の省力化の究極は無人運転である。 このうち，夜間にオペレータなしでジョブの処理を行なう形態は実用の段階に入っており，京都大学の大型計算センタで実際の運用にとり入れられている。 一般に無人運転の時間帯には，人手の操作を必要とするジョブの入力，出力は行なわれず，スプールファイル中にスタックずみのジョブの実行のみが行なわれる。 すべてのジョブの実行が終るか，指定された時刻になるとOSからの指令によって自動的に電源が切断される。何らかの異常でシステムが正常に動作しなくなった時も，自動的に電源が切断される。 また無人運転中に火災などの環境異常が発生した時に自動的にこれを検出し，守衛室に通報するような設備が同時に必要である。 図1に富士通のMシリーズでの無人運転システムの概略を示す。

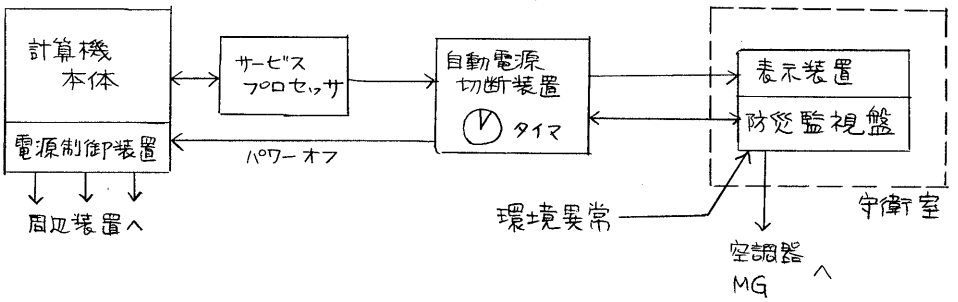

図1. 夜間無人運転システムの概略

(5) 入出力ステーション

以上に述べた省力化とは性格がやや異なるが，同様に計算機システムの運転操作の合理化，効率化をめざしたものに，セルフサービス方式の入出力ステーションがある。 これはセンタのオペレータではなく利用者が直接ジョブの入力，出力結果のとりだし，磁気テープの着脱などの操作を行なう方式であり，京都大学の大型計算センタで全面的に採用されている。 入出力ステーションには上にあげた入出力装置のほかに，OSからのメッセージを表示するコンソール，表示装置などがおかれる。 またジョブの結果をプリンタに出力する前にディスプレイで検索し，必要な部分のみを出力することもできる。

3. 自動的な最適化制御

計算機システムの使用効率を上げるために，人手によるいろいろな工夫が行なわれてきた。 それらのうちのいくつかは，OSによる自動的な最適化制御にとり入れられている。 その典型的な例は仮想記憶制御（demand paging）であるが，そのほかにも次の例がある。

(1) ダイナミックディスパッチング

ジョブの資源利用特性を実行時に計測し，その結果にもとづいてCPU割当ての優先度を変動させる。 CPUをよく使う（CPUバウンド）ジョブには低い優先度，入出力装置をよく使う（I/Oバウンド）ジョブには高い優先度を与え

る。　これによってCPU使用率が高まり，システムのスループットが向上することは数多くの実測例で検証されている。

(2) 入出力負荷の自動平滑下（I/O load balancing）

ユーザのジョブ用の作業用ファイルをディスク，ドラムなどの二次記憶上に割当てる時，入出力チャネルおよび入出力装置の使用率の計測値にもとづいて，使用率の低いチャネルおよび入出力装置を選択する。　これによって特定のチャネルや入出力装置に負荷が集中することを避けることができ，ボトルネックを除去できる。

4．設置先の運用政策を反映した制御

計算機システムの設置先は，システムの運用に関してさまざまな条件をもっている。　たとえば，複数の業務の間の優先順位，特定のジョブに関する完了期限，ファイルにおさめられている各種のデータをどの範囲のユーザに利用可能にするかなどである。　これらの運用上の政策は，従来はオペレーションの工夫や，計算機室の外でのいろいろのとりきめや申し合わせによって実行されてきた。　今後のOSは，これらの運用政策を反映したシステムの制御を可能にするだろう。すでに実現されているか，近い将来に実現されると思われるものには以下のものがある。

(1) 予算管理

ユーザが計算機システムの資源をどれだけ使ってよいかは，予算によって規定されているのが普通である。　これまでのシステムでの予算管理は，OSが収集した課金情報をもとに，定期的に使用実績を集計し，それを人間が予算と見比べてチェックする方法によることが多い。　今後のOSは，ユーザごとの予算と使用実績を常時把握し，予算を超過して資源を使おうとするジョブはスケジュールしないとか警告をだすとかの制御を行なうようになるだろう。

(2) ファイルに対するアクセスの管理

ファイルに対するアクセスの管理（どのユーザに対し，どれだけのアクセスを認めるかという制御）に関しては，設置先の企業体の組織に即した管理を可能にするものが求められている。　通常の計算機外の「情報」が組織に沿って流れ，管理されているのと同じことが，計算機システム内のデータに対しても求められている。　これにこたえて，組織と対応した構造をもつユーザとデータの登録簿をもつ，新しいタイプのアクセス管理機能をOSがもつようになっている。

(3) ジョブのスケジュール論理の精巧化

複数のジョブの実行の相互順序，依存条件をユーザが指定して，それに従ってOSがジョブのスケジュールを行なう制御（ジョブグループ制御），指定された締切り時刻が近づくにつれて優先順位を高くし，その時刻までに必ずジョブを終らせる制御（デッドラインスケジュール）など，運用政策を直接に反映する精巧なジョブ・スケジュール機能が求められている。

(4) 資源配分の自動制御

ジョブのターンアラウンドに関する設置先の運用政策を反映して，実行中のジョブに対する資源の配分を巧妙に制御する機能については，次の5章で詳説する。

5. 資源配分の自動制御

システム全体の単位時間あたりの処理量（スループット）を高めたいという要求と，個々のジョブのターンアラウンドやレスポンスをユーザが満足する範囲内におさめたいという要求は，しばしば矛盾する性格をもっている。これらの要求を実現するための制御が，ひとつのOSの中で独立して別々に行なわれるのが，これまでの通例であった。これに対し，富士通のOSTV/F4のSDM，MONITOR VIIIのHOPEは，ともに集中的なシステム資源管理方式を採用することによって，スループットとターンアラウンドという相反する要求を満たすための制御をひとつの論理で実現している。SDM，HOPEともその背景にある考え方は共通しており，設置先およびユーザが指定したターンアラウンドについての条件を満たす範囲内で，システム内の資源の利用率を高め，高いスループットを得ることをねらっている。

5.1 SDM（SYSTEM DECISION MANAGER）

SDMは，ユーザが指定したターンアラウンドに対する条件とシステム内の資源の使用状況の両者を考慮に入れて，最適なジョブの集合を主記憶上におくように制御する。SDMの制御の手段はスワップ（swap）である。ここでスワップとはジョブを主記憶上からディスク，ドラムなどの二次記憶へ追いだす制御（スワップアウト）と，追いだしたジョブを再び主記憶上に戻す制御（スワップイン）の両方を意味する。SDMの制御の内容は，ターンアラウンド保証のための制御とスループット増大のための制御に大別できる。

(1) ターンアラウンドを保証するための制御

(a) サービス率とパフォマンス目標

ターンアラウンドを保証するための制御を行なうために，各ジョブに対し，次の式で表わされるサービスと呼ばれる量を計測する。

$$\text{サービス} = A \times \text{CPU時間} + B \times \text{入出力回数} + C \times \text{実記憶量} \times \text{CPU時間} \quad (1)$$

（A，B，Cは設置先が指定する重みづけの係数）

ジョブに対し単位時間に与えられるサービスをそのジョブのサービス率と呼ぶ。SDMはシステム内のジョブあるいはジョブの集合に対し，サービス率を適正に配分する。この配分は設置先によって定義されるパフォマンス目標と呼ばれる関数を使って行なわれる。パフォマンス目標の例を図2，図3に示す。

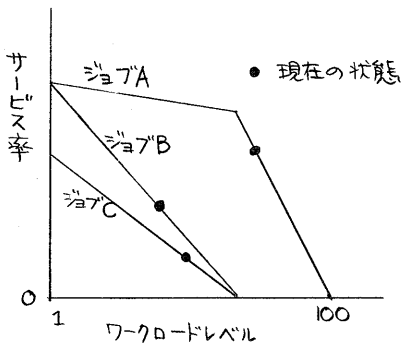

図2. ジョブに対するパフォマンス目標

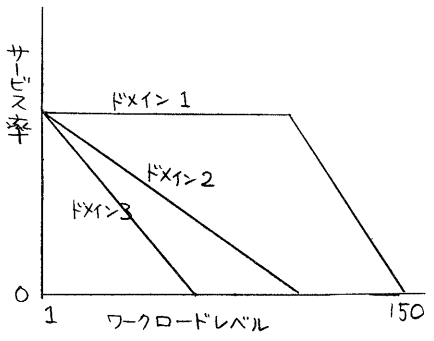

図3. ドメインに対するパフォマンス目標

パフォマンス目標はサービス率をワークロードレベルと呼ばれる量に対応づける単調減少の関数である。ワークロードレベルはサービスの不満足度を表わす無次元の量であり，ワークロードレベルが最小値（1）をとる時がサービスが最高に満足されていることを意味する。SDMは各ジョブ（あるいはジョブの集合）のワークロードレベルができるだけ同じ値になるように，大きいワークロードレベルをもつものに対してはより多くのサービスを与え，小さいワークロードレベルをもつものに対してはより少ないサービスを与える。図2の例でいえば，ワークロードレベルの最も大きいジョブAに対してより多くのサービスを与え，逆にジョブBに対しては与えるサービスを減らそうとする。

(b) ドメインに対するサービスの分配

サービスの配分の実際の制御は，各ドメイン（ジョブの集合）に対する配分とドメイン内の個々のジョブに対する配分との2段階に行なわれる。ドメインは性格の類似したジョブの集合であり，設置先で定義される。例としては，バッチ処理，TSS，オンラインのトランザクション処理などの処理形態ごとにジョブを分類し，さらにそれらをターンアラウンドやレスポンスに対する条件の違いに従って細分したものをドメインとする方法が典型的である。

ドメインに対するサービスの配分の制御は次のように行なわれる。

各ドメインに対してパフォマンス目標が設置先によって定義される。一定時間間隔でSDMは各ドメインのサービス率（ドメイン内ジョブのサービス率の合計または平均）とパフォマンス目標にもとづき，ドメインに対するサービスの過不足を判断し，ドメインの主記憶内ジョブ多重度の目標値（TMPLと呼ぶ）を増減する。ただし，ドメイン内に存在しているジョブの数を越えて，そのドメインのTMPLをふやしても効果がないので，TMPLはドメイン内の実行可能ジョブ数以上にはしない。

$$TMPL \leq \text{実行可能ジョブ数の平均値} + 1 \qquad (2)$$

主記憶内ジョブ多重度の変更の実施はスワップによって行なわれる。スワップのオーバヘッドが過大にならないように，TMPLの変更は比較的長い時間間隔（数十秒）をおいて行なわれる。しかし，多数のTSS端末からのLOGON（TSSジョブの開始）が短い期間に集中した場合は，主記憶内ジョブ多重度の変更が行なわれるまで，端末へのレスポンスが遅れる可能性がある。このため設置先が指定したドメインに関しては，TMPLの上限値は式(2)ではなく，次の式によって制御される。

$$TMPL \leq \text{実行可能ジョブ数の平均値} + N \qquad (3)$$

（Nは設置先の指定した値）

ドメイン単位でサービス配分を行なうねらいは，複数の業務を同一のシステムで行なう場合に，各業務に対する資源の配分率をユーザが直接制御できることにある。

(c) ドメイン内のジョブに対するサービスの分配

ドメイン内のジョブに対するサービスの配分は次のように行なわれる。

ドメイン内の主記憶内ジョブ多重度は，上に述べたTMPLの値以下に制御され，ジョブの数がそれより多い場合は，一部のジョブはスワップアウトされる。どのジョブを主記憶上におくべきかの判断にはパフォマンス目標が用いられるが，

各ジョブに対して直接パフォマンス目標が指定されるのではなく，パフォマンスグループのIDが指定される。　この指定はバッチジョブであればジョブ制御文で行なわれる。　パフォマンスグループは設置先により定義され，ジョブの進行の各過程（パフォマンスピリオドと呼ぶ）ごとにドメイン番号，パフォマンス目標，CPU優先度，主記憶優先度などの値が指定される（図4参照）。

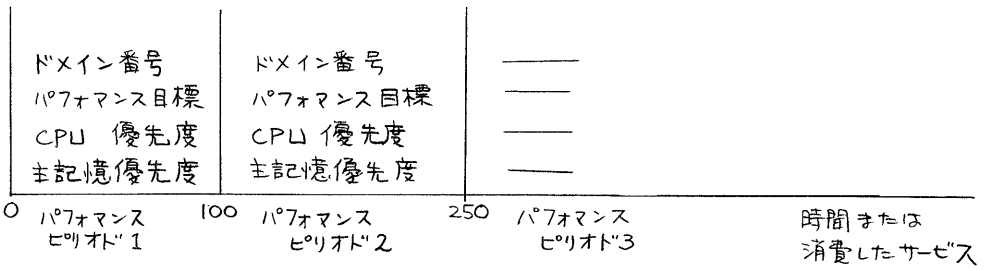

図4．パフォマンスグループ

パフォマンスピリオドは，ジョブが指定された量のサービスをうけるまでの期間または，一定の時間である。　ひとつのパフォマンスピリオドが終ると，そのジョブの制御は次のパフォマンスピリオドにもとづいて行なわれる。　CPU優先度とは，ジョブに対してCPUを割当てる時の優先順位である。　主記憶優先度とはジョブの実記憶に対する優先度であり，主記憶優先度の高いジョブは，実記憶を奪われにくいページリプレースメントアルゴリズムが採用されている。

(d) パフォマンスピリオドの意味

パフォマンスピリオドごとに，ドメイン，パフォマンス目標などを変えることができるようにしている理由のひとつは，これによって短いジョブに対しては高いサービス条件を与え，長いジョブに対してはターンアラウンド条件をゆるくする制御が自動的にできるからである。　たとえば，最初のパフォマンスピリオドはサービス量100までと定義し，その間は早いレスポンスを保証するパフォマンス目標を指定し，二番目のパフォマンスピリオドでは比較的悪いサービス条件を規定するパフォマンス目標を指定することにより，サービス量100をさかいにして，それより短いジョブと長いジョブに対し，異なるサービスレベルを与えることができる。　もちろんこの制御は，TSSのコマンドに対してもオンラインのトランザクションに対しても共通に適用される。

(2) スループットを高めるための制御

(a) システム資源使用率によるジョブ多重度の変更

SDMはCPU，チャネル，実記憶などのシステムの資源の使用率を計測し，それに応じて主記憶内ジョブ多重度を変更する。　資源の使用率が低ければ多重度をふやし，どれかひとつの資源にボトルネックがあれば多重度を減らす。　多重度の変更は各ドメインのTMPLの値を変更することによって行なう。

(b) システム資源使用率を反映したサービスの分配

スループットを高めるためには，使用率の低い資源を使うジョブを主記憶内におき，使用率が高くボトルネックとなっている資源を使うジョブをスワップアウトするように制御することが望ましい。　(1)で述べたドメイン内のジョブへの

サービス分配アルゴリズムは，ターンアラウンドについての条件のみにもとづいており，このような配慮はない。　実際のSDMの制御は，これに次のように偏り（スループットバイアスと呼ぶ）を加えて，システム全体の資源使用状況が反映されるようにしている。　スループットバイアスをかける場合のサービスの計算式は式(1)ではなく，次の式で表わされる。

$$\text{サービス} = A \times \text{CPU時間} + B \times \text{入出力回数} + C \times \text{実記憶量} \times \text{CPU時間} + f_1(\text{CPU時間}, \text{CPU使用率}) + f_2(\text{入出力回数}, \text{チャネル使用率}) + f_3(\text{実記憶量} \times \text{CPU時間}, \text{実記憶使用率}) \quad (4)$$

f_1, f_2, f_3 の各項は，システム全体の使用率の高い資源を多量に使用しているジョブのサービス率を見かけ上ふやし，スワップアウトされ易くするように働く。　この制御はユーザの指定したターンアラウンド条件をゆがめてしまう可能性があるので，スループットバイアスを適用するのは，パフォアンスグループに指定があったジョブのみである。

(c) 最低保証サービス量

SDMの制御の手段はスワップであるが，スワップはかなりの量の資源（CPU，チャネル，入出力装置）を消費し，そのオーバヘッドは無視できない。　以上に述べてきた論理で単純にスワップすべきジョブを決めて実施すると，スワップの振動がおこり，システムのスループットが大きく落ちる危険がある。　このため，一旦スワップインされたジョブは一定量のサービスをうけるまでスワップアウトされないよう制御している。　この最低保証サービス量は，設置先によってパフォマンスグループ中に定義される。

5.2 HOPE (Heuristic & Optimizing Evaluation Manager)

ユーザが指定したターンアラウンドに対する条件とシステム全体の資源の使用率の両者を考慮して最適なジョブの集合を主記憶上におくよう制御する点では，HOPEもSDMと同じである。

(1) ジョブの資源利用率

HOPEの制御の中心となる変数は，ジョブの資源利用率と呼ぶ量であり，次の式で表わされる。

$$\text{ジョブの資源利用率 } R = \frac{\text{CPU時間} + \text{入出力装置使用時間}}{\text{経過時間}} \quad (5)$$

上式の分子は，ジョブがシステムの資源を利用した時間の総計である。　ひとつのジョブに関する，CPUと入出力装置の並行動作，複数の入出力装置の並行動作を無視すれば，この時間はそのジョブを単独に（多重度1で）実行させた時の経過時間に等しい。　すなわち，多重度1の条件では $R=1$ である。　マルチジョブ環境下でのシステム内のすべてのジョブの資源利用率を合計したものは，多重処理の効果がどの程度かを表わす量であり，実効多重度と呼ばれる。

実効多重度　$I = \sum_j R_j$　　(6)

（R_j はジョブ j の資源利用率）

HOPE では，各ジョブに対して設置先とユーザが指定した資源利用率の目標値 r_j に実際の資源利用率 R_j が比例するように，システムの資源を配分する。実際に設置先が指定するのは資源利用率の逆数であり，これはサービス冗長度と呼ばれる。サービス冗長度は，一多重でそのジョブを実行した時と比べて，ジョブの実行時間が何倍に伸びたかを示す値である。

HOPE は定期的にシステム内のすべてのジョブの R_j を計算し，それにもとづいて規格化された資源利用率の目標値 r_j° を求め，R_j ができるだけ r_j° に一致するように制御する。

規格化された資源利用率の目標値　$r_j^\circ = \frac{r_j}{\sum_k r_k} \times I$　　(7)

(2) HOPE 優先権

ジョブの実際のサービス冗長度 $\frac{1}{R_j}$ とその目標値 $\frac{1}{r_j^\circ}$ との比を HOPE 優先権と呼ぶ。

HOPE優先権　$H_j = \frac{r_j^\circ}{R_j}$　　(8)

$H_j = 1$ はジョブが適切なサービスをうけていることを，$H_j \leqq 1$ はサービスに過不足があることを表わす。HOPEは，HOPE優先権1をふくむある区間Aを設け，その中にジョブをおくように制御する。あるジョブの HOPE 優先権が区間Aの下端 h_L より小さい時，HOPEはそのジョブがサービスを受け過ぎていると判断し，主記憶の外に追い出す。逆に HOPE 優先権が区間Aの上端 h_U より大きい時，HOPE はそのジョブを最優先で主記憶におくようにする。HOPE 優先権が区間Aの中にあるジョブについては，HOPE はスループットを高めることをねらいとして，以下のアルゴリズムで主記憶内におくジョブを選ぶ。

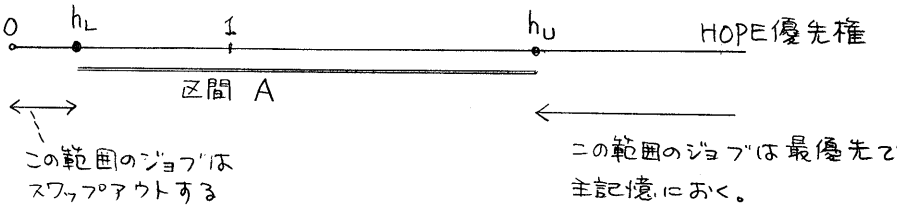

図5. HOPE 優先権

(3) CPU バウンドジョブと I/O バウンドジョブ

HOPE は，一定の CPU 時間を使用する間の入出力回数にもとづいて，ジョブを CPU バウンド，I/O バウンド，中間の3種類に分類する。そして，CPU に対しても入出力装置に対しても適量の負荷を与えるジョブの集合を主記憶上におくために，CPU バウンドジョブ，I/O バウンドジョブそれぞれの主記憶内多重度を次のように決める。

CPUバウンドジョブの多重度　I_{CPU} = CPU台数 / a_{CPU}　(9)

I/Oバウンドジョブの多重度　I_{IO} = 入出力装置台数 / a_{IO}　(10)

ここで a_{CPU}, a_{IO} はそれぞれCPUバウンドジョブ, I/Oバウンドジョブの CPU, 入出力装置に対する平均負荷率であり, システムで固定の定数が使われる。 入出力装置としては, システム全体で共用して使われる作業用ファイル用のディスク, ドラムを対象とする。

(4) 主記憶上におくジョブの選択

HOPE 優先権が区間A内に入るジョブの中から主記憶上におくジョブを選ぶプロセスを以下に示す。

(a) CPUバウンドジョブの中からHOPE 優先権が大きいものの順に I_{CPU} 個のジョブを選ぶ。

(b) I/Oバウンドジョブの中からHOPE 優先権が大きいものの順に I_{IO} 個のジョブを選ぶ。

(c) 中間の性格のジョブの中からHOPE優先権が大きいものの順にジョブを選ぶ。

ジョブの選択と並行して主記憶の割りあてが行なわれ, 使用可能な主記憶をすべて割りあてつくすと, 選択のプロセスは途中で終了する。 (a)(b)(c)をどの順序で実施するかは設置先の指定による。

(5) h_L のフィードバック制御

HOPE 優先権が区間Aにあるジョブの比率が小さい場合, スループットを高めるためのジョブ選択の自由度が少ないので, スループットが十分に上らない危険がある。 逆に区間Aにあるジョブの比率が大きい場合は, ターンアラウンドの安定性がそこなわれる。 そこでHOPEは, 次の情報をフィードバックさせて, 区間Aの下端 h_L を動的に変動させる。

- 処理を終えたジョブのターンアラウンドとその目標値との偏差
  (処理を終えたジョブのHOPE 優先権の1からの偏差)
- 実効多重度 I とその目標値との差

(6) 資源利用率目標値 r_j の変更

ジョブの資源利用時間の累積値が一定の値をこえたあとは, そのジョブに対する資源利用率の目標値 r_j を別の値に変えることができる。 資源利用時間のいき値と, 変更後の r_j は設置先によって指定される。 この制御はSDMのパフォマンスピリオドと同様のねらいをもっている。

6. おわりに

自動的な制御というテーマで現在のOSを概観したが, この分野の技術は実用化の途上にあるもの, 実用化されて間もなく評価の定まってないものが多く, 今後の研究と使用経験の反映が重要である。 また, SDM や HOPE に見られるような制御に関しては, 理論的な解析を試みるのも興味深い。

SDMとHOPEの制御方式について御教示いただいた富士通ソフトウェア事業部の新開慶武, 田淵治樹の両氏に感謝します。