# サービス関数による応答時間制御方式の下での計算機システム性能のボトルネック解析†

山本 彰†† 西垣 通††

サービス関数に基づいた応答時間制御を行っている計算機システムの解析的性能評価手法を提案する．サービス関数とは，各処理要求に与えるべき単位時間当りのサービス量（CPU 時間，入出力要求発行回数などの和）をシステムの負荷レベル（混み具合）の関数として表したものであり，柔軟な応答時間制御が可能であることから近年多くの商用システムで用いられている．しかし，従来の待ち行列網理論に基づいた解析的手法で取り扱うことができるスケジューリング方式は非常に限定的であり，上記の制御機構のモデル化は困難であった．本解析手法は，計算機システムのボトルネック資源に着目し，平均応答時間をその漸近解で近似するボトルネック解析手法に基づいている．本手法の特徴は，繰返し演算によりシステムの負荷レベルを予測し，その結果に基づいたボトルネック解析を行う点にある．実システムの測定データと本手法による予測結果を比較検討した結果，システム内に明確なボトルネック資源が存在する場合，本手法によりサービス関数制御を行っている計算機システムの大局的挙動を把握することが可能であるという結論を得た．

## 1. はじめに

多重プログラミング・システムの応答性要求をみたすための制御方式として，近年サービス関数を用いた方式が多用されている[1)~3)]．サービス関数とは，各バッチ・ジョブや TSS コマンドに与えるサービス供給量を時刻や負荷量の関数として表したものである．現在までに，いくつかのサービス関数が提案されているが，SRM[2)](System Resources Manager)，GRM[1)](General Resources Manager) などのスケジューラで用いられている Performance Objective (P.O.) はその代表的なものである．P.O. は，単位時間当りの供給サービス量，すなわち，サービス率を負荷量の関数として与えるものであり，負荷量の変動に適応した制御が可能である点において優れている．

これらのスケジューラの下で動作する計算機システムの性能予測を行うためには，上記の制御機構のモデル化が必須である．性能予測手法には，解析的手法とシミュレーション手法があるが，システムの大局的挙動を把握するためには，前者が適している．しかし，従来の待ち行列網理論に基づく解析的手法によりモデル化が可能なスケジューリング方式は，非常に限定されており（プロセッサ・シェアリング等の4方式）[7)]．上記制御機構のモデル化は困難である．

以上のような背景をもとに，筆者らは，サービス関数のもとでの計算機システムの性能予測を行うための解析的手法として「漸近モデル」[4),5)]を提案した．漸近モデルとは，利用率が高くボトルネック[6)]となる資源での待ちに着目し，平均応答時間をその漸近解で近似するものである[4)~6)]．しかし，これまでの報告[4)]では，解析の対象は各処理要求に与えられるサービス率の配分比が負荷量の変動によらず一定であるようなサービス関数が指定された場合に限られていた（一般的に，サービス率の配分比は負荷量により変化する）．また，処理の途中で処理要求が対応づけられているサービス関数の変更が行われる場合があるが，この機能（サービス関数変更機能）も取り扱うことができなかった．

本論文では，筆者らが先に提案した上記手法をさらに拡張して，一般的なサービス関数制御のモデル化を可能とした性能予測手法を提案する．本手法は，繰返し計算により，システムの負荷レベル（混雑度）を予測し，それに基づいた平均応答時間を求めることを特徴とする．また，サービス関数変更機能もモデル化の対象とする．最後に，実システムの実測データにより本手法の精度検証を行う．

## 2. サービス関数による応答時間制御と漸近モデル

本論文における計算機システムのモデルを**図1**に示す．図に示したように，計算機システム・モデルを構成する資源は，CPU，I/O (Input/Output) 装置，メモ

† Bottleneck Analysis of a Computer System with Service-Function-Driven Scheduling Disciplines by AKIRA YAMAMOTO and TOHRU NISHIGAKI (Systems Development Laboratory, Hitachi, Ltd.).

†† (株)日立製作所システム開発研究所

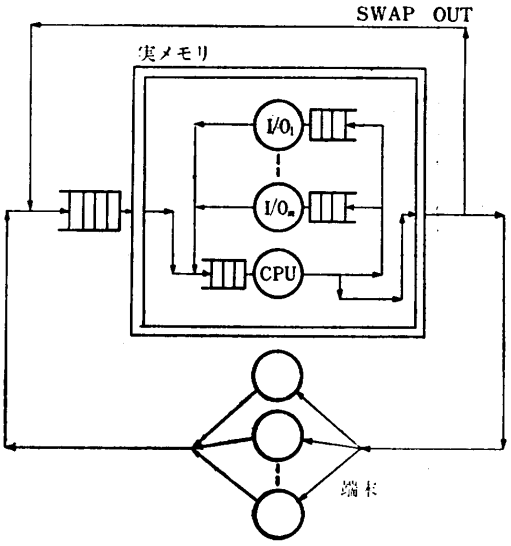


**図 1** 本論文における計算機システム・モデル
Fig. 1 The computer system model.

リ，端末である．各資源を割り当てる単位を「トランザクション」とよぶが，これは現実システムのバッチ・ジョブ，TSS コマンドに対応する．バッチ・ジョブについては，ジョブ・スケジューラ（イニシェータ）を思考時間＝0の端末と考える．トランザクションは，思考時間が終了するとともに発生し，実メモリが割り当てられ，その後 CPU や I/O 装置を複数回使用して完了する．記憶装置管理方式は仮想記憶方式を仮定する．総実メモリ容量を$V$，トランザクションの平均ワーキング・セット・サイズを $W$ とすると，その最大多重度$S$は，平均的には $V/W$ で表すことができる．思考時間中でない端末台数が$S$を越えると，越えた分だけのトランザクションが，主記憶待ちとなる．トランザクションが発生してから終了するまでに使用した資源の使用量をサービス量とよぶ．具体的には，CPU 使用時間，I/O 発行回数などに，それぞれ重みづけ係数をかけて和をとったものである[1),2)]．本

**表 1** クラス $i$ のトランザクションに関する記号の定義
Table 1 Definition of the symbols with respect to the transaction in the class $i$.

| 用　語 | 記号 | 用　語 | 記号 |
|---|---|---|---|
| 平均応答時間 | $T_i$ | トランザクション発生端末数 | $N_i$ |
| 平均実メモリ占有時間 | $R_i$ | 待ちなしの平均応答時間 | $R_{i0}$ |
| 平均思考時間 | $z_i$ | 処理完了に必要なサービス量 | $C_i$ |
| プロセッサ$p$の平均使用時間 | $d_{ip}$ | 最大多重度（実メモリ量/平均ワーキング・セット・サイズ） | $S_i$ |

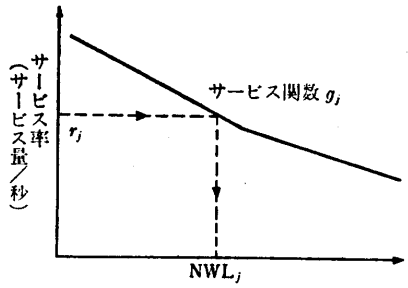


**図 2** サービス関数 $g_j$ に基づいてサービスを受けているトランザクションの NWL
Fig. 2 The NWL of the transaction associated with the service function $g_j$.

論文では，CPU, I/O 装置はプロセッサとよび，等価に取り扱う．また，各資源の使用特性が似たトランザクションの集合をクラスとよぶ．次に，記号の定義を行う．クラスの集合，プロセッサの集合をそれぞれ，I, P とする．クラス $i$ のトランザクションに関する記号の定義を**表 1** に示す．ただし，$S_i$ は $V$ をクラス $i$ のトランザクションの平均ワーキング・セット・サイズで割ったものである．

サービス関数 (P.O.) は，**図 2** に示すようにシステムの混雑度を示す負荷レベルとトランザクションのサービス率(単位時間当りのサービス量)との間の関数である．サービス関数 $g_j$ に基づいてサービスを受けているトランザクションに与えるサービス率を $r_j$ とする（トランザクションに与えるサービス率はどのサービス関数に基づいてサービスを受けているかによって定まり，属するクラスにはよらない）．サービス関数のセットを$J$とする．$g_j$ の逆関数を $g_j^{-1}$ とするとき，次式より定義される値を $g_j$ に基づいてサービスを受けているトランザクションの NWL (Normalized Workload Level)，すなわち，$NWL_j$ とよぶ．

$$NWL_j = g_j^{-1}(r_j) \tag{2.1}$$

制御目標は，

$$\overline{NWL} \leftarrow NWL_j \tag{2.2}$$

である．ただしここで，$\overline{NWL}$ はシステム内に存在する全トランザクションの NWL の平均値であり，これをシステム負荷レベルとよぶ．

制御目標が，達成されたとき，クラス $i$ のトランザクションが $g_j$ のサービスを受けるとすると，$T_i, C_i, r_j$ の間には，次の関係式が成立する．

$$T_i = C_i/r_j \quad (\forall i \in I, \forall j \in J) \tag{2.3}$$

なお，一般に，トランザクションに対応するサービス関数は唯一でなく，供給したサービス量に従って変

化する．これをサービス関数変更機能とよぶ．この機能を考慮したとき，クラス $i$ のトランザクションが $g_j$ に基づいて受けるサービス量，および，その時間をそれぞれ，$C_{ij}, T_{ij}$ とすると次式が成立する．

$$T_i=\sum_j T_{ij}=\sum_j C_{ij}/r_j \quad (\forall i\in I, \forall i\in J) \qquad (2.4)$$

サービス関数は，通常図2のように傾きが負の折れ線であるが，例外として傾きが0の直線の場合がある．ここでは，これをとくに $g_A$ とよぶ．NWL の定義から明らかなように，$g_A$ に基づいてサービスを受けているトランザクションの NWL は存在せず，たんに他のトランザクションより常に高い優先順位が与えられる．

以上がサービス関数制御方式の内容であるが，ある種のサービス関数のもとでの性能解析は，筆者らがすでに提案した漸近モデルを用いて解析することができた．漸近モデルの詳細は参考文献 4),5) に記したので省略し，ここでは主要な結果と残された問題について述べる．一般にプロセッサ，メモリの利用率を $u_p$, $u_{mem}$ とすると，$u_p, u_{mem}$ は次式を満たす．

$$u_p=\sum_i N_i d_{ip}/(T_i+z_i) \quad (\forall p\in P) \qquad (2.5)$$

$$u_{mem}=\sum_i N_i(R_i/S_i)/(T_i+z_i) \qquad (2.6)$$

漸近モデルとは，平均応答時間を端末数の関数として表したときに，平均応答時間が有する漸近解を近似解とする（仮定1）ものである．漸近解においては，ボトルネック資源がシステム内に存在しないときには，待ち時間を評価しない．また，ボトルネック資源が存在するときには，漸近解はその資源の利用率を100% であるとすることにより求めることができる．したがって次式が成立する．

ボトルネック資源なし：$T_i=R_{i0}$ $(\forall i\in I)$ (2.7)

プロセッサ $p$・ネック：$1=\sum_i N_i d_{ip}/(T_i+z_i)$ (2.8)

メモリ・ネック：$1=\sum_i N_i(R_{i0}/S_i)/(T_i+z_i)$ (2.9)

(2.7)式より，ボトルネック資源が存在しないときの $T_i$ は明らかであるため，以後は，ボトルネック資源が存在するものとして議論を進める．

いま，**図3**に示すサービス関数が指定されたとする．この場合，次式で定義される値，すなわち，各トランザクションに与えるべきサービス率の配分比 $q_j$ は $\overline{\text{NWL}}$ によらず一定になる．ここで，$g_k$ は $J'$ に含まれるサービス関数の中から適当に選ぶとする．

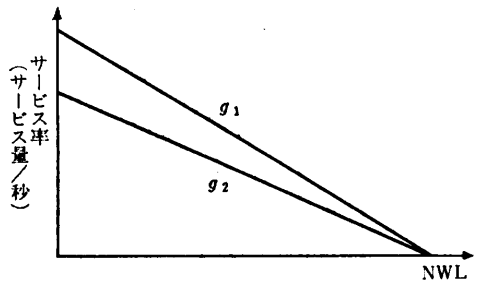


**図3** $g_1(\overline{\text{NWL}})/g_2(\overline{\text{NWL}})$ が一定値をとるケース
Fig. 3 The case in which $g_1(\overline{\text{NWL}})/g_2(\overline{\text{NWL}})$ is constant

$$q_j=g_j(\overline{\text{NWL}})/g_k(\overline{\text{NWL}}) \quad (\forall j\in J') \qquad (2.10)$$

ここで，$J'$ はサービス関数 A を除くサービス関数の集合を表す．このとき，サービス関数変更機能が動作しないという前提に立ち，定常状態を仮定すると次式が成立する．

$$T_i=C_i/r_j$$
$$=C_i/(q_j\omega) \quad (\because \omega=g_k(\overline{\text{NWL}}))(\forall i\in I, \forall j\in J') \qquad (2.11)$$

したがって，プロセッサ $p$・ネックの場合(2.11)式を(2.8)式に代入し，未知数 $\omega$ を得ることにより $T_i$ を得ることができる．同様に，メモリ・ネックのときには，(2.11)式を(2.9)式に代入することにより $T_i$ を算出できる．

以上の方式[4]については，すでに報告したが，さらに残された問題として次の2点があげられる．

（1） 一般的に，サービス率の配分比 $q_j$ は $\overline{\text{NWL}}$ によって変化する．このとき解を得ることができない．

（2） サービス関数変更機能，サービス関数Aに対する考慮がない．

これらを解決する手法について次章で述べる．

## 3. 一般的なサービス関数制御の取扱い

本章では，前章で述べた問題を解決するために，これまで筆者らが報告した手法を拡張し一般的なサービス関数制御方式の取扱いを可能にした手法について述べる．

前章であげた問題のうち(2)については，比較的単純に解決することが可能であるため，まず，これについて述べる．サービス関数変更機能を考慮すると，(2.3)式に代って(2.4)式が成立する．したがって，次式が成立する．

$$T_i=\sum_j T_{ij}$$
$$=\sum_j C_{ij}/(q_j\omega) \quad (\forall i\in I, \forall j\in J') \qquad (3.1)$$

(2.11)式の代りに(3.1)式を用いることにより $T_i$ を得ることができる．ここでも，(2.11)式を導いたときと同様に定常状態を仮定している．各トランザクションがサービス関数を遷移していく速さは十分遅いとして，トランザクションがある一つのサービス関数に基づいてサービスを受けている間に定常状態になりうると仮定する．(実際，(3.1)式が適用されるときにはボトルネック資源が必ず存在し，システム内に存在するトランザクション数は多いので，ある特定のトランザクションが $g_1$ に基づいてサービスを受けていようと，$g_2$ に基づいてサービスを受けていようと，システム全体に与える影響は少ない．したがって，システム全体としては，定常状態になりうるという仮定は成立しやすい．) これより，サービス関数変更機能を考慮した場合でも，1変数 ($\omega$) だけで，$T_i$ を表すことが可能となる．

次にサービス関数$A$を考慮した場合について述べる．$g_A$ に基づいてサービスを受けているトランザクションは，他のトランザクションに比べ高い優先順位が与えられる．したがってこの制御方式のモデル化を行う必要があるわけであるが，これについての詳細は参考文献4),5) を参照されたい．ここでは結論のみを述べる．優先スケジュールを割込み優先権出直し基準とした(仮定2)ときには，漸近解においては，ボトルネック資源に対し待ちが生ずるのはその優先順位の低いトランザクションのみである．新たに，次の仮定を設ける．

[仮定3] 各クラスのトランザクションに関して次式が成立する．

$$d'_{ip}/\sum_p d'_{ip}=d_{ip}/R_{i0} \quad (\forall i\in I, \forall p\in P) \qquad (3.2)$$

$d'_{ip}$ はクラス $i$ のトランザクションがある時点までにプロセッサ $p$ を使用した時間である．(3.2)式の物理的意味は，各トランザクションが各プロセッサを均等の割合で使用していくということである．

以上によりサービス関数$A$を考慮すると，(3.1)式に代り(3.3)式が成立する．

$$\begin{aligned} T_i&=T_{iA}+\sum_j Tij \\ &=(C_{iA}/C_i)R_{i0}+\sum_j C_{ij}/(q_j\omega) \\ &\quad (\forall i\in I, \forall j\in J') \end{aligned} \qquad (3.3)$$

(3.3)式と(2.8)式，あるいは，(3.3)式と(2.9)式を用いることにより $T_i$ を得ることができる．

しかし，(3.3)式が得られるのは，サービス率の配分比 $q_j$ が既知の場合だけである．一般には，$\overline{\mathrm{NWL}}$ によって $q_j$ が変化するため，(3.3)式を得ることができない．次に $q_j$ が $\overline{\mathrm{NWL}}$ によって変化する一般的な場合について論じる．

サービス関数 (P.O.) における制御目標は，$\overline{\mathrm{NWL}}$ が確定したとき，$g_j$ に基づいてサービスを受けているトランザクションのサービス率を，$g_j(\overline{\mathrm{NWL}})$ に等しくすることである．したがって，(2.10)式により，$\overline{\mathrm{NWL}}$ を適当に決めることにより，$q_j$ を定めることが可能となる．いま，$Z_i, S_i, d_{ip}, R_{i0}, C_i, C_{ij}, N_i$ のパラメータをまとめて $\boldsymbol{L}, q_j(\forall j\in J')$ を $\boldsymbol{q}$ とそれぞれベクトルで表すと次式が成立する．

$$T_{ij}=f_{ij}(\boldsymbol{L},\boldsymbol{q}) \quad (\forall i\in I, \forall j\in J') \qquad (3.4)$$

(3.4)式における $f_{ij}$ は，$\boldsymbol{L}$ と $\boldsymbol{q}$ が与えられたときに，$T_{ij}$ を得る関数である．これは(3.3), (2.8), (2.9)式を変形した形に等しい．ただし，クラス $i$ のトランザクションが $g_A$ に基づいてサービスを受けている時間 $T_{iA}$ は(3.3)式から明らかなように $\boldsymbol{q}$ に依らず一定である．

$T_{ij}(\forall j\in J')$ が得られたとき，$\mathrm{NWL}_j$, $\overline{\mathrm{NWL}}$ の定義から明らかに，次式により，それぞれの値を得ることができる．

$$\begin{aligned} \mathrm{NWL}_j&=g_j^{-1}(C_{i_1j}/T_{i_1j}) \\ &=g_j^{-1}(C_{i_2j}/T_{i_2j})=\cdots \quad (\forall j\in J') \end{aligned} \qquad (3.5)$$

$\overline{\mathrm{NWL}}$ は，システム内のトランザクションの $\mathrm{NWL}_j$ の平均であるため，サービス関数 $g_j$ に基づいてサービスを受けているトランザクション数を知る必要がある．いま，クラス $i$ のトランザクションのなかで，サービス関数 $g_j$ に基づいてサービスを受けているトランザクション数を $N'_{ij}$ とすると，$N'_{ij}$ はリトルの公式より次式から得られる．

$$N'_{ij}=N_i\cdot T_{ij}/(T_i+z_i) \quad (\forall i\in I, \forall j\in J') \qquad (3.6)$$

$$(\because N'_{ij}/T_{ij}=N_i/(T_i+z_i))$$

これより，$\overline{\mathrm{NWL}}$ は次式から算出可能となる．

$$\overline{\mathrm{NWL}}=(\sum_i\sum_j N'_{ij}\cdot \mathrm{NWL}_j)/(\sum_i\sum_j N'_{ij}) \qquad (3.7)$$

各トランザクションに与えるサービス率はクラスによらず，サービス関数により決定されるため(3.5)式の成立は明らかである．$g_A$ に基づいてサービスを受けているトランザクションの NWL は存在しないため，(3.5)～(3.7)式には関係しない．

したがって，$\overline{\mathrm{NWL}}$ を適当に仮定し，(2.10)式より $\boldsymbol{q}$ を定め，(3.4)～(3.7)式により $\overline{\mathrm{NWL}}$ を算出したとき，これが最初仮定した $\overline{\mathrm{NWL}}$ と等しければ，制御目標を満たす $\overline{\mathrm{NWL}}$ が得られたことになる．これ

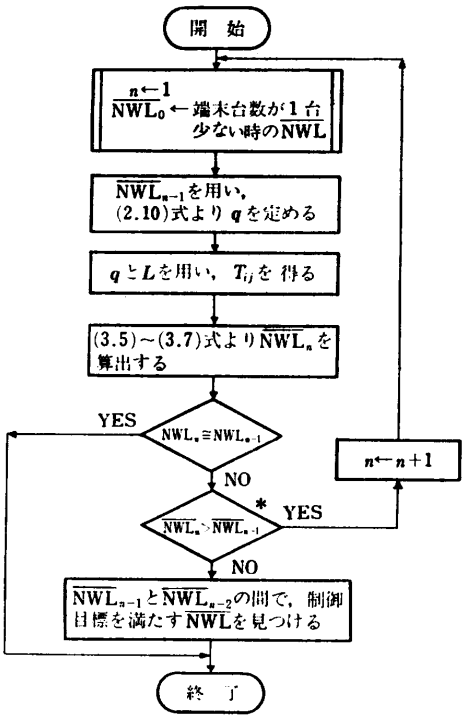


**図 4** 平均応答時間を得るための繰返し演算
Fig. 4 The iterative algorithm to obtain mean response time.

より，$T_{ij}$ を算出することが可能となる．

**図4**は，制御目標を満たす $\overline{NWL}$ を得るための具体的なアルゴリズムである．$\overline{NWL}_n$ を，$n$回めの繰返し演算で得られた $\overline{NWL}$ であるとする．$\overline{NWL}_n$ の初期値，すなわち，$\overline{NWL}_0$ はトランザクションの総数が1少ないときの $\overline{NWL}$ を用いることにした．ただし，この方法を用いると$N$個のトランザクション数のときの解を求めるために，1個から$N-1$個までのトランザクション数のときの解をすべて求める必要がある．

図4に示した繰返し演算においては，$T_{ij}$ を得るのに必要な二つのベクトル，$\boldsymbol{L}$ と $\boldsymbol{q}$ のうち，変化するのは $\boldsymbol{q}$ のみで，$\boldsymbol{L}$ は常に一定である．$T_{iA}$ は，(3.3)式より，明らかに，$\boldsymbol{L}$ によってのみ定まるため，繰返し演算の過程では常に一定の値をとる．また，$g_A$ に基づいてサービスを受けているトランザクションにはNWLが存在しないということから，これらのトランザクションは制御目標を満たす $\overline{NWL}$ を得るという目的には，まったく関与しないことになる．したがって，ここでは，$g_A$，および，これらに基づいてサービスを受けているトランザクションを考慮する必要はない．また，$\boldsymbol{q}$ が $\overline{NWL}$ によらず一定であるときには，二つのベクトル $(\boldsymbol{q}, \boldsymbol{L})$ がどちらも変化しないことになる．したがって，この場合は $\overline{NWL}_1=\overline{NWL}_2$ となり，制御目標を満たす $\overline{NWL}$ が，提案したアルゴリズムによって得られることは明らかである．

なお，$\boldsymbol{q}$ が $\overline{NWL}$ によって変化する一般的なサービス関数が指定された場合でも，提案したアルゴリズムによって，制御目標を満たす $\overline{NWL}$ を得ることが可能である．この証明は，付録で行う．

## 4. 実システムによる精度検証

実システムの測定データと本モデルによる予測結果を比較することにより，モデルの精度検証を行った．実システムは CPU が HITAC M 200 H，オペレーティング・システムが VOS 3 (Virtual storage Operating System 3) メモリ容量 16 M bytes, I/O 装置として HITAC 8575 型磁気ドラム装置2台，磁気ディスク装置74台により構成される．磁気ディスク装置のうちわけは，HITAC 8595 型，8589-1 型，8589-11 型がそれぞれ 20, 6, 48 台である．システムに投入される負荷は，バッチ・ジョブと TSS コマンドである．検証に用いたデータは，1日の実稼動データである．TSS コマンドに関しては，平均思考時間，平均アクティブ端末台数は不明であり，平均システム内多重度のみ実測データとして得られた．漸近モデルにおいては，ボトルネック資源の利用率を100%として，平均応答時間を得るわけだが，TSS コマンドが占める各資源の利用率は，そのスループットと平均使用時間の種で表すことができる．さらに TSS コマンドのスループットについては次式が成立する．

$$\lambda_{TSS}=N_{TSS}/(T_{TSS}+Z_{TSS})$$
$$=N'_{TSS}/T_{TSS} \qquad (4.1)$$

(TSS コマンドのスループット，平均応答時間，平均思考時間，平均アクティブ端末台数，平均システム内多重度をそれぞれ，$\lambda_{TSS}, T_{TSS}, Z_{TSS}, N_{TSS}, N'_{TSS}$ とす

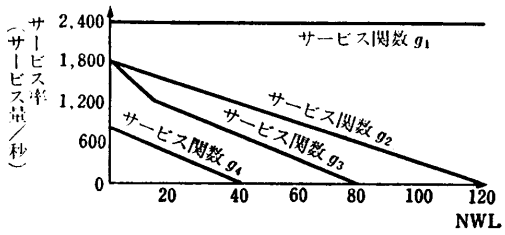


**図 5** 精度検証用の計算機システムで指定されているサービス関数
Fig. 5 The specified service functions in the system to validate this model.

表 2 平均応答時間の評価結果
Table 2 Validation with respect to mean response time

| 項目 / クラス | 予測結果 | 実測結果 | 相対誤差 |
|---|---|---|---|
| *TSS コマンド | 7.07 秒* | 7.88 秒* | −10.3% |
| バッチ・ジョブ | 242 秒 | 229 秒 | +5.7% |
| 両クラス平均 | 14.8 秒 | 15.2 秒 | −2.7% |

* TSS コマンドの平均応答時間が比較的大きいのは，長大な TSS コマンドが多かったためである．

る．)

したがって，ここでは，平均思考時間，平均アクティブ端末台数の代りに平均システム内多重度を用いて，TSS コマンドのモデル化を行った．

**図 5** に実測システムで定義されているサービス関数を示す．TSS コマンドは，まずサービス関数 $g_1$ に基づくサービスを受け，供給サービス量の増加に伴いサービス関数 $g_2$, $g_3$, $g_4$ に基づいたサービスを受ける．一方，バッチ・ジョブはまずサービス関数 $g_2$ に基づくサービスを受け，次にサービス関数 $g_3$ のサービスを受ける．ただし，それぞれの平均サービス量を考慮して，モデルにおいては，TSS コマンドはサービス関数 $g_1$ と $g_2$ に基づいたサービスを受けるようにし，バッチ・ジョブはサービス関数 $g_2$ のサービスを受けるようにした．

実測システムにおいては，ボトルネック資源は CPU でその利用率は 97% であった．**表 2** に，TSS コマンド，バッチ・ジョブ，両者の平均値に関する平均応答時間の評価結果を示す．ここで，TSS コマンドの平均応答時間が大きいのは，翻訳・実行型のような長大なコマンドが多いためである．一般に，長大なコマンドが多いシステムのほうが，各トランザクションの NWL を等しくするという制御目標は実現されやすいと考えられる．予測結果は，実測結果に比べて，TSS コマンドを過小評価し，バッチ・ジョブを過大評価している．また，両者の平均値では 3% 程度過小評価しているが，これは CPU の利用率が実測値では 97% であるのに対し，漸近モデルでは 100% として近似解を求めるためである．したがって，本モデルにおいてバッチ・ジョブと TSS コマンドの間のサービス率の配分比を正しく予測できたと仮定すると，どちらの平均応答時間も，それぞれ 3 %ずつ過小評価するはずである．このため，本モデルにおいては，TSS コマンドに与えるサービス率の配分比を 7% 過大評価していることになる（バッチ・ジョブの場合は 7% 過小評価している）．おもな原因は，モデル自体に起因して生ずる誤差以外に，データが 1 日の実稼動システムの平均データであるため，時間により稼動ジョブのばらつきが大きいという点があげられる．しかし，7% 程度の誤差であれば，システムの大局的挙動を把握するのには十分であると考えられる．これは，システム内に明確なボトルネック資源が存在するときには，(本システムの場合は CPU の利用率が 97%) 本モデルにより，サービス関数制御を行っている計算機システムの性能予測が可能であることを示している．

## 5. おわりに

サービス関数による応答時間制御を行っている計算機システムの解析的性能予測手法を提案した．本手法は，計算機システム内のボトルネック資源に着目し，平均応答時間が有する漸近解を近似解とするものである．従来，筆者らが提案したモデルでは特殊なサービス関数が指定された場合のみ，解析が可能であったが，今回提案した手法においては，繰返し演算を用いて，システム負荷レベルを予測することにより，一般的なサービス関数制御の取扱いを可能にした．また，サービス関数変更機能を考慮した．さらに，実測データにより本手法の精度検証を行った．システム内に，ボトルネック資源が存在する場合，本手法により，サービス関数制御方式を用いる計算機システムの大局的挙動を把握することが可能であることを確認した．

今後の課題としては次の二つの項目が考えられる．

（1） 各種のデータにより，本手法が有効な範囲を把握していく必要がある．

（2） 提案したアルゴリズムにより制御目標を満たす実用的な解を得ることはできるが，理論的に複数の解が存在する場合については，検討の余地を残している点がある．このような場合でも，実用的な解は得られるため，ほとんど問題ないと考えられるが，今後この問題を検討する必要がある．

**謝辞** 本研究の機会を与えて下さった(株)日立製作所取締役・コンピュータ事業本部三浦武雄本部長，同社システム開発研究所川崎淳所長，有益な助言を与えて下さった同システム開発研究所大町一彦主任研究員，また，精度検証用のデータ収集にご協力いただいた同社ソフトウェア工場多田博光氏の諸氏に深く感謝いたします．

## 参考文献

1) Nishigaki, T. et al.: An Experiment on the

General Resources Manager in Multiprogrammed Computer Systems, *JIP*, Vol. 1, No. 4, pp. 187-192 (1979).
2) Linch, H. W. and Page, J. B. : The OS/VS 2 Release 2 System Resources Manager, *IBM Syst. J.*, Vol. 13, No. 4, pp. 274-291 (1974).
3) Bernstein, A. J. and Sharp, J. C. : A Policy-Driven Scheduler for a Time-Sharing System, *Comm. ACM*, Vol. 14, No. 2, pp. 74-78 (1971).
4) Nishigaki, T. et al. : An Approach to the GRM Performance Analysis by Asymptotic Approximation, *JIP*, Vol. 3, No. 2, pp. 59-67 (1980).
5) 西垣，山本：資源割り当て優先度のある多重プログラミングシステムのボトルネック解析，情報処理学会論文誌，Vol. 23, No. 5, pp. 562-569 (1982).
6) Muntz, R. R. et al. : Asymptotic Properties of Closed Queueing Network Models, *Proc. 8th Annu. Princeton Conf. on Inf. Sci. Syst.*, pp. 348-353 (1974).
7) Baskett, F. et al. : Open, Closed, and Mixed Networks of Queues with Different Classes of Customers, *J. ACM*, Vol. 22, No. 2, pp. 248-260 (1975).

## 付　録

付録では，$\boldsymbol{q}$ が $\overline{\mathrm{NWL}}$ によって変化する一般的なサービス関数に対しても，図4に示したアルゴリズムによって，制御目標を満たす $\overline{\mathrm{NWL}}$ を得ることが可能であることを証明する．

まず，$\overline{\mathrm{NWL}}_0<\overline{\mathrm{NWL}}_1$ を証明する．端末台数が $N$ 台のときの $\boldsymbol{L}$ を $\boldsymbol{L}_N$ とする．$\overline{\mathrm{NWL}}_0$ は，端末台数が $N-1$ 台のときの制御目標を満たす解である．したがって，$\overline{\mathrm{NWL}}$ を $\overline{\mathrm{NWL}}_0$ として，この場合の $\boldsymbol{q}$ を用い，端末台数が $N-1$ 台のときの $T_{ij}$ を得，(3.5)～(3.7)式より $\overline{\mathrm{NWL}}$ を計算すると，これが $\overline{\mathrm{NWL}}_0$に等しくなるということになる．一方，$\overline{\mathrm{NWL}}_1$ は，同じ $\boldsymbol{q}$ を用い，端末台数が $N$ 台の $\overline{\mathrm{NWL}}$ を算出したものである．したがって，両者を算出するときの各トランザクションに与えるサービス率の配分比は同一である．

いま，$\boldsymbol{L}_n>\boldsymbol{L}_{n-1}$ であるから，明らかに次式が成立する（∵サービス率の配分比が同じであれば，負荷量が大きいほうが，平均応答時間は大きくなる）．

$$f_{ij}(\boldsymbol{L}_{n-1},\boldsymbol{q})<f_{ij}(\boldsymbol{L}_n,\boldsymbol{q}) \quad (\forall i\in I,\forall j\in J') \qquad (付.1)$$

(付.1)，(3.4)～(3.7)式およびサービス関数の傾きが負であることから，明らかに，$\overline{\mathrm{NWL}}_0<\overline{\mathrm{NWL}}_1$ が成立する．

$\overline{\mathrm{NWL}}_0<\overline{\mathrm{NWL}}_1$ が成立するため，図4に示したアルゴリズムの中で＊を付けた分岐において，$n=1$ の場合には，YES が成立する．したがって，次は $n=2$ とし，$n=1$ のときと同様の処理に入ることになる．これより，図4に示したアルゴリズムで，制御目標を満足する $\overline{\mathrm{NWL}}$ を見つけることが不可能となるのは，以下に示す二つの事象が発生した場合に限定される．

(1) $\overline{\mathrm{NWL}}_n>\overline{\mathrm{NWL}}_{n-1}$ $(n=1,2,\cdots,\infty)$ (付.2)

かつ，$\lim_{n\to\infty}(\overline{\mathrm{NWL}}_n-\overline{\mathrm{NWL}}_{n-1})\neq 0$ (付.3)

が成立する．

(2) $\overline{\mathrm{NWL}}_n>\overline{\mathrm{NWL}}_{n-1}$ $(n=1,2,\cdots,m-1)$ (付.4)

かつ，$\overline{\mathrm{NWL}}_m<\overline{\mathrm{NWL}}_{m-1}$ (付.5)

が成立するにもかかわらず，$\overline{\mathrm{NWL}}_{m-1}$ と $\overline{\mathrm{NWL}}_{m-2}$ の間に制御目標を満たす解が存在しない．

以上(1)，(2)に示した事象が発生しなければ，図4に示したアルゴリズムにより，制御目標を満たす $\overline{\mathrm{NWL}}$ が得られることになる．以下この証明を行う．

まず，(1)の場合について述べる．(1)に示した事象が発生し，(付.2)，(付.3)式が同時に成立すると，次式が必ず成立する．

$$\lim_{n\to\infty}\overline{\mathrm{NWL}}_n\to\infty \qquad (付.6)$$

$\overline{\mathrm{NWL}}$ を仮定し，$\boldsymbol{q}$ を定め，$T_{ij}$ を得，(3.5)式より，$\mathrm{NWL}_j$ を得たとき，サービス関数の形から考えて $\mathrm{NWL}_j$ が無限大に発散することはない．したがって，(3.6)～(3.7)式より，$\overline{\mathrm{NWL}}$ が無限大に発散することがないことも明らかである．これより，(付.2)式が成立したときには，(付.3)式が成立することはなく次式が成立することになる．

$$\lim_{n\to\infty}(\overline{\mathrm{NWL}}_n-\overline{\mathrm{NWL}}_{n-1})\to 0 \qquad (付.7)$$

したがって，(1)に示した事象が発生することはなく，$n$ を十分大きくとれば，$\overline{\mathrm{NWL}}_n\cong\overline{\mathrm{NWL}}_{n-1}$ となるため，制御目標を満たす $\overline{\mathrm{NWL}}$ を得ることが可能となる．

次に，(付.4)(付.5)式が成立するときには，$\overline{\mathrm{NWL}}_{m-2}$ と $\overline{\mathrm{NWL}}_{m-1}$ の間には少なくとも一つは，制御目標を満たす解が存在し，(2)で示した事象が発生することはないことを示す．

ここで，$\overline{\mathrm{NWL}}$ を仮定したとき，同一負荷量の下

で，(2.10)，(3.4)～(3.7)式に従い，再び $\overline{\mathrm{NWL}}$ を算出する関数を考え，これを $h$ と呼ぶ．この場合，$q$ は $\overline{\mathrm{NWL}}$ に従って，連続に変化するため，$h$ も $\overline{\mathrm{NWL}}$ に対して連続な関数となるはずである．いま，制御目標を満たす $\overline{\mathrm{NWL}}$ を $\overline{\mathrm{NWL}}_x$ とすると $\overline{\mathrm{NWL}}_x$ は次式を満たす必要がある．

$$\overline{\mathrm{NWL}}_x = h(\overline{\mathrm{NWL}}_x) \qquad \text{(付.8)}$$

したがって，$\overline{\mathrm{NWL}}_{m-1}$ と $\overline{\mathrm{NWL}}_{m-2}$ の間に制御目標を満たす $\overline{\mathrm{NWL}}$ が存在することは，次に定義する関数 $h'$ が，$\overline{\mathrm{NWL}}_{m-1}$ と $\overline{\mathrm{NWL}}_{m-2}$ の間で根をもつということである．

$$h'(\overline{\mathrm{NWL}}) = h(\overline{\mathrm{NWL}}) - \overline{\mathrm{NWL}} \qquad \text{(付.9)}$$

いま，$\overline{\mathrm{NWL}}_n = h(\overline{\mathrm{NWL}}_{n-1})$ であるため，(付.4)，(付.5)式が成立すると，次式が成立する．

$$h'(\overline{\mathrm{NWL}}_{m-2}) = h(\overline{\mathrm{NWL}}_{m-2}) - \overline{\mathrm{NWL}}_{m-2} > 0 \qquad \text{(付.10)}$$

$$h'(\overline{\mathrm{NWL}}_{m-1}) = h(\overline{\mathrm{NWL}}_{m-1}) - \overline{\mathrm{NWL}}_{m-1} < 0 \qquad \text{(付.11)}$$

$h'$ は明らかに $\overline{\mathrm{NWL}}$ に関して連続な関数であるため，(付.10)，(付.11)式と中間値の定理より，$h'(\overline{\mathrm{NWL}})$ は，$\overline{\mathrm{NWL}}_{m-2}$ と $\overline{\mathrm{NWL}}_{m-1}$ の間に少なくとも一つの根をもつことがわかる．したがって，2分検索法などを用いることにより，制御目標を満たす $\overline{\mathrm{NWL}}$（複数存在する場合にはそのなかの一つ）を見いだすことができる．以上により，(2)に示した事象が発生することがないということも証明できた．

ただし，制御目標を満たす解が複数存在するとき，これらをすべて見いだす手法は現在のところわかっていないため，今後の課題としたい．しかし，筆者らの実験例によれば，解が複数存在する場合はなかった．また，実用的観点から見ても，解が一つ見いだすことができれば十分であると考えられる．

以上により，図4に示したアルゴリズムにより，制御目標を満たす $\overline{\mathrm{NWL}}$ を得ることができることを証明できた．

(昭和57年11月2日受付)
(昭和58年3月11日採録)