解　説

情報処理システムの性能評価(2)

紀　一誠

1. はじめに

情報処理システムの性能評価技術とは，想定されるシステム構成の下で，利用者特性とシステム性能評価指標との間の関係を定量的に示す技術である事を前稿で述べた．システム性能評価技術は大きく分けると，性能測定技術と性能予測技術に分ける事ができる．本稿ではまず性能測定技術を第2章，性能予測技術の概要を第3章に紹介する．

2. 性能測定技術

2.1 測定の目的

性能測定技術はその目的に応じてさらに，システム性能測定技術と単独ワークロード測定技術の二種類に分類する事ができる．

システム性能測定技術は複数の利用者がシステムを使用している状態において種々のシステム性能評価指標を測定するための技術である．既に稼働中のシステムにおいては，システム稼働状況を監視し負荷増加の検出，性能異常の早期発見と対策を行う．一方，実システムが完成する以前にもシステム測定によって性能評価指標を知りたい場合もある．前稿で紹介したOLTP(On Line Transaction Processing)システムのベンチマークテストであるTPC-Aを利用した測定もその一例である．実機環境を準備し，所定のワークロードを発生させ，その状況の下でシステムの稼働状況を測定する．このとき実機環境の上に実現するシステムは，対象とすべきシステムそのものを完成させたものではなく，性能に関与する部分の動的なふるまいのみを模擬的に実現するシステムである．例えば，データベースについては，各レコードの情報内容は問うことなく，その構造とデータ件数のみを実現する．このようにシステムの骨格部分のみを実現したものは，プロトタイプシステムといわれ，その構築技術はプロトタイピング技術といわれる．プロトタイプシステムを利用してシステム性能を測定する場合には，所定の利用者特性を実現するワークロードを正確に効率良く発生させる必要がある．小規模なシステムの場合には実際にクライアント(端末)側にワークロードを発生させるオペレータを配置し測定を行う事も可能ではあるが，端末数が数百から数千台というように大きくなってくるとこの人海戦術は破綻をきたす．質の良い測定データを得るためには，質の良い負荷の発生方法が必要である．この目的のためにエミュレーション技法といわれる擬似負荷発生法が考えられている．エミュレーション技法とは図1に示すように，多数のクライアントから発生するトランザクション負荷を実際のクライアントからではなくその動きを擬似する負荷発生用のマシンを使用して発生させる方法である．多数の利用者の動作を1台のエミュレータ・マシンで実現するため，効率よく測定が実施できる．しかし，負荷を発生させるエミュレータ・マシンの能力が十分高くないと所定の負荷を実現できないばかりでなく，応答時間の測定誤差も大きなものになる危険性ももつているので，実施に当たっては十分な注意が必要である．

システム性能測定が実稼働状況を想定した測定を行うのにたいして，単独ワークロード測定は一つのプログラムのみを走行させ測定を行う．その目的は，測定対象を一つに絞ることにより，高い精度の測定を行いシステムの設計に反映させることにある．CPUの設計を行う際には何種類ものプログラムについてその1命令毎の動作を詳細に測定したデータが利用される．また，後に述べ

きの　いっせい　NEC C&C研究所
〒216 川崎市宮前区宮崎 4-1-1

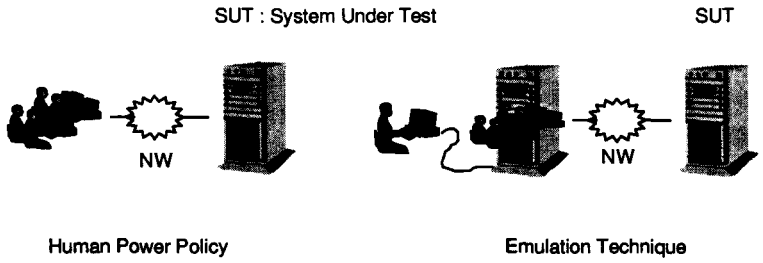

図 1: 擬似負荷発生法

るシングルプロファイル法においては，システム資源に競合が発生しないように1クライアントのみを接続した状態において，対象システムが扱うトランザクションを発生させその処理プロセスの詳細を測定し，そのデータを利用して多数の端末から非同期にトランザクションが発生する場合のシステム性能の予測を行う．

2.2 性能測定技法

性能測定の目的，必要とするデータの測定精度，測定環境等々に応じて性能測定に用いられるツール(測定機能)は変わる．性能測定ツールは測定モニタともいわれる．モニタはその形態により，ハードウエア・モニタ，ソフトウエア・モニタ，ハイブリッド・モニタに分類される．

ハードウエア・モニタは高インピーダンスのハードウエア・プローブ(測定端子)を対象システムに接続して電気信号の形で測定情報を得る方法である．ソフトウエアに全く変更を加える事が無いため，測定のオーバヘッドが全く無い事が最大の特徴である．その反面，ハードウエア的に検出可能な情報に限られるため，収集できるデータの種類に制約がある．

ソフトウエア・モニタは測定機能をすべてソフトウエアで実現する方法であり，様々な種類のものが開発されている．ソフトウエア機能により情報を収集するため，プロセスの切替えといったようなソフトウエア的なイベントの検出が容易であり，また各種テーブル類のようなソフトウエア情報へのアクセスが容易であるため柔軟性のある測定が行える．その反面，総ての機能をソフトウエアで実現するため，測定のオーバヘッドが高くなる危険性をもっている．

ハイブリッド・モニタは両者を併用する方法である．ソフトウエア・プローブをあらかじめ情報収集に必要な箇所に埋め込み，それにより検出されたイベントをハードウエア・インターフェイスに記録していく．測定データの収集はハードウエア的に行われるため測定オーバヘッドは少なく，またソフトウエア・プローブを用いるために測定の柔軟性は高い．いわば両者の利点を生かした形の技法であるが，特殊なハードウエアによる測定機構が必要である事，あらかじめソフトウエア・プローブをシステムに挿入しておかなければならない事などの問題点ももっている．

測定技法は情報収集の方法により，イベント・ドリブン方式とサンプリング方式(タイマ・ドリブン方式ともいわれる)とに分類する事もできる．イベント・ドリブン方式では，注目するイベントが発生する毎にその種別と関連情報を発生時刻とともに記録する．注目するプロセスやタスクの動作分析に適している． サンプリング方式は，一定時間間隔で測定プログラムが起動されシステム状態を記録していく．インプリメントが簡単であるという利点をもっているが，特定のプロセスの動作を追跡し分析する事は得意ではない．また，サンプリングを行う事に起因する測定誤差にも注意が必要がある．性能測定技術に関する詳細は [4][10][16] 等を参照されたい．

3. 性能予測技術

システム性能を評価する際に実機環境を整えることができ実測を行うことが簡単にできるならば，測定を行う事が最も確実に精度の良い結果を得る方法であろう．しかしながら，実機を用いたシステム測定を行うにはプロトタイプの開発，測定環境の整備などと多くの人手と時間がかかる．また，システム設計の初期の段階では実機を使っての測定環境を整えること自体が一般には困難である． このように実機による測定が期間的，コスト的に無理な場合には，予測精度を多少犠牲にしても手軽にシステム稼働時の性能を予測する技術が必要になる．性能予測技術の役割はこのような要請に応えるものである．

性能予測技法はモンテカルロ型シミュレーション技法と待ち行列モデルを利用する解析的性能評価技法の二種類に分類する事ができる．

3.1 シミュレーションによる性能予測

シミュレーションを用いてシステム性能の予測を行う場合には，対象とするシステムの構造や動作をモデル化しそれをシミュレーション言語で記述を行う．この過程はシミュレーション言語によるプログラミング作業そのものである．シミュレーションによるモデル化は労力さえ厭わなければいくらでも詳細な記述が可能であり，システムの細かい動作や条件を反映したモデルを作成することができる． そのため，汎用性の高い製品 [OS や CPU 等] の方式を決める場合にはシミュレーションによる詳細なモデル化と十分な検討が必要とされる．その結果採用された方式の影響は多数のシステムに及ぶことになる．一方，シミュレーションにおいては，一般には実時間の 100 ～ 1000 倍程度の実行時間がかかる．モデルを詳細にすればするほどこの比率は高くなる．従って，短期間に数多くの条件に対応して性能評価結果を求めなければならない状況には必ずしも向いていない．

モンテカルロ型シミュレータとしては，歴史的には GPSS, SIMULA, DYNAMO 等が有名ではあるが，現在では視覚的インターフェースを備えているものやネットワーク専用のシミュレータ等優れたものが多数開発されている．詳しくは [1] を参照．

シミュレーションは誰にでも簡単に使用できる有力な性能予測の道具である．しかしながら，シミュレーションを GIGO(Garbage In Garbage Out) に終らせないためには注意深いモデル化と結果の検討が不可欠である．シミュレーションの実施に際して注意しておくべきことを以下にいくつかあげておく．

- バランスのとれたモデル化を行う．担当者は自分の良く知っている部分を詳細にモデル化し，知らない部分は大雑把なモデル記述をする傾向がある．全体の記述レベルを揃え，本質を見失わないモデル化を行う事が最も大切である．
- 大規模なシミュレーションは避ける．「規模が大きければそれだけ正確な結果が得られる」という考え方はシミュレーションに関する迷信であろう．
- 事前に理論解析を行った上で実施する．トラヒック量の算定と M/M/1 程度の簡単なモデルでも良い．このことはシステム特性についてのイメージをできる限り事前に明確にしておく役割を果たすとともに，シミュレーション結果との照合を行う事によりバグの発見に役立てる事ができる．
- 結果をなるべく大勢の関係者で検討する．一見もっともそうな形で出てくるシミュレーション結果も，実システムの開発を担当する専門家からみると不自然に見える場合がある．視点の異なる多くの関係者による検討により思わぬ誤りを防ぐことができる．
- 結果の精度にあまり神経質にならない．シミュレーションはおおまかな傾向を手軽に把握できる事が利点なので，その特徴を生かす事が大切である．

3.2 待ち行列モデルを用いた性能予測

システムを数理的にモデル化しその解析をすることによって性能予測を行う方法である．混雑現象を数理的にあつかうための理論である待ち行列理論がつかわれる．待ち行列理論やその応用については多数の解説があるのでそれらを参照して頂きたい [5][7][8][9][14] [15][17][19]．ここでは簡単な例によりその概要を示す．

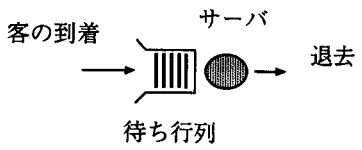

図 2: シングルサーバ待ち行列モデル

待ち行列モデルの中で最も簡単な待ち行列であるシングルサーバモデルを図 2 に示す． 待ち行列モデルにお

いては一般に客の到着時間間隔やサービス時間はある分布に従い確率的に変動するものと考える．待ち行列のふるまいは一般にこれらの分布の形やサービス規律等に依存して変化する．しかし，ほとんどの待ち行列システムの安定性(系の定常性)は分布の形には影響されず，平均到着間隔と平均サービス時間から定まるトラヒック量とサービス能力(サーバ数)との関係を調べることにより判断できる．待ち行列理論では，平均到着時間間隔および平均サービス時間の逆数をそれぞれ，到着率(人/時間)，サービス率(人/時間)と定義しており，記号 λ, μ を使用することが多い．即ち，

$$\lambda = \frac{1}{\text{平均到着間隔}} \tag{1}$$

$$\mu = \frac{1}{\text{平均サービス時間}} \tag{2}$$

である．考えている待ち行列系への負荷を表現するパラメータは次のように定義される．

$$\begin{aligned}\rho &= \text{到着率} \times \text{平均サービス時間} \\ &= \frac{\lambda}{\mu}\end{aligned} \tag{3}$$

この ρ はトラヒック密度(Traffic Intensity)または電話トラヒックの表現にならい呼量といわれる．呼量は無名数で物理的な意味での単位はないが，その量を表すのに，アーラン(Erl.)という単位が使用される事が多い．その由来は電話トラヒック理論の始祖である A.K.Erlang から来ている．トラヒック密度 (3) を用いてS人のサーバをもつ複数サーバの待ち行列の安定条件(定常状態をもつ条件)を次のように表す事ができる．

$$\rho < S \tag{4}$$

条件 (4) はシステムを安定にたもつにはサービス能力以上の負荷をかけてはいけないことを示している．

システム性能評価の初期の段階でこのトラヒック量を推定しサービス能力以上の負荷をかけてしまうような設計になっていないかどうかを確認することは極めて重要である．即ち，システム資源に関する呼量計算をするだけでも処理能力以上に大きな負荷をかけてしまうという大きな設計誤りは最低限防ぐ事ができる．例えば，1トランザクション当たりのCPU使用時間が平均 120ms であるトランザクションが λ 件/秒で到着するものとする．CPU呼量はこのとき，

$$\rho = \frac{\lambda}{\mu} = \lambda \times 0.12 \tag{5}$$

となる．従って，CPUは1台とすると系が定常状態をもつための条件 $\rho < 1$ から，$\lambda < 8.\dot{3}$ が得られ，このシステムの最大トランザクション処理能力は約 8.3 件/秒 ということがわかる．これ以上の負荷を想定したシステム設計はできないことは明らかであろう．

それでは，このシステムに上限値 $8.\dot{3}$ 件/秒よりは少々軽い負荷，例えば 8.0 件/秒 をかけたらどうなるであろうか？ CPU呼量(使用率)は (5) と同様にして，

$$\rho = 0.8 \times 0.12 = 0.96 \tag{6}$$

となり，96% の使用率で稼働することがわかる．この時のCPUの待ち時間はどの程度になるであろうか？ この待ち時間には到着間隔とサービス時間の平均値だけではなくそれぞれの分布の形が影響してくる．両者共に指数分布に従うものと仮定した場合には，M/M/1 といわれる待ち行列モデルとなり，その公式から平均待ち時間 D および待ち時間(確率変数)の α% 値 $w(\alpha)$ は

$$D = \frac{\rho}{1-\rho} \cdot \frac{1}{\mu} \tag{7}$$

$$w(\alpha) = -\frac{1}{\mu(1-\rho)} \ln \frac{1-\alpha}{\rho} \tag{8}$$

となる．これらの結果 (7) (8) から，$\rho = 0.96$ のときには $D = 2.88$, $w(0.9) = 6.8$ 等の値が計算でき，1トランザクション当たり 0.12 秒の CPU 使用時間に対して平均 2.88 秒の待ち時間が発生し応答時間を悪化させること，また到着するトランザクションの 10% は待ち時間が 6.8 秒を越えること等がわかる．

このように，応答時間に関わる性能評価指標を予測する場合にはシステム資源(この場合はCPU)の競合により待ち時間を推定する事が必要になり，待ち行列モデルはそのための有力な手段として活用されている．

3.3 待ち行列網モデルを用いた性能予測

前節では一つの待ち行列から成るモデルを考えたが，複数のプロセスが複数のシステム資源を利用しながら処理を進めるシステムでは，ある資源の待ち時間は他の資源の状態とは独立ではなく相互に関連をもっている．従って，システム全体を考えた性能評価作業を行う場合には各待ち行列を個別に扱うのではなく，複数の待ち行列相互の関連を的確に反映した待ち行列モデルが必要となる．その要請に応えることができたものが待ち行列網(ネットワーク)モデルである．ここでは閉鎖型待ち行列網モデルの最も簡単な応用であるセントラルサーバモデル [3] を用いてその考え方を紹介する．

コンピュータシステムにおいて，1トランザクションがシステム内の各種資源を使用しながら処理を進めて行く様子は，例えば図3のように考えてみることができる．

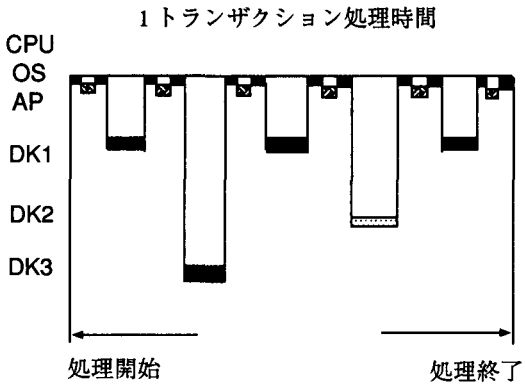

図 3: トランザクション処理時間の内訳

即ち，トランザクションを扱うプロセスはまずOSにより起動され，続いてそのプロセスはアプリケーションプログラム(AP)を起動し所定の処理を行いCPU資源を使用する．また，データベース(DB)にアクセスする場合にはOSカーネルの機能を利用しI/O(Input/Output)処理を行いアクセスの終了を待つ．アクセスが終了すると割り込みによりOSが起動されその処理を行った後再びプロセスに制御を渡す．以下このような処理が何回か続いた後最後にOSが終了処理を行いトランザクション処理を終了する．このようにCPUと入出力装置を交互に使用しながらトランザクション処理をすすめるK個のプロセスがシステムに存在している状況をモデル化したものが図 4 に示されるセントラルサーバモデルであり，コンピュータシステムの性能評価モデルとして古くから利用されてきている．コンピュータシステムの性能評価モデルとしては開放型待ち行列網(網外と客の出入りがある)よりもむしろセントラルサーバモデルに代表される閉鎖型待ち行列網(網外との客の出入りがない)の方が良く利用される．その理由は，閉鎖型待ち行列網モデルを用いる事によりプロセス数が有限であることの効果，システム資源間の構造的関係等を的確に表現することができる事にある．

実際の性能評価作業においては，このセントラルサーバモデルをさらに拡張発展させたモデルが使用される．その一例として，QM-X [11][12] におけるモデル化の例を図 5に示す．ここでは網内客数制限のある二つの閉鎖型サブシステム(サブシステム1および2)と，一つの開放型サブシステム(サブシステム3)から構成されるシステムを扱っている．待ち行列網モデルを実際に利用するためには，与えられた条件のもとに様々な性能評価指標を計算するためのソフトウエア・パッケージが必要になる．先に述べた QM-X の他に主なものとして，BEST-1[2]，RESQ[18]，蟻塚 [13]，QNAP2[20] 等がある．また，パッケージの解説については [6] を参照されたい．

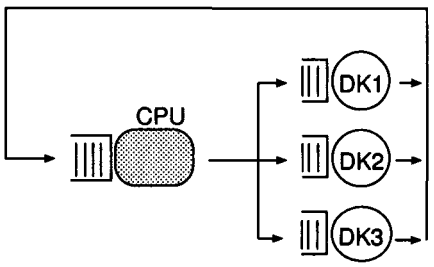

図 4: セントラルサーバモデル

4. おわりに

情報処理システムの性能評価技術の基盤技術である性能測定技術と性能予測技術について概説した．性能評価技術は待ち行列理論に代表される理論的な側面と実際のシステムを測定するという実験的な側面とを合わせもつ総合的な技術体系と考えることができよう．そこでは最先端の理論的成果が応用に直接結び付いた例や，現実の要請が理論の進展を促した例等が数多くみられる．性能評価技術の分野は理論と応用の両輪がうまく回転して技術の進歩を生み出している分野であるといえよう．

参考文献

[1] 特集 ORソフト - 離散系シミュレーション，オペレーションズ・リサーチ, vol.38, no.11, 1993.

[2] J.P. Buzen , R.P. Goldberg, A.M. Langer, E. Lentz, H.S. Schwenk, D.A. Sheetz and A. Shum, BEST/1-Design of a tool for computer system capacity planning,*Proc. AFIP NCC* (1978),pp.447-455.

[3] J.P. Buzen, Computational algorithm for closed queueing networks with exponential servers, *Comm. ACM* **16** (9) (1973) 527-531.

[4] D.Ferrari,G.Serazzi, and A.Zeinger, Measurement and Tuning of Computer Systems, Prentice-Hall,1983.

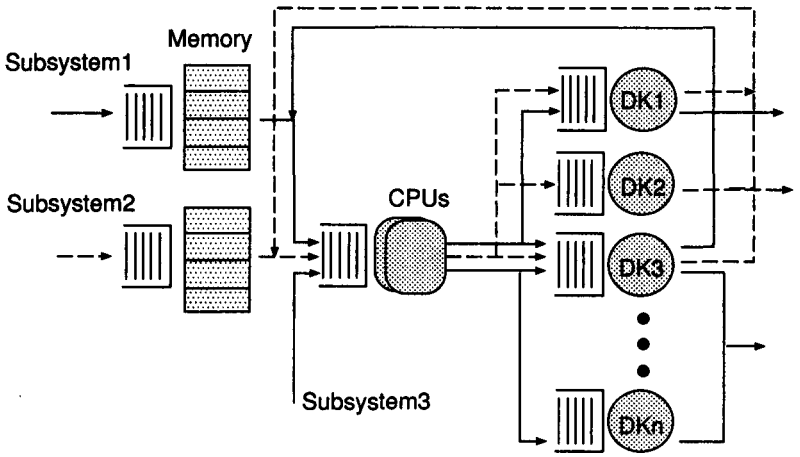

図 5: QM-X で用いられる待ち行列網モデル例

[5] E. Gelembe, I. Mitrani (秋丸, 橋田監訳), 計算機システムの解析と設計, オーム社, 1988.

[6] 長谷川利治 (Ed.), 特集 待ち行列網のパッケージとシミュレータ, オペレーションズリサーチ学会誌, **30** (7) (1985).

[7] 橋田温, 川島幸之助, 待ち行列ネットワークモデルによる計算機システムの性能評価, 情報処理学会誌, **21** (7) (1980) 743-750.

[8] 橋田温, 情報処理システムにおける待ち行列理論の応用 (1)/(2), 情報処理学会誌, **18** (2)/(3) (1977) 184-194 / 289-297.

[9] 橋田温, 計算機システムにおける待ち行列, オペレーションズリサーチ学会誌, **22** (4) (1977) 242-250.

[10] P.Heidelberger and S.S.Lavenberg, Computer Performance Evaluation Methodology, IEEE Trans.Comput.,vol.c-33,pp.1195-1220,1984.

[11] 紀一誠, 納富研造, 待ち行列網モデルによる計算機システムの性能評価用ソフトウェアパッケージ QM-X, 情処学論, **25** (4) (1984) 570-578.

[12] I. Kino and S.Morita, PERFORMS- A support system for computer system performance evaluation, *Proc. Modelling Techniques and Tools for Performance Analysis*, Paris (North-Holland, 1984) 119-138.

[13] 小島, 春木, 他, 待ち行列網解析ツール「蟻塚」の開発 (1)/(2), 第 30 回情処全大, (1985) 7D-7/8.

[14] H. Kobayashi, *Modeling and Analysis*, (Addison-Wesley, 1978)

[15] S.S.Lavenberg (Ed.), *Computer Performance Handbook*, (Academic Press,Newyork,1983).

[16] P.McKerrow, Performance Measurement of Computer Systems, Addison-Wesley,1987.

[17] 三上徹, 紀 一誠, 吉澤康文, 計算機システム性能解析の実際, 情報処理学会, 情報処理叢書 10, オーム社, 1982.

[18] C.H. Sauer,M. Reiser and MacNair : RESQ- A package for solution of generalized queueing networks, *Proc. NCC* 1977, pp.977-986.

[19] 高橋豊, 宮原秀夫, 長谷川利治, 情報システムの性能評価 -2, 待ち行列システム, システムと制御, **27** (4), (1983) 260-267.

[20] M. Veran and D. Potier, QNAP2: A portable environment for queueing network modeling, *Proc. Modelling Techniques and Tools for Performance Analysis* Paris (1984).