計算機システムの評価について*

萩　原　　宏**

1. まえがき

計算機の発展の初期の段階で，手操作によって孤立したプログラムを処理していた時代には，処理能力は専らハードウェアの性能によって左右されており，したがって，その評価も簡単であり，ハードウェアの機能，演算速度，記憶容量あるいは稼動率，信頼性，保守性などが中心になっていた．しかし，手操作による遊び時間をなくすために，モニターを用いて連続処理を行なうようになると，モニターによる処理が関係してくるために，問題はかなり複雑になってきた．さらに，マルチプログラミングが行なわれるようになると，ソフトウェアが計算機システムの性能に及ぼす影響は大きくなり，ハードウェアだけあるいはソフトウェアだけでなく，これらを総合したシステム全体として性能評価をしなければならなくなってきた．

また，応用分野の拡大に伴ない，利用目的や利用形態の多様性，それに伴なって必要となる機能の相違によって，種々の観点から評価が行なわれなければならなくなり，その項目，方法，基準などもさまざまなものとなる[1]．

ここでは，計算機システムの評価の目的，項目，方法などについて概観し，私見を述べることにする．

2. 評価の目的

システムの評価は種々の目的のために行なわれるであろうが，つぎのようなことが考えられる．

（1） 機種の選定

新しく計算機システムを導入しようとするとき，あるいは既に存在するシステムを入れ換えようとするときに生ずる問題であり，後に述べる項目の他に，既存のシステムがあれば，それとのプログラムの互換性，システムの拡張性あるいは要員の教育の難易などについても考えなければならないであろう．

特に候補となる機種が数種あり，一方が完成してすでに稼動しており，他が未完成である場合の比較は，難しい問題になってくる．

（2） 性能の改善

システムの実際の動作状態を解明して，システムの不均衡や隘路を見出し，システムの構成の変更や増強をはかるための資料を得る目的で行なわれるもので，システムのわずかな修正で大幅に性能が向上した具体例が少なくない．当初のシステム構成の際に不明確であった点が明かになるわけで，新しいシステムを設置した後，安定した状態になったとき，後で述べるような方法を用いて評価を行なうことは重要なことであろう．

（3） 新しいシステムの開発，設計

新しいシステムを設計し製作するときには種々の観点からの事前評価が行なわれなければならない．ハードウェアの設計に当っても，ソフトウェア・システムの開発に当っても，その可能性や性能について，設計や開発の各段階で適切な評価がなされなければならない．この場合，対象とするシステムが実在しないので，既存のシステムのデータを参考にして適当なモデルを想定し，解析的な方法やシミュレーションなどによって予測しなければならないと思われる．特に新しい方式や手法を採用する際には，その及ぼす影響についての推定は不明の点が多いだけに十分慎重に行なわなければならない．

3. 評価の項目

システムを評価する場合，そのシステムに必要な機能は具備されているという前提の下に行なわれるものとして，評価に関する項目として考えられるものをまとめてみよう．

3.1 ハードウェア

ハードウェアに関しては装置によって固有の問題もあると思われるが，一般的なものとしてつぎのようなものが考えられる．

* On the Evaluation of Computer Systems, by Hiroshi Hagiwara (Department of Information Science, Faculty of Engineering, Kyoto University)
** 京都大学工学部情報工学教室

（1） 処理速度

CPU に関しては命令実行時間，メモリーのサイクル時間などが考えられる．命令実行時間としては個々の命令についてだけでなく，頻度を考慮した荷重平均，すなわち，Instruction Mix（たとえば，Gibson Mix[2]）が考えられるが，先行制御が行なわれているとこれだけでは不十分で，適当なテスト・プログラムによる必要がある．さらに Pipe Line 方式，Parallel Processing 方式などがとられると，その処理速度の評価は非常に複雑で困難なものとなる．

（2） 信頼性

ハードウェアの信頼性は技術の進歩によって著しく向上しているが，なお考慮に入れなければならない重要な事項であろう．その尺度としてふつう MTBF*（平均故障間隔）が用いられる．

（3） 保守性

ハードウェアが故障した際，これを修理して再び使用に供されるまでの時間は短い方が望ましいことは明らかである．これは保守に対する配慮がなされているか否かによって大きな影響を受ける．その尺度としてふつう MTTR**（修理に要する平均時間）がとられる．

（4） 稼動率

ハードウェアの運転時間を稼動時間，保守時間および故障時間に分けたとき，すなわち

運転時間＝稼動時間＋保守時間＋故障時間

としたとき，

稼動率＝稼動時間／運転時間

で定義されるもので，ハードウェアが完全な状態で動作し得る時間の割合である．

（5） 利用率

上述の稼動時間の間に無駄に費される時間があるとすると，

稼動時間＝動作時間＋無駄時間

と分けられる．したがって，利用率として，

利用率＝動作時間／稼動時間

と定義される量を考えることができる．

しかし，この値ではハードウェア・システムの中に利用されていない装置があっても，それは無視されてしまっている．そこで，これを考慮に入れるため，各装置の利用率に適当な荷重をかけて平均した量を考えるのが一つの方法であろう．いま，その荷重としてコストを考えれば，その量は

$$\frac{\Sigma\ (\text{装置の利用率}) \times (\text{装置のコスト})}{\Sigma\ (\text{装置のコスト})}$$

と表わされる．これによって，ハードウェアの機器構成を考慮して利用率を考えることができると思われる．

なお，記憶装置の場合は記憶容量の大きさが他の装置の利用率に複雑に関係してくるので，その利用率に関してはこれだけでは不十分でさらに適切な尺度を考える必要があると思われる．

3.2 ソフトウェア

ソフトウェアに関してはハードウェアの場合のように定量的な尺度を求めることは困難な面が多く，またプログラムの種類によって要求される機能がさまざまであり[3]，したがって評価の基準も変ってくるであろう．その中で共通して考えられる具体的な尺度としてはつぎのようなものがある．

（1） 処理速度

制御プログラムでは，機能と処理方式に関係し，タスク・スイッチの時間，リソース割当ての時間，プログラムの転送時間などが問題になるであろう．また，言語処理プログラムでは翻訳処理の時間と目的プログラムの実行時間が問題になるが，これらはある程度相補的な関係にあり，使用目的によって重点のおき方が変る．

（2） 経済性

プログラムを実行するに当って必要な主記憶容量や制御プログラムのうち主記憶に常駐する部分の大きさなどの主記憶占有量の問題の他，各種のリソースの使用率なども問題となるであろう．

（3） 人間に関する要素

計算機に直接関係するプログラマおよびオペレータに関するものである．プログラマに関することとしては，プログラミング言語および制御言語の書き易さ，読み易さ，プログラムの誤りに対する対策，訂正の容易さなどが考えられる．一方，オペレータに関することとしては特に操作性が重要であろう．すなわち，入出力機器（たとえば，カード読取機，ラインプリンタ，磁気テープ装置など）の取扱い易さ，スイッチやランプなどの扱い易さや見易さ，オペレータに対する警報の判別や指示の見易さ，オペレータ・コマンドの入力の容易さ，誤操作に対する対策など種々の事項が考えられよう．さらに，円滑な運転が行なえるように，現在の動作状況を常に掌握している必要があり，

* mean time between failure
** mean time to repair

このための情報が容易に得られるような配慮も望まれるであろう.

3.3 総合システム

ハードウェアおよびソフトウェアを総合したシステム全体としてみると，個々の場合にとりあげた項目の他につぎのようなものがある.

(1) 処理効率

稼動時間をさらにつぎの3つに分けて考える.

○処理プログラムの走行時間: T_P

○制御プログラムの走行時間: T_C

○アイドル時間 : T_I

このとき，処理効率 η として,

$$\eta = T_P/(T_P+T_C+T_I)$$

を考えることができる. ここで T_P は処理プログラムの作り方によって大きく左右されるので，上式の η だけで処理効率を論ずるには問題があり，処理プログラムの効率についても考慮すべきであるが，個々の問題であるのでここでは論じないことにする. T_C はすでに述べたように制御プログラムの機能と処理方式に関係するものであり，T_I は OS の構成，ハードウェアの構成によって左右されるが，できるだけ短縮することが望ましい. しかし，アイドル時間を無くすことができないとすれば，これを積極的に利用し，この時間に診断プログラムを流して予防保守に役立たせたり，後述のソフトウェア・モニタを走らせて動作状況の観測を行うことなどが考えられる.

(2) 総合性能

計算機システム全体としての総合性能としてはいろいろな量が考えられるが，処理形態と利用面から重点のおき方は自ら変ってくる. すなわち，システムの運用の立場からいえば，スルー・プットが最も重要な項目になるであろうし，バッチ処理の利用者の立場からは，ターン・アラウンド・タイムが最も問題になるであろう. また，オンラインのシステムでは応答時間が重要な要素になる.

これらの項目は場合によっては互に相反する関係になるので，性能の評価に当っては目的を明確にして，その立場をはっきりさせることを忘れてはならない.

4. 評価の方法

システムの評価に当っては，種々の方法が考えられるが，システムが設計段階にあるか，試作が完了して一応動作可能な状態にあるか，システムとして完成して定常的な運用が行われているかによって，利用できる資料が異なるので，評価の方法も当然変ってくる. 個々の方法については，それぞれに解説が行われるので，ここではそれらについて概観し，注意すべき点などについての私見を述べよう.

4.1 カタログ・データ（設計データ）による評価

カタログはもっとも容易に手近かに得られる資料であり，システムの概要を知る上では大いに役立つものである. 設計データはシステムが未完成の場合にも利用できるデータとして重要である.

(1) ハードウェア

CPU の個々の命令実行時間などは論理素子が決まり論理設計が完了すれば正確に算出することができるわけであるが，条件によって，たとえば演算数の値などによって命令実行時間の値に差が出ることがあるので注意しなければならない. また，すでに述べたように先行制御が行なわれていると，これから単純にプログラム実行時間を推定することはできない. ハードウェアのその他の装置たとえば周辺機器，チャネル装置などの個々の性能も設計が終れば明らかになるので，システムの構成計画には役立たせることができる.

(2) ソフトウェア

ソフトウェアは，その部分にもよるが，計画時にその機能を明らかにすることはできるが，そのプログラムの大きさや動作時間などは完成時期と共に一般に正確に予測することは困難であり，完成するまではおおよその見当をつけ得るに止まると思われる. すなわち，ソフトウェアに関しては事前に評価するには困難な点が多いといえる. また，一応完成したものであっても，さらにレベルアップが行なわれて性能向上がはかられるので，この点も考慮しなければならない.

(3) カタログ上の情報

上記の事情をふまえて，カタログ上には，ハードウェアに関しては性能（CPU のサイクル時間や演算時間，周辺機器の速さなど）の他に，特に周辺機器の特長的な機能や利用可能時期（プログラムのサポートを含めて）などが，また，ソフトウェアに関しては，言語処理プログラムや応用プログラムではその種類と規模および特長，制御プログラムについてはその機能と規模，また，それぞれの利用可能時期が示されることが望まれる. 最近のカタログにはシステム構成例がいくつかあげられていることがあるが，これらについては，その具体的な機能，能力が示されることが望まれる.

4.2 テスト・プログラム

実際にハードウェアが使用できる状態にあれば，テスト・プログラムを実行させてみて，時間を測定し，あるいは後述のモニタリングによって動作状況を調べることにより，その性能を評価することができる．このテスト・プログラムはつぎの3つに分けて考えられる[1]．

（1） Kernel

これは入出力動作は考慮せずに，主としてハードウェアの性能を評価するためのもので，オペレーティング・システムの効果についてのデータはとれない．

（2） Benchmarks

それぞれのプログラミング言語でプログラムされた実用のプログラムで，これを実行させることによって対象とする機械の評価を行なおうとするもので，入出力装置や2次記憶などの評価にも役立ち，またソフトウェアの評価にも役立つ．すなわち，一連の Benchmark をオペレーティング・システムの制御の下に実行させることによって，マルチプログラミングの効果を調べたり，ジョブの型やジョブ・ミックスの影響を知るのにも役立つであろう．このためには可成り沢山のプログラムを用意する必要がある．

（3） Synthetic Program

システムの性能評価の標準となるようなプログラムという意味で考えられたもので[4]，すでに作られているプログラムでなくてもよいが，広い範囲の仕事を行なわせる必要がある．たとえば，計算に重点をおく部分，入出力に重点をおく部分などから成り立つことになる．

4.3 モデルの解析

システムを適当にモデル化して解析的方法によってその動作を究明しようとするものであり，対象とするシステムは必ずしも実在する必要がない．したがって，新しいシステムの開発の際などには，種々のモデルを作り，解析的方法によって評価が行なわれ，設計の参考に供されるのがふつうである．

また，実在するシステムに対しても，解析的方法によれば，システムの動作の大局的な把握が可能となり，パラメータの変化に伴う動作状況の変動などを知るのには極めて有効である．

しかし，システムのモデルの作成に当っては，対象とするシステムの特徴を正確に反映するようにしなければならない．特に，実在していないシステムに対しては種々のパラメータを仮定してモデルを作成することになるが，この仮定の適否が結果に重大な影響を与えることに十分注意しなければならない．

（1） 数学的解析法

システムを数学的なモデルとして近似的に表現して解析しようとするのである[5)~7)]．種々のモデルが考えられるが，待ち行列の理論によるものが多い[8),9)]．

（2） シミュレーション[10),11)]

解析的な手法で解くことが困難な場合に適用される有力な方法であるが，精密なモデルを作るとシミュレーションを行なうのに著しく時間がかかるのが問題である．そこで，要点を精密に表現し，余り重要でないところは簡略化してモデルを作ることが必要である．

実在するシステムの場合には，実測したデータを用いてシミュレーションを行なうことができ，実測できないようなシステムの細部の動作を明かにすることも可能になる．

また，新たに計画中のシステムに対しては，シミュレーションによって動作状況を予測，評価して，システムの最適設計に役立たせることができる．

4.4 モニタリング

実際のシステムを動作させておいて，その動作状況を観測し，性能評価に役立つデータの蒐集を行なうのである．システムの性能に影響する隘路を見出し，システム構成の変更を行ない，性能向上をはかる場合などには最も有効な手段である．また，計算機システムの動作状況が明らかになるので，システムの運用条件の改善に役立たせることも可能であろう．

モニタリングに当って注意すべきことは，モニターすることによる妨害のため実際の動作状況を乱し，性能に影響を与えないように注意することが必要である．モニタリングの手段としてはハードウェア的に測定を行なう方法とソフトウェアによる方法の2つに大別される．

（1） ハードウェア・モニタリング

ハードウェアによるモニタリングはシステムの動作に影響を与えることなく測定が行なわれる点に特長があるが，測定のための装置が必要であり，また，測定される項目が限定されるなどの欠点がある．

ハードウェア・モニタリングの方法としては，つぎのようなものがある．

a） 表示ランプを利用する方法[12)]

コンソールなどに出ている表示ランプの点滅を検出し，その出力を利用して，CPU やチャネル装置などの利用率を測定しようとするものである．この場合，光電素子を用いて光学的にランプの点滅を検出すれ

ば，ハードウェアの内部に接続するものがないので，ハードウェアに対しては全く影響を与えることはないが，測定の精度は良くない．ランプの接続端子に結線して出力を取り出せば，精度の良い測定を行なうことができる．

b） カウンタを利用する方法[9]

ハードウェアの内部から適当な信号をとり出して，この信号によって，パルス発生器からのパルスをゲートして，これをカウンタで計数することにより，その信号の継続時間を測定する方法である．

c） ハードウェア・モニタ[13)~17)]

上述のようにカウンタを利用するのであるが，計算機システムのハードウェアの測定のために，幾組かのカウンタの他に結合のためのプローブ，ケーブルなどを組合せて装置にまとめられたものがある．

特定のハードウェアを対象にして設計された専用のものから，ある程度汎用性をもったものまであり，その機能も，カウンタによる計数結果を単に記録するだけのもの，内蔵されている論理回路により簡単な処理を加えて表示したり記録したりするもの，専用の処理装置を用いて，あるいは汎用の小型計算機と結合することによって測定結果を処理して表示あるいは記録するものなど種々のものがある．

これらのハードウェア・モニタを利用する際には，これを結合することによってハードウェアの動作に悪影響を与えないようにすること，モニタの応答速度が測定されるハードウェアの動作速度より速いこと（たとえば，モニタの中の回路やカウンタが測定されるハードウェアのクロック周波数に対して十分応答できるものでなければならない）などに注意して目的に合った装置を選ばなければならない．

（2） ソフトウェア・モニタリング

ハードウェア・モニタリングではハードウェアの各部分の動作状態を知ることはできるが，どのプログラムがどれだけのリソースを使用してどういう状態で動作しているかというような処理状況を知ることはできない．このようなプログラムの動的な振舞いについては制御プログラムの管理する各種の制御表などの内容を見ることにより，また，プログラム・モジュールの動きを観察することによって，その状態についての情報を得ることができる．

ソフトウェア・モニタは対象とするシステムの中に測定プログラムを入れておき，これによって上述の状態を観測しようとするものである．すなわち，適当な時間毎に制御表などの内容を読み出すことによって動作状態に関する情報のサンプルを得ること，主記憶上の指定された番地が参照されたことを知ることによってプログラム・モジュールの動きを追跡することなどが考えられる．

一般の計算機では記憶の保護が行なわれているので，これらの情報をとり出すためには，測定プログラムを制御プログラムの一部としてスーパバイザ・モードで動作させるか，あるいは，読み出そうとする記憶部分の保護を解くか，などの手段が必要になる．

また，測定プログラムが動作するため，観測されるシステムの状態を乱すことになる．特に，精度よく観測しようとしてサンプリング周期を短くすれば，その影響は大きくなり，また，オーバヘッドも増大することになる．したがって，システムの動作状態に余り影響を与えず，オーバヘッドも余り大きくならない範囲で必要な情報を得ることが望まれる．

こうして，データを蒐集するだけでなく，動作状態を明らかにし，評価に有効な形の情報とするためのデータ処理も行なわれねばならない．このために蒐集したデータを記録しておき，後刻一括して処理する方法がとられるが，場合によっては，そのデータを結合された別の小型計算機に送ってここで直ちに処理してその結果を表示することにより，動作状況を時々刻々知ることができるようなものもある[18]．

また，別の計算機を結合して，これによって対象とするシステムの記憶内容を直接読み出す方法も考えられている[19]．

5. むすび

システムの評価法は上述のように種々あるが，それらをまとめてみると第1表のようになる．

未完成のシステムでは実測はできないが，実在するシステムでは実測したデータを基にして解析し，必要があれば，システムの改良を行なってさらに測定し，そのデータの解析を行なうという繰返しによってシステムの最適化をはかるのが望ましい．

また，計算機システムはその利用形態あるいはジョブの種類や性質によって，その動作状況が著しく変化するので，随時システムの動作状態を監視し，最適状態で動作させるのが望まれる．そのためには制御プログラムの中にソフトウェア・モニタを内蔵させて，随時動作状態を測定し，その結果を分析して，必要があれば動作条件の変更を行なうようなことを考えるべき

第 1 表

評価の方法		適用可能時期*	利用の容易さ**	精度**	経済性**	評価の目的に対する適用可能性		
						機種の選定	性能の改善	新システムの開発
カタログ・データ		A	5	1	5	○	×	×
テスト・プログラム	Kernel	B	4	目的により 2～5	4	○	×	×
	Benchmarks	C	4		3	○	○	×
	Synthtic Program	C	3		3	○	○	×
モデルの解析	数学的解析法	A	3	2	4	○	○	○
	シミュレーション	A	2	3	3	○	○	○
モニタリング	ハードウェア・モニタリング	C	1	5	1	同じ構成のシステムが利用可能の場合適用可	○	×
	ソフトウェア・モニタリング	C	2	4	2		○	×

* A：未完成時より，B：ハードウェア試作完了時より，C：システム完成時
** 5 段階に分け下位より 1, 2, ……, 5 とつけた．
† ○印：適用可能　×印：適用不能

であろう．

さらに，新しい技術の進歩，たとえば，データ通信関係，ファイル関係の機能の発展は必然的にシステムの巨大化，複雑化をもたらすことになるであろう．これを効率よく運営し活用するためにはさらに評価方法を進歩させ，その結果をシステム設計に反映させて，効率のよいシステムを完成するよう努力すべきである．

参考文献

1) H. C. Lucas: Performance Evaluation and Monitoring, Computing Surveys, vol. 3, no. 3, pp. 79-91, (Sept. 1971).
2) 石田晴久：ギブソン・ミックスの起源について，情報処理，vol. 13, no. 5, pp. 333-334, (May 1972).
3) 藤井，鈴木：オペレーティング・システム，pp. 265-277，産業図書，(1970).
4) W. Buchholl: A synthetic job for measuring system performance, IBM Syst. J. vol. 8, no. 4, pp. 309-318, (1969).
5) V. L. Wallace, R. S. Rosenberg: Markovian Models and Numerical Analysis of Computer System Behavior, Proc. AFIPS, vol. 28, SJCC '66, pp. 141-148.
6) A. L. Scherr: An Analysis of Time Shared Computer Systems, MIT Pr. (1967).
7) D. P. Gaver: Probability Models for Multiprogramming Computer Systems, J. of ACM, vol. 14, no. 3, pp. 423-438, (July 1967).
8) 田中穂積：並列循環待ち行列を用いたオンラインシステムの解析，電子通信学会論文誌，vol. 53-c, no. 10, pp. 756-764, (1970).
9) 池田克夫：大型電子計算機システムの効率測定と循環待ち行列の理論による解析，情報処理，vol. 12, no. 9, pp. 568-576, (1971).
10) L. R. Huesmann, R. P. Goldberg: Evaluating computer systems through simulation, Computer J. vol. 10, no. 2, pp. 150-155, (Aug. 1967).
11) M. H. MacDougall: Computer System Simulation: An Introduction, Computing Surveys, vol. 2, no. 3, pp. 191-209, (Sept. 1970).
12) H. Stang, P. Southgate: Performance Evaluation of Third-generation Computing Systems, Datamation, vol. 15, no. 11, pp. 181-190, (Nov. 1969).
13) 富岡他：汎用コンピュータ・パフォーマンス・アナライザ，昭 46 情報処理学会第 12 回大会予稿，pp. 369-370, (1971).
14) 箱崎他：ハードウェアモニタ (SYDAS) によるシステム性能評価，昭 46 情報処理学会第 12 回大会予稿，pp. 371-372, (1971).
15) 西本他：超高性能電子計算機のハードウェアモニター，昭45情報処理学会大会予稿，pp. 367-368, (1970).
16) C. E. Kohn: The Evaluation of a Large Computer System, INFOR, vol. 9, no. 2, pp. 140-147, (July 1971).
17) R. A. Aschenbrenner, L. Amiot, N. K. Natarajan: The neurotron monitor system, Proc. AFIPS, vol. 39, FJCC '71, pp. 31-37.
18) R. Sedgewick, R. Stone, J. W. McDonald: SPY-A program to monitor OS/360, Proc. AFIPS, Vol. 37, FJCC '70, pp. 119-128.
19) 北川他：Computer Complex による OS の Performance Monitoring，昭 45 情報処理学会予稿，pp. 93-94, (1970).

(昭和 47 年 8 月 8 日受付)