計算機システムの性能評価における モデリングと計測

宮崎正俊　小畑征二郎　松沢　茂
（東北大学大型計算機センター）

1. はじめに

計算機システムの性能評価における主な手法には，解析的手法，モニタリング（計測），シミュレーションなどがあり[1),(2)]，稼動記録の解析も手近な方法としては重宝である。これらの手法にはそれぞれ利点と欠点があるが，従来より最も多く検討されてきたのは解析的手法であり，種々のモデルが考案され解析されている。

モデルは主に待ち行列の形で扱われていることが多いが，このモデルは対象とする計算機システムによって異なる。たとえば，バッチ処理システムとタイムシェアリング・システム（オンライン・システム）とではモデルの組立て方が本質的に異なるのは明らかである。これは性能評価の尺度が違うからである。バッチ処理においては，スループットやターン・アラウンド・タイムが評価尺度として重要である。オンライン・サービスにおいては応答時間が問題にされる。

これらの評価尺度に影響を与える要素として，システム構成，主記憶管理，補助記憶管理，多重プログラミング，制御方式（スケジューリング），プログラム動作などが挙げられる。これらの評価要素のどれに着目するかによって，やはりモデルは異なってくる。

解析的手法の利点は性能を予測できることである。シミュレーションでも性能予測は可能であるが，この手法は一般に大がかりになる傾向がある。解析的手法において問題になるのはモデルの適合性である。システムを詳細に表現しようとすれば，モデルが複雑になり解析が困難になる傾向がある。そこで，一般には適当な仮定を設けてモデルを単純にすることを試みるわけであるが，単純化が行き過ぎると，モデルが実際のシステムを反映しなくなり，適合性がなくなってしまう。そこで，単純で扱いやすく拡張や応用に対する柔軟性もあり，しかもある程度の適合性をもったモデルの検討が必要になってくる。

モデルの適合性を議論するには，当然のことながら現実のシステムから得られた生のデータとの比較検討が必要である。このためにはどうしてもモニタリングを行わなければならない（ただし，稼動記録の分析で十分なこともある）。モニタリングはシステムの現状を把握するには有効であるが予測は不可能である。したがって，解析的手法とモニタリングの組合せが最も有力な手法であるといえる。

本稿では，バッチ処理システムを対象にして，解析的手法とモニタリングによる性能評価について述べる。評価要素としては多重プログラミングに重点を置くが，システム構成の問題として多重プロセッサについても検討する。評価尺度はスループットを取上げるが，これに直接的影響を与える中央処理装置（CPU）の効率を中心に議論を進める。

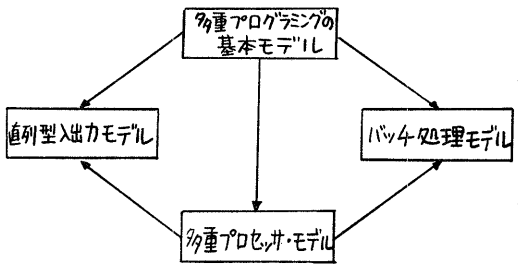

図1-1 モデルの関連図

提案するモデルは図1-1に示すように4種類ある。中心となるのは基本モデルで，これは多重プログラミングの機構を待ち行

列の1つである有限行列によって表現したものである。これを拡張したものが多重プロセッサ・モデルである。さらに，カード読取装置とラインプリンタを考慮した典型的なバッチ処理のモデルへも拡張を行っている。また，ディスク装置の入出力動作をより正確に表現するように拡張したのが直列型入出力モデルである。

以下，モニタリングの結果を用いて適合性を論じながら，これらのモデルを順に説明していく。そして，最後に，開発したモニタリング・システムの方式について説明する。

2. 多重プログラミングの基本モデル(4)

多重プログラミングの機構を図2-1のように表現する。同図において，主記憶装置にCPUが実行すべきプログラム（プロセス）がN個置いてあるものとする。CPUはこれらのプロセスを適当に切替えて実行する。切替えの時点は実行中のプロセスから入出力要求が出されたときである。これにより，CPUからは入出力用のチャネルに対して適当な間隔で入出力要求が出ていくことになる。ただし，N個のプロセス全部が入出力要求の完了を待つ状態になると，入出力要求の発生は停止する。

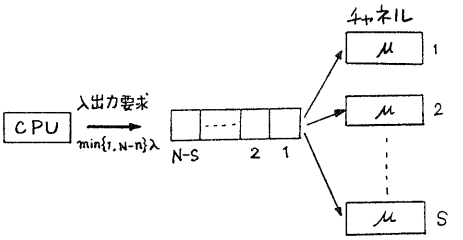

図2-1 多重プログラミングの基本モデル

チャネルはS個あって，入出力要求はどのチャネルで処理されてもよいものとする。チャネル側に存在する入出力要求の数を，処理中のものも含めてn個とする。nは当然 $0 \leqq n \leqq N$ である。CPUからの入出力要求の発生間隔およびこれがチャネルで処理される時間は，それぞれパラメータλおよびμの指数分布に従うものとする。このような待ち行列は有限行列M/M/S(N)と同じことである。この系の平衡状態において，入出力要求の個数がnである平衡状態確率を P_n で表わすと(2.1)式のようになることは既に知られている。(5)

$$P_0 = \left\{ \sum_{n=0}^{S} \frac{\rho^n}{n!} + \frac{\rho^S}{S!} \sum_{n=1}^{N-S} \left(\frac{\rho}{S}\right)^n \right\}^{-1},$$

$$P_n = \begin{cases} \dfrac{\rho^n P_0}{n!}, & (1 \leqq n \leqq S) \\ \dfrac{\rho^n P_0}{S!\, S^{n-S}}, & (S < n \leqq N) \end{cases},$$

$$\rho = \frac{\lambda}{\mu} \qquad (2.1)$$

系の状態がNのときは，実行できるプロセスがないから，CPUは待ち状態になる。この確率は P_N である。したがってCPUが動作できる確率は $1 - P_N$ となり，この確率をCPU効率と定義する。CPU効率を α $(0 \leqq \alpha \leqq 1)$ で表わすと，これは(2.2)式のようになる。CPU効率と同様にチャネルの効率も定義できる。系の状態が0のときは，S個のチャネル全部が遊びの状態になる。$n \geqq S$ であれば全部のチャネルが動作する。1個のチャネルの効率の最大は1であるから，S個のチャネルでは最大はSである。チャネルの総効率を θ で表わすと（2.3）式のようになる。

$$\alpha = 1 - \frac{\rho^N P_0}{S!\, S^{N-S}} \qquad (2.2)$$

$$\theta = S - \sum_{n=0}^{S-1} (S-n) P_n \qquad (2.3)$$

図2-2は基本モデルのCPU効率と測定データを比較したものである。測定は2つのシステムで行ったものを示してある。1つはNEACシリーズ2200モデル700（以下M700）であり，他はACOSシリーズ77 NEAC システム700（以下S700）である。オペレーティング・システムは，M700がMOD7，S700がACOS-6である。

測定にはソフトウェア・モニタリングの手法を用いた。M700の場合は，我々が開発したソフトウェア・モニタ[6]を使用した。これについてはその概要を後述する。S700では，日本電気が開発したモニタ（リソース測定モニタ）を使用した。

図2-2の計算値と実測値は，いずれの場合もだいたい合っているといえる。したがって，この基本モデルは比較的よい適合性をもつと考えてよい。なお，図2-2の(b)において，横軸が平均多重度となっているのは，測定中に変動する多重度の平均という意味である。(a)の場合は，多重度はほとんど変動しない。図2-3はチャネル効率を比較したものであるが，これもほぼ合うことがわかる。

以上のCPU効率には，当然のことながらモニタのオーバヘッドが含まれている。このCPU効率からモニタの部分を引いたものが実際にプロセスに使える部分であるから，これをプロセス効率と呼ぶことにする。CPUが次の入出力要求の発生まで連続してプロセスを実行する時間の平均は$1/\lambda$である。この中にモニタが使う時間δが含まれているものと仮定する。すると，$1/\lambda-\delta$が実際にプロセスに使える時間となる。$1/\lambda-\delta$と$1/\lambda$の比を$\eta=(1/\lambda-\delta)/(1/\lambda)=1-\lambda\delta$のように表わすと，$\eta$はCPU効率のうちプロセスに使える部分の比率を表わすことになる。多重度によって変化しないものとすると，プロセス効率βは$\beta=\eta\alpha$のようになる。図2-4はプロセス効率の実測値との比較である。これも，計算値と測定値はだいたい合っているといえる。したがって，モニタのオーバヘッドをこのような形で取扱っても，プロセス効率の推定ではかなりよい結果の得られることがわかる。

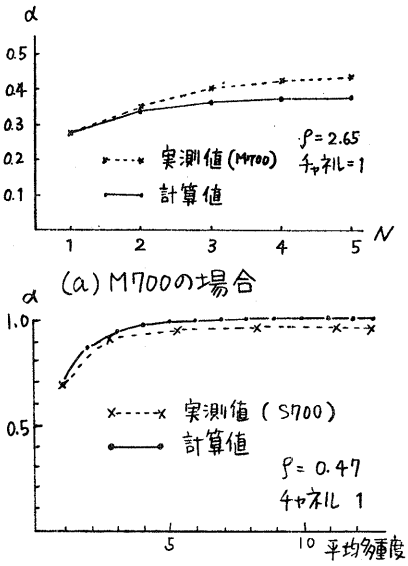

図2-2 基本モデルのCPU効率の比較

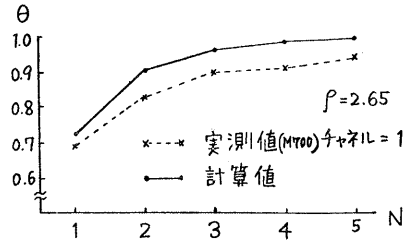

図2-3 チャネル効率の比較

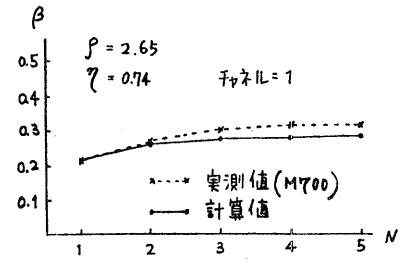

図2-4 プロセス効率の比較

3. 多重プロセッサのモデル[7]

基本モデルを多重プロセッサの場合に拡張するには，図2-1において，独立にしかも同時に動作できる同種のCPUを複数個考えればよい。このモデルは図3-1のようになる。CPUの個数をCとすれば，入出力要求の発生率は$\min\{C, N-n\}\lambda$となる。このときの平衡方程式は，(3.1)式のようになる。これからp_nの一般形を求めると(3.2)式のようになる。

多重プロセッサのCPU効率は，前に定義したチャネル効率と同様に考える。CPUがC個のときのCPU効率をα_Cで表わすと，これは式(3.3)のようになる。図3-2

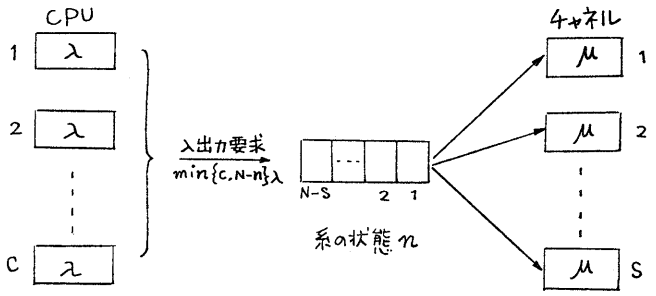

図3-1 多重プロセッサ・モデル

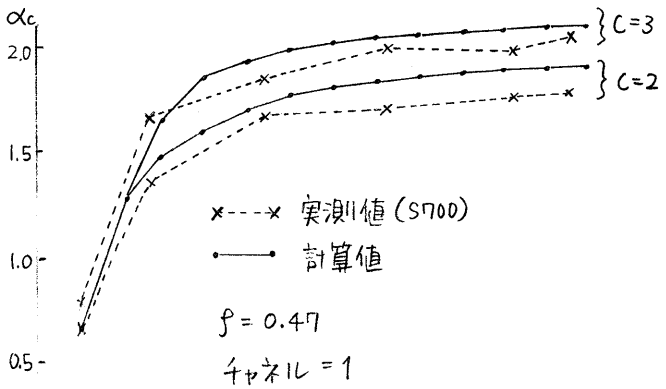

図3-2 α_cの比較

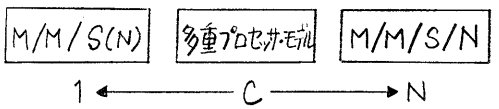

図3-3 多重プロセッサ・モデルの関係図

はα_cの計算値と測定値を比較したものである。このデータはC=2とC=3の場合だけであるが，いずれもこの多重プロセッサ・モデルの適合性をある程度裏付けている。

なお，(3.2)式において，C=1とするとP_nは基本モデルのものと一致する。また，C=Nとすると，P_nは有限入力限の待ち行列M/M/S/Nのものと等しくなる[5]。したがって，この多重プロセッサのモデルは，図3-3のように，有限行列と有限入力限の中間の待ち行列になっているといえる。

$$
\begin{aligned}
&-C\lambda P_0 + \mu P_1 = 0,\\
&a_1\lambda P_{n-1} - (a_2\lambda + a_3\mu)P_n \\
&\quad + a_4\mu P_{n+1} = 0, \quad (1 \leqq n \leqq N-1),\\
&\lambda P_{N-1} - S\mu P_N = 0,\\
&a_1 = \begin{cases} C, & (1 \leqq n \leqq N-C) \\ N-n+1, & (N-C+1 \leqq n \leqq N-1), \end{cases}\\
&a_2 = \begin{cases} C, & (1 \leqq n \leqq N-C) \\ N-n, & (N-C+1 \leqq n \leqq N-1), \end{cases}\\
&a_3 = \begin{cases} n, & (1 \leqq n \leqq S-1) \\ S, & (S \leqq n \leqq N-1), \end{cases}\\
&a_4 = \begin{cases} n+1, & (1 \leqq n \leqq S-1) \\ S, & (S \leqq n \leqq N-1) \end{cases}
\end{aligned}
\qquad (3.1)
$$

$$
\begin{aligned}
&P_n = \frac{A_n}{B_n}\rho^n P_0,\\
&A_n = \begin{cases} C^n, & (1 \leqq n \leqq N-C) \\ C^{N-C}\prod_{k=N-n-1}^{C} k, & (N-C+1 \leqq n \leqq N), \end{cases}\\
&B_n = \begin{cases} n!, & (1 \leqq n \leqq S) \\ S!\,S^{n-S}, & (S < n \leqq N), \end{cases}\\
&P_0 = \left\{1 + \sum_{n=1}^{N}\frac{A_n}{B_n}\rho^n\right\}^{-1},\\
&\rho = \frac{\lambda}{\mu}
\end{aligned}
\qquad (3.2)
$$

$$
\alpha_C = C - \sum_{k=0}^{C-1}(C-k)P_{N-k} \qquad (3.3)
$$

4. バッチ処理システムの解析 (9)

基本モデルおよび多重プロセッサ・モデルにおいては，CPU効率あるいはチャネル効率について議論をした。バッチ処理システムを考えた場合，CPU効率などの検討だけでは十分ではなく，最終的にはスループットやターン・アラウンド・タイムが必要となる。この節では基本モデルを更に拡張し，典型的なバッチ処理システムのスループットについて考察する。

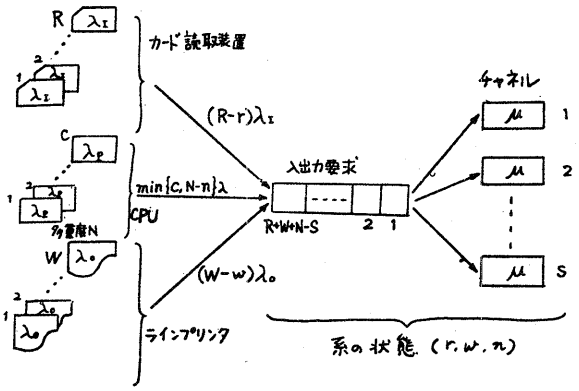

図4-1 バッチ処理システムのモデル

ここで考えるバッチ処理では，ジョブはカード読取装置からディスク装置へ入力され，ジョブが実行中に出す入出力要求（結果なども含む）もすべてディスク装置に対するものとし，さらに計算結果はディスク装置からラインプリンタへ出力されるものとする。カード読取装置，CPUおよびラインプリンタがディスク装置に対して行う入出力要求を，すべて同種のものとみなせば，バッチ処理システムのモデルは図4-1のように表現することができる。ただし，この場合ディスク装置をチャネルとみなしている。

ジョブの入力動作はカードの読取動作とディスク装置への書込み動作の繰返しである。同様に結果の出力もラインプリンタの印字動作とディスク装置からの読出し動作の繰返しである。カードの読取動作およびラインプリンタの印字動作時間はそれぞれパラメータλ_Iとλ_oの指数分布に従うものとする。カード読取装置とラインプリンタの個数をそれぞれRとWとし，待ち行列の中で処理中のものも含めて，カード読取装置から出された入出力要求の数をr ($0 \leqq r \leqq R$)，ラインプリンタからのものをw ($0 \leqq w \leqq W$)で表わす。入出力要求の発生率は，カード読取装置では$(R-r)\lambda_I$，ラインプリンタでは$(W-w)\lambda_o$と表わせる。CPUからのものは，多重プロセッサのときと同じで，$\min\{C, N-n\}\lambda$となる。

系の平衡状態において，系の状態が(r, w, n)である平衡状態確率を$P_{r,w,n}$で表わす。系の平衡方程式を作るには(r, w, n), $(r+1, w, n)$, $(r, w+1, n)$, $(r, w, n+1)$, $(r-1, w, n)$, $(r, w-1, n)$, $(r, w, n-1)$の状態から(r, w, n)の状態への遷移確率を求める必要がある。ところが，たとえば$(r+1, w, n)$から(r, w, n)への遷移は，新しい入出力要求の到着がなく，かつ現に処理中の入出力要求の処理が1つ終了し，しかもそれがカード読取装置からのものでなければならない。処理中の入出力要求がどの装置からのものであるかどうかは不明であるので，当然終了したものについても不明である。そこで，ここでは終了した入出力要求がカード読取装置からのものである確率を$(r+1)/(r+w+n+1)$として遷移確率を求める。

いま，$r+w+n+1 \geqq S$とすると，処理中の入出力要求が1つ完了し，他に1つも入出力要求が到着しない確率は，到着も処理も指数分布を仮定しているのでμSである。したがって，$(r+1, w, n)$から(r, w, n)への遷移確率は$\mu S(r+1)/(r+w+n+1)$となる。$r+w+n < S$の場合はμSが$\mu(r+w+n+1)$となるので，遷移確率は$\mu(r+1)$となる。$(r, w+1, n)$および$(r, w, n+1)$から(r, w, n)への遷移確率についても同様に考えればよい。

以上の方法で遷移確率を求めるものとすると，系の平衡方程式は(4.1)式のようになる。$P_{r,w,n}$の一般形を求めるのは困難である。したがって，(4.1)式より連立一次方程式をつくり，これを解いて個々の$P_{r,w,n}$を求めることになる。

スループットを求めるためには，各装置の効率を求めなければならない。カード読取装置，ラインプリンタ，CPUの効率をそれぞれ$\alpha_R, \alpha_W, \alpha_{BC}$で表わすものとすると，これは以前と同様にして(4.2)式のように表わされる。

表4-1はR=1，W=1，C=1，N=3の場合の各効率の計算値と実測値を比較したものである。Ⓡは入力処理，Ⓦは出力処理，Ⓟはプロセスの実行（ジョブの実行）を表わす。Ⓟにおいては多重度を常に3（N=3）にしてある。処理の組合せは，たとえばⓇ+Ⓦのように表わす。これは入力と出力を同時に行うことを示す。この例における処理の組合せは全部で7通りである。計算値と実測値は，ほぼ似たような傾向を示しているので，このモデルの適合性は一応あると考えてよいであろう。

各装置の効率が求まれば，それぞれの単位時間当りの処理件数（スループット）を求めることができる。たとえば，表4-1を基にして，あるジョブ群について計算したスループットは表4-2のようになる。ジョブの処理においては，ある定められた時間内で，ジョブの入力，実行および

処理形式	α_R		α_W		α_{BC}	
	実測値	計算値	実測値	計算値	実測値	計算値
Ⓡ	0.43	0.43	—	—	—	—
Ⓦ	—	—	0.90	0.90	—	—
Ⓟ	—	—	—	—	0.36	0.30
Ⓡ+Ⓦ	0.41	0.41	0.84	0.86	—	—
Ⓡ+Ⓟ	0.26	0.18	—	—	0.28	0.23
Ⓦ+Ⓟ	—	—	0.73	0.73	0.36	0.28
Ⓡ+Ⓦ+Ⓟ	0.27	0.16	0.73	0.69	0.30	0.22

R=1，W=1，N=3，S=1，C=1
ρ_I=1.3，ρ_0=0.1，ρ=3.3

表4-1 $\alpha_R, \alpha_W, \alpha_{BC}$の比較

$$-(a_1\rho_I + a_2\rho_0 + a_3\rho + a_4)P_{r,w,n} + a_5 P_{r+1,w,n} + a_6 P_{r,w+1,n} + a_7 P_{r,w,n+1} + a_8\rho_I P_{r-1,w,n} + a_9\rho_0 P_{r,w-1,n} + a_{10}\rho P_{r,w,n-1} = 0,$$

$$\rho_I = \lambda_I/\mu,\quad \rho_0 = \lambda_0/\mu,\quad \rho = \lambda/\mu,$$

$$a_1 = R-r,\quad a_2 = W-w,$$

$$a_3 = \begin{cases} C, & (n \leqq N-C) \\ N-n, & (n > N-C), \end{cases} \qquad a_4 = \begin{cases} r+w+n, & (r+w+n \leqq S) \\ S, & (r+w+n > S), \end{cases}$$

$$a_5 = \begin{cases} r+1, & (r+w+n+1 \leqq S,\ r \leqq R-1) \\ \dfrac{S(r+1)}{(r+w+n+1)}, & (r+w+n+1 > S,\ r \leqq R-1) \\ 0, & (r=R) \end{cases}$$

$$a_6 = \begin{cases} w+1, & (r+w+n+1 \leqq S,\ w \leqq W-1) \\ \dfrac{S(w+1)}{(r+w+n+1)}, & (r+w+n+1 > S,\ w < W-1) \\ 0, & (w=W) \end{cases}$$

$$a_7 = \begin{cases} n+1, & (r+w+n+1 \leqq S,\ n \leqq N-1) \\ \dfrac{S(n+1)}{(r+w+n+1)}, & (r+w+n+1 > S,\ n \leqq N-1) \\ 0, & (n=N) \end{cases}$$

$$a_8 = \begin{cases} 0, & (r=0) \\ R-r+1, & (r \neq 0) \end{cases} \qquad a_9 = \begin{cases} 0, & (w=0) \\ W-w+1, & (w \neq 0) \end{cases}$$

$$a_{10} = \begin{cases} 0, & (n=0) \\ C, & (1 \leqq n \leqq N-n) \\ N-n+1, & (n > N-C) \end{cases}$$

(4.1)

$$\alpha_R = R - \sum_{k=0}^{R-1} \Sigma\,(R-k)P_{R-k,w,n},$$

$$\alpha_W = W - \sum_{k=0}^{W-1} \Sigma\,(W-k)P_{r,w-k,n},$$

$$\alpha_{BC} = C - \sum_{k=0}^{C-1} \left\{ (C-k) \Sigma P_{r,w,N-k} \right\}$$

(4.2)

結果の出力件数は等しくなっていなければならない。これには処理形式を適当な比率で用いなければならない。この比率を表4-2の右側に示したx_1～x_7で表わす。ジョブの入力，実行，出力件数が等しいものの中から最大なものを求めると，それがそのシステムにおけるジョブの最大処理件数（最大トータル・スループット）となる。これには線形計画法を用いればよい。表4-2の例で求めた最大トータル・スループットは表4-3のようになる。

処理形式	入力(件/時) 実測値より	入力(件/時) 計算値より	出力(件/時) 実測値より	出力(件/時) 計算値より	実行(件/時) 実測値より	実行(件/時) 計算値より	比率
ⓡ	101	101	—	—	—	—	x_1
ⓦ	—	—	50	50	—	—	x_2
ⓟ	—	—	—	—	123	103	x_3
ⓡ+ⓦ	96	96	46	47	—	—	x_4
ⓡ+ⓟ	61	42	—	—	82	77	x_5
ⓦ+ⓟ	—	—	40	40	108	93	x_6
ⓡ+ⓦ+ⓟ	63	39	40	38	77	72	x_7

表4-2 スループットの例

処理形式	x_1	x_2	x_3	x_4	x_5	x_6	x_7	最大トータル・スループット(件/時)
実測値より	0	0.13	0	0.46	0	0.41	0	44.1
計算値より	0	0.07	0	0.46	0	0.47	0	43.9

表4-3 トータル・スループットの例

以上の議論では，処理形式は7通りであったが，実際には$(R+1)(W+1)(N+1)-1$の組合せがある。$\bar{r}$$(0 \leqq \bar{r} \leqq R)$と$\bar{w}$$(0 \leqq \bar{w} < W)$をそれぞれ使用するカード読取装置とラインプリンタの台数，$\bar{n}$$(0 \leqq \bar{n} \leqq N)$を多重度とする。そして，このときの入力，出力，実行のスループットをそれぞれ$a_{\bar{r},\bar{w},\bar{n}}$，$b_{\bar{r},\bar{w},\bar{n}}$，$c_{\bar{r},\bar{w},\bar{n}}$とする。ただし，$\bar{r}=\bar{w}=\bar{n}=0$は意味がないので除外する。処理形式の使用比率を$x_{\bar{r},\bar{w},\bar{n}}$とすると，制約条件は(4.3)式となる。目的関数は入力，出力，実行のいずれを用いてもよいが，入力に関するものを使用すると(4.4)式のようになる。

$$\sum_{\bar{r}=0}^{R}\sum_{\bar{w}=0}^{W}\sum_{\bar{n}=0}^{N}(a_{\bar{r},\bar{w},\bar{n}}-b_{\bar{r},\bar{w},\bar{n}})x_{\bar{r},\bar{w},\bar{n}}=0,$$

$$\sum_{\bar{r}=0}^{R}\sum_{\bar{w}=0}^{W}\sum_{\bar{n}=0}^{N}(a_{\bar{r},\bar{w},\bar{n}}-c_{\bar{r},\bar{w},\bar{n}})x_{\bar{r},\bar{w},\bar{n}}=0,$$

$$\sum_{\bar{r}=0}^{R}\sum_{\bar{w}=0}^{W}\sum_{\bar{n}=0}^{N}x_{\bar{r},\bar{w},\bar{n}}=1,$$

$$x_{\bar{r},\bar{w},\bar{n}} \geqq 0 \qquad (4.3)$$

$$\sum_{\bar{r}=0}^{R}\sum_{\bar{w}=0}^{W}\sum_{\bar{n}=0}^{N}a_{\bar{r},\bar{w},\bar{n}} \qquad (4.4)$$

5. 直列型入出力モデル[9]

基本モデルにおいては，チャネルとして暗黙のうちにディスク装置を対象にしていた。ディスク装置における入出力動作を細く見れば，シーク動作と転送動作に分けることができる。転送動作には回転待ちと移送動作が含まれる。シーク動作と転送動作は，いくつかのそれほど無理でない仮定を設ければ，全く独立に並行してその動作が行われると考えてよい。入出力要求の処理は最初にシーク動作が行われ，次に転送動作が行われる。シーク動作が行われるチャネルと転送動作が行われるチャネルを分けて考え，これをそれぞれ第1チャネル，第2チャネルと呼ぶことにする。つまり，入出力要求は第1と第2チャネルで直列に処理される。この直列型の入出力のモデルは図5-1のようになる。

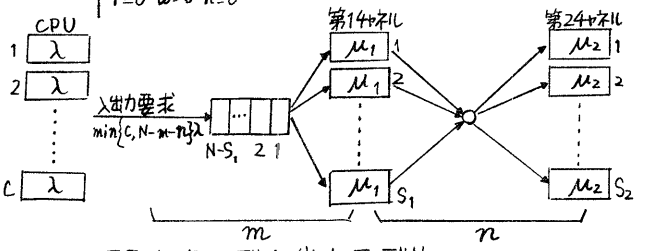

図5-1 直列型入出力モデル

第1および第2チャネルの個数をそれぞれS_1，S_2とし，それぞれの処理時間分布はパラメータμ_1，μ_2の指数分布に従うものとする。第1チャネル側の入出力要求の個数を処理中のものも含めて$m$$(0 \leqq m \leqq N)$とする。ここで処理が完了した入

出力要求は，第2チャネルが空かないと移動せず，その間処理が行われた第1チャネルをブロックの状態にする（ディスク装置の機能）。nはこの処理を完了した入出力要求を含まない。第2チャネルで処理中の入出力要求と，第1チャネルで処理を終了し第2チャネルへ移されるのを待っているものの個数をnとする。S_1より多くの入出力要求は第2チャネル側に行かないので，$n \leqq S_1$であり，さらに多重度との関係で$0 \leqq n \leqq N$であるから，結局$0 \leqq n \leqq \min\{N, S_1\}$となる。

この待ち行列の平衡方程式は，$N \leqq S_1$と$N > S_1$の場合で異なる。$N \leqq S_1$では到着した入出力要求はすぐに第1チャネルでの処理を受けることができるので，S_1の影響がなくなって，平衡状態確率の一般形が容易に求まる。ここでは$N \leqq S_1$の場合についてのみ述べる。平衡状態確率を$P_{m,n}$で表わすものとすると平衡方程式は(5.1)式のようになり，その一般形は(5.2)式のようになる。そして，このモデルにおけるCPU効率をα_{TC}で表わすと，これは(5.3)式のようになる。

図5-2はα_{TC}の計算値と実測値の比較である。$C=1$, $S_2=1$の場合ではあるが，このモデルの比較的よい適合性を示している。

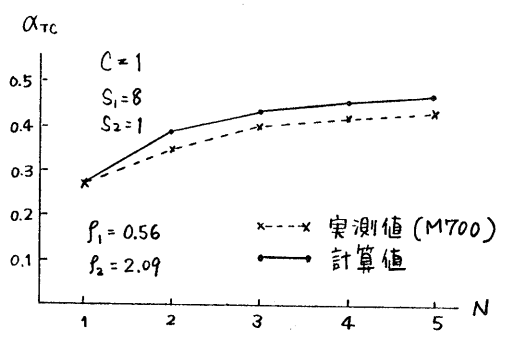

図5-2　α_{TC}の比較

$$
\begin{aligned}
&-C\lambda P_{0,0}+\mu_2 P_{0,1}=0,\\
&-(a_1\lambda+a_2\mu_2)P_{0,n}+a_3\mu_2 P_{0,n+1}\\
&\quad+\mu_1 P_{1,n-1}=0,\ (m=0,\ 1\leqq n\leqq N-1),\\
&-S_2\mu_2 P_{0,N}+\mu_1 P_{1,N-1}=0,\ (m=0, n=N),\\
&-(a_1\lambda+m\mu_1)P_{m,0}+\mu_2 P_{m,1}\\
&\quad+a_4 P_{m-1,0}=0,\ (1\leqq m\leqq N-1,\ n=0),\\
&-N\mu_1 P_{N,0}+\lambda P_{N-1,0}=0,\ (m=N,\ n=0),\\
&-(a_1\lambda+m\mu_1+a_2\mu_2)P_{m,n}+a_3\mu_2 P_{m,n+1}\\
&\quad+(m+1)\mu_1 P_{m+1,n-1}+a_4\lambda P_{m-1,n}=0,\\
&\qquad(m\neq 0,\ n\neq 0,\ m+n\leqq N-1),\\
&-(m\mu_1+a_2\mu_2)P_{m,n}+(m+1)\mu_1 P_{m+1,n-1}\\
&\quad+\lambda P_{m-1,n}=0,\ (m\neq 0,\ n\neq 0,\ m+n=N),
\end{aligned}
$$

$$
a_1=\begin{cases} C, & (1\leqq m+n\leqq N-C)\\ N-(m+n), & (N-C<m+n\leqq N+1),\end{cases}
$$

$$
a_2=\begin{cases} n, & (1\leqq n<S_2)\\ S_2, & (S_2\leqq n\leqq N-1),\end{cases}\qquad
a_3=\begin{cases} n+1, & (1\leqq n<S_2)\\ S_2, & (S_2\leqq n\leqq N-1),\end{cases}
$$

$$
a_4=\begin{cases} C, & (1\leqq m+n\leqq N-C)\\ N-(m+n)+1, & (N-C<m+n\leqq N-1),\end{cases}
\qquad (5.1)
$$

$$
\begin{aligned}
&P_{m,n}=A_m B_n Z_{m+n}\rho_1^{\,n}\rho_2^{\,n}P_{0,0},\\
&A_m=\frac{1}{m!},\qquad P_{0,0}=\left\{\sum_{0\leqq m+n\leqq N}A_m B_n\rho_1^{\,m}\rho_2^{\,n}\right\}^{-1},\\
&B_n=\begin{cases}\dfrac{1}{n!}, & (0\leqq n\leqq S_2)\\[2ex] \dfrac{1}{S_2!\,S_2^{\,n-S_2}}, & (S_2<n\leqq N),\end{cases}\qquad \rho_1=\frac{\lambda}{\mu_1},\\
&Z_{m+n}=\begin{cases} C^{m+n}, & (0\leqq m+n\leqq N-C)\\[2ex] \dfrac{C!\,C^{N-C}}{(N-m-n)!}, & (N-C<m+n\leqq N)\end{cases}\qquad \rho_2=\frac{\lambda}{\mu_2},
\end{aligned}
\qquad (5.2)
$$

$$
\alpha_{TC}=C-\sum_{N-C+1\leqq m+n\leqq N}(C+m+n-N)P_{m,n}\qquad (5.3)
$$

6. ソフトウェア・モニタ "TOMS"

これまで提案したモデルはすべて実測データに基づいてその適合性を検討したが,本節ではM700 における実測データを収集するために用いたソフトウェア・モニタで,我々が開発した"TOMS"(Tohoku University Monitoring System)について,その機能と制御方式について概要を述べるとともにいくつかの測定例を示す。

TOMSはオペレーティング・システムMODのモニタに組込む形で作られている。データの収集方法としては,サンプリング方式とイベント方式の両方の機能をもっている。サンプリング方式の制御は図6-1に示すようになっている。一定間隔の観測時点を得るために,タイマ・レジスタを用いている。このレジスタにセットした値は,2μsごとに1が引かれ,値がゼロになると制御割込みが発生する。この割込みは,通常は優先度が一番高いので,観測時点は乱れることはない。各観測時点で収集するデータは表6-1のようなものである。

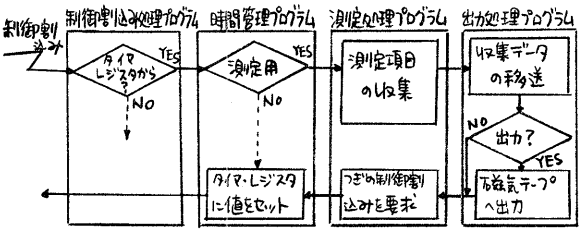

図6-1 サンプリング方式の制御

収集データ	目的
(1) プログラム・ステータス・レジスタ	CPUの状態
(2) つぎに実行する命令のアドレス	観測時点がモニタ内かユーザ・プログラム内かの判定
(3) 標準チャネル・ステータス・テーブル	標準チャネルの使用状態
(4) 高速チャネル・ステータス・テーブル	高速チャネルの使用状態
(5) 周辺装置テーブル	ドラム装置,ディスク装置の使用状態
(6) つぎに実行する命令	つぎに実行する命令が参照するデータのアドレスを求める
(7) インデックス・レジスタの内容	
(8) アドレス・レジスタの内容	
(9) リロケーション・レジスタの内容	

表6-1 サンプリングにおける収集データ

イベント方式は特定事象の発生ごとに観測を行うものであるが,TOMSではモニタ・プログラムの実行時間を測定することを目的としており,特定事象として不正命令コードの実行を用いている。モニタの実行中にこの不正命令コードを実行すると緊急割込みが発生するので,これによりイベントの発生を知ることができる。

プログラムの実行時間を測定するには,その入口と出口を測定点にし,そこでの時刻を調べればよい。測定点に不正命令コードを埋め込む場合,もとの正しい命令コードを保存しておく必要がある。このために図6-2に示すような測定点テーブルを用いる。図6-3はイベント方式の制御の流れである。データ収集の作業が終って測定点に戻るとき,そこの不正命令を正しいものに置き替える。つまり,正しい命令の実行は測定点で行われる。このため,この測定点の命令コードはあとで再び不正命令コードに置き替えておく必要がある。

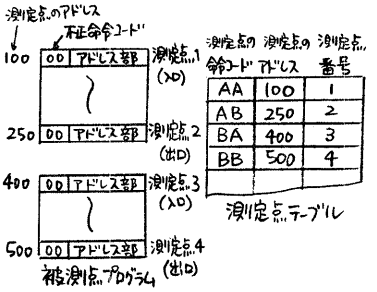

図6-2 測定点テーブルの例

以下にCPUとチャネルがそれぞれ1台のときの測定例を示す。表6-2,6-3はサンプリング方式によるもので,測定間隔は100msである。表6-2はCPU

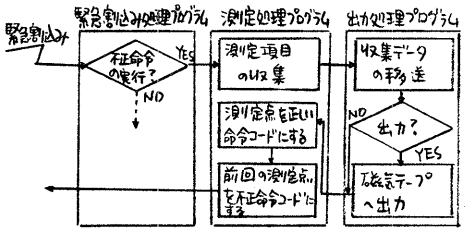

図6-3 イベント方式の制御

とチャネルの動作状況を示したものであるが，8つの測定項目のうち，奇数番のものは値が大きく，偶数番のものは小さい程よいシステムであるといえる。表6-3はCPUの配分とチャネルの動作状況を示したものである。パーティション1～3は，ここに置かれたユーザのプログラムを意味する。表6-4はイベント方式の測定結果である。

測定項目 ＼ 多重度	多重度1	多重度3	多重度5
① CPU Active	28.6%	43.2%	48.0%
② CPU Only	22.0	5.4	1.8
③ CPU/Channel Overlap	6.6	37.8	46.2
④ Channel Only	59.2	52.6	49.2
⑤ Channel Busy	65.8	90.4	95.4
⑥ CPU Wait	71.4	56.8	52.0
⑦ User State	22.1	32.0	36.1
⑧ Monitor State	6.5	11.2	11.9

表6-2 CPUとチャネルの動作状況

CPU Active	モニタ・プログラム	14.92%
	入力プログラム	0.19
	出力プログラム	0.21
	パーティション1	4.29
	パーティション2	6.19
	パーティション3	3.67
Channel Busy	カード読取り機	24.66
	ラインプリンタ	75.10
	ディスク装置	98.24

表6-3 CPUの配分とチャネルの動作状況

モニタ・プログラム	機能	平均実行時間(μs)	平均使用回数(回/秒) 多重度1	多重度3	多重度5
ディスパッチャ	タスクへのCPUの割当て	52	102.7	145.7	162.2
シリアル制御	資源のシリアル制御	88	0.8	1.0	1.3
タスクの同期処理1	事象待ちタスクの制御	71	23.7	31.3	35.4
タスクの同期処理2	事象完了後のタスクの制御	120	29.2	36.9	40.5
標準出力制御	システムの出力ファイルの制御	66	17.3	29.7	32.4

表6-4 モニタ・プログラムの実行時間と使用頻度

7. むすび

多重プログラミングと多重プロセッサに関する簡単な待ち行列モデルを提案し，それが現実のシステムをある程度反映していることを，計測によって確認した。モデルは一応バッチ処理システムを対象にしており，タイムシェアリング・システムは考慮していない。現在では1つの計算機システムでこの両方を同時に実現するものが多くなっているので，総合的なモデルの検討が必要と思われる。

TOMSについては，その機能がMODになかったので，新たに開発したのであるが，最近はある程度のハードウェア・モニタあるいはソフトウェア・モニタを汎用の機能として備えているシステムが多くなっており，よい傾向といえる。今後は，常時リアルタイムでシステムがモニタリングでき，しかもオーバヘッドの少ないモニタリング・システムの開発が望まれる。

参考文献

(1) 三上他：計算機システムの性能評価技術[I],[II]，電子通信学会誌，Vol.59, No.10,12 (1976).
(2) C.D. Warner: System performance and evaluation - Past, present, and future, Proc. of FJCC, pp. 959～969 (1972).
(3) 宮崎他：TSSの利用分析とシステム評価，情報処理，Vol.18, No.3, pp.265～272 (1977).
(4) 宮崎他：多重プログラミング・システムの効率について，情報処理，Vol.16, No.12, pp.1040～1047 (1975).
(5) 待ち行列研究会編：応用待ち行列事典，丸川書店（昭和46）。
(6) 宮崎他：ソフトウェア・モニタの方式設計とその応用，情報処理学会論文誌，Vol.20, No.1, pp.53～60 (1979).
(7) 松沢他：多重プロセッサ・システムの効率測定について，情報処理学会第19回全国大会予稿集，pp.995～996 (1978).
(8) 宮崎他：計算機システムのトータル・スループットに関する考察，情報処理，Vol.19, No.2, pp.113～118 (1978).
(9) 宮崎他：直列型待ち行列による計算機システムの効率解析，情報処理，Vol.18, No.2, pp.120～127 (1977).