（1981. 6. 12）

分散型計算機システムの性能予測ツールの開発

本山博司，大町一彦（日立システム開発研究所）

1．序論

分散型計算機システムがいよいよ普及しようとしている。分散システムには大別して地理的分散システムと機能分散システムとがあるが、いずれにしても、分散システム設計者は複数のノードとそれを結ぶリンクをどのように構成するか、その構成によって所期の性能が出るか否かを決定し、判定しなければならない。しかもその判定は未だシステムが稼動しないうちに行なう必要がある。すなわちシステム設計者はどうしても性能予測をしなければならない。この事自体はシングルノードの計算機システムの設計者にとっても同様ではある。しかし分散システムはシングルノードのシステムに比べてノードの数は複数であり、しかもメッセージがノード間を渡り歩くなど従来に無い要素を加わって格段に複雑になって来た。そこで性能評価ツールが必要になるのだが、分散システム用のしかも設計初期段階での性能評価ツールが、シングルノードの詳細を極めたシミュレータと同じ発想で造られてよいだろうか。設計初期段階では、後期段階と異なり、回線バッファ長や、こまかな通信制御方式上の相違，CPU走行ステップ数の微細な変化等は不明である。設計初期段階ではノードとリンクの接続のしかたや、ノード内の各資源の基本性能が、これでよいのか否か知りたく、もしよくないのであれば、別のシステム構成ではどうか等々、茫洋とした設計選択枝の中からできるだけ早く、しかも誤またず正しい解を見出したいのである。したがってこの設計段階で用いるべきツールは、少々精度は落ちても、少数の基本的設計データのみ入力すれば、素速く結果をユーザに示せるようなものでなければならない。

筆者等は最近、以上のような段階における性能設計のあり方を吟味し、それに適したツールを開発し設計者の支援に供しているので、ここに報告したい。

なお本報告は、第22回大会の報告[7]を発展させたものである。

この報告では、まず第2章で、分散システム性能評価における問題点を挙げ、その解決策について既存の研究成果を検討する。第3章で、既存成果で不十分な点に関する解決策を示し、その効果に言及する。第4章では、仮想的かつ現実的な分散システムに関し実際の評価例を示す。第5章では結論と今後の課題を記して全体を終る。

2．分散型計算機システム評価の問題点と既存の研究成果

2.1 評価時間の長大化

性能予測の手法にはシミュレーションと解析手法の二種あるが、モデリング能力が高い点から、従来多くシミュレーションが行なわれてきた。シングルノードの計算機システムをシミュレートする際、応答時間その他の評価項目の値が十分安定するまでには、通常数10分のCPU時間を要する。従ってこれと同程度の細かさで分散システムをモデル化し、シミュレーションを実行すると、少なくとも，このノード数倍は時間がかかる。ちなみに，ノードあり20分を要するモデルで，10ノードある分散システムを評価するには、少なくとも3時間20分のCPU時間必要である。

この長さは、評価を1ケースだけ行なう場合には、禁止的ではない。しかし設計時には、多数のシステム構成代替案を評価したいのが実情であり、この場合は評価時間を十分短かくしておかなければならない。

シミュレーション時間を短縮する方法として、モデルの分解(decomposition)がある。分解の考え方はCOURTOIS [1]により提出された。[1]では分解を解析的手法について適用しているが、基本的にはシミュレーションモデルと分解してよい。分解し、上下二段階にしたモデルをさらに高速化するため、一方を解析モデルで評価する方式、いわゆるハイブリッド方式がある。SCHWETMAN [2]は、単純なハイブリッドモデルについて、その方式の有効性を示し、CHIU and CHOW [3]、木下[8]は、具体システムについて、ハイブリッド方式で評価し、容量計画(capacity planning)に適用し効果を上げた。しかし分散システム性能評価を行なうには、さらに高速化が必要であり、本報告ではこの点につき新しい工夫をした。

2.2 分散システム特有のトランザクション動作のモデル化

分散システムでは、あるノードに入ってきたトランザクションを全てそのノード内で処理するとは限らない。処理の一部は、他ノードで行なわれ、その結果を用いて、後の処理を行なう。このような動作は地理的分散システム、機能分散システム双方に見られるが、集中システムには無い。この点が自由にモデル化できるしくみを、設計者に提供しなければならない。またノードとリンクの接続法(ネットワークトポロジー)も、簡単に定義できなくてはならない。

以上の点を考慮したコンピュータネットワーク専用のシミュレータには、田中、中村[4],[5]およびCHIU and CHOW[3]がある。これらは、性能評価の力点を通信系に置き、各通信制御方式個有の動きについて十分な精度でモデル化している。このためネットワーク定義は十分に表現できるが、ノード内での処理については大巾に簡略化している。本報告では力点をネットワークからノード内処理の方にかえ移し、通信系を上記方式より簡略化し、ノード内処理及び、トランザクション内部でのノード間渡り処理について、より詳細化した。

3. 問題点の解決法とその効果

3.1 評価時間の短縮

(1) 利用率の事前チェック

システム内に1個でも利用率が100%を超えるものが有れば、シミュレーションは安定せず、待ち行列内のトランザクション数は次第に増加する。従って特にこのような状態における過渡応答を知りたいのでないかぎり、シミュレーションを行なう意味は無い。このようなムダを避けるため、シミュレーションに入る前に、利用率の計算を行なう。利用率は、トランザクションの到着率入と、トランザクション当りの資源使用量が与えられれば開ネットワークについては、原理的に簡単に求まる。しかしながら、後述するようにトランザクションの内部が複数個の処理に分かれ、その各々について資源使用量を指示していく場合には、利用率の計算は、まず各々の処理毎に行ない、次にそれらの総計を行なうという2段階を経なければならないので、複雑である。従って利用率の計算をツール内で自

動的に行なう事には意味がある。

3・2 モデルの階層化，ハイブリッド化，静的インタフェース化

モデルの分解により、上下2段階に階層化する事が有効である事は、SWETMAN[2]で示されている事は既に述べた。[2]ではシステム資源を主メモリ，磁気テープ装置等の「長期間資源（long-term resource）」と、CPU、チャネル等の「短期間資源（short-term resource）」とに分け、前者をシミュレーションで，後者を解析的に評価している。その理由は短期間資源は時間当りの使用回数が多く、これを解析化すればより評価時間短縮効果が高いからである。我々も基本的にはこの方式を取る事にした。CHIU and CHOW [3]も同じくこの方式を実施している。

しかしながら、[2]，[3]とも、階層間の情報の受授が動的である点が、報告者等の静的インタフェース方式と異なる。以下この点について説明しよう。

(a) 動的インタフェース

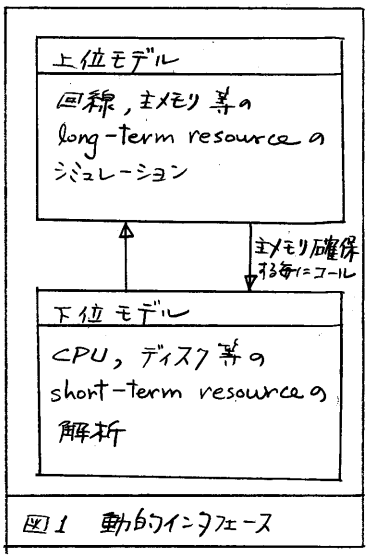


図1 動的インタフェース

上位モデルと下位モデルの動的インタフェースとは、上位モデル中でトランザクションが主メモリの占有／解放毎に下位モデルを動作させ、そのタスクの処理終了時刻を予測し、その予測時刻でタスク終了事象を発生させるよう上位モデルがシミュレートする方式を言う。すなわち主メモリ占有時間を、トランザクション毎に評価するわけであるから、上位モデルと下位モデルは交互に頻繁に動作する。下位モデルの機能を数式で書けば、次のようになる。

$$\Pi = R(N) \quad \cdots\cdots \quad ①$$

但し、
$\Pi = (T_1, T_2, \cdots, T_i, \cdots, T_m)$
T_i：トランザクションタイプ i の処理終了時刻
$N = (N_1, N_2, \cdots, N_j, \cdots, N_m)$
N_j：トランザクションタイプ j の主メモリ内多重度
R：NよりΠを評価する機能

(b) 静的インタフェース

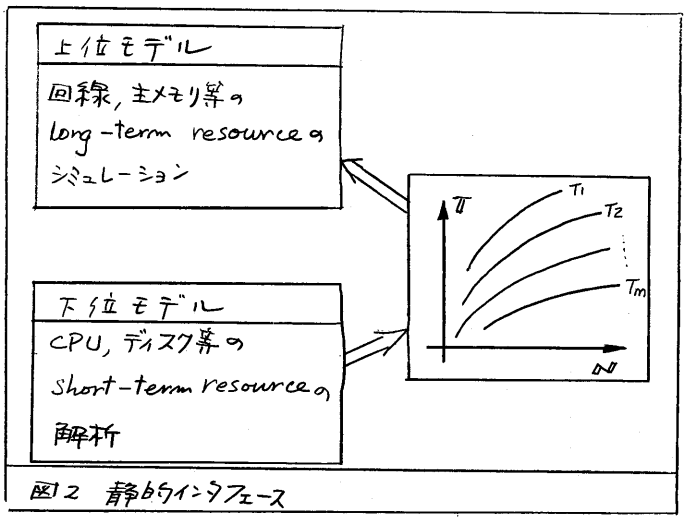


図2 静的インタフェース

静的インタフェースとは①式の$\Pi = R(N)$を、毎回計算するかわりに、左図のように計算結果をテーブルの形で持っておく方式を言う。従って下位モデルは、このテーブルを作るために、1回だけ動作すればよく、上位モデルはこのテーブルを引くのみで、タスク終了時刻が予測できる。

静的インタフェースを取る事により、下位モデルの走行回数は、次のように短縮された。下位モデルとして BUZEN [6] のセントラルサーバモデルを用いた場合、I/O 台数12～50, 多重度 2～50 における CPU 処理ステップ数実測値を図3に示す。これはトランザクションタイプが1種類のみの場合である。但し、値は I/O 50台, 多重度 12 のときの走行ステップ数を1単位として表わした。以下 I/O 50台, 多重度 12 のケースについて考えるとする。いま、シミュレーションを行い、全部で N 回トランザクションが、主メモリを占有したとすれば、メモリの解放も同様に N 回だから計 2N 回確保／解放される。動的インタフェース方式では、この場合

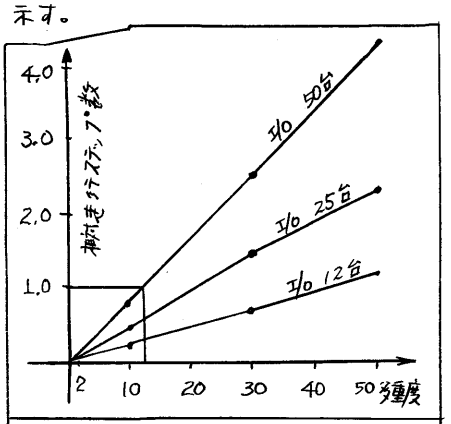


図3 セントラルサーバモデルの CPU 走行ステップ数

$$2 \times N \times 1.0 \text{ 単位}$$

の走行ステップ数が下位モデルで費されるのに対し、静的インタフェース方式で費されるのは、

$$1.0 \text{ 単位}$$

のみであり、通常 N は $10^3 \sim 10^5$ はあるので、その差は 2N 単位としてよい。すなわち静的の方が、動的より 2N 単位ステップ数だけ走行ステップ数が短かくてすむ。しかし静的インタフェース方式はテーブルサイズに限界があるため、トランザクションタイプが増え、各タイプ内の多重度が大きくなると、実施できない

次に上下モデル間のインタフェースは上記と同じように静的ではあるが、下位モデルがセントラルサーバモデルのような解析手法ではなく、GPSS で組んだシミュレーションモデルにした場合を想定し、下位モデルの走行ステップ数を予測してみよう。モデルは上記の場合同様 I/O 50台, 多重度 12 とする。1トランザクション当り平均 P 回の I/O 要求を出すとすると、CPU は P+1 回使用する事になるので計 (2P+1) 回資源を使用する。一般にシミュレーションでは資源の利用率により、資源当りシミュレーションに要する走行ステップ数は異なる。GPSS による 3M／M／1 待行列の走行ステップ数を実測すると、図4のようになる。

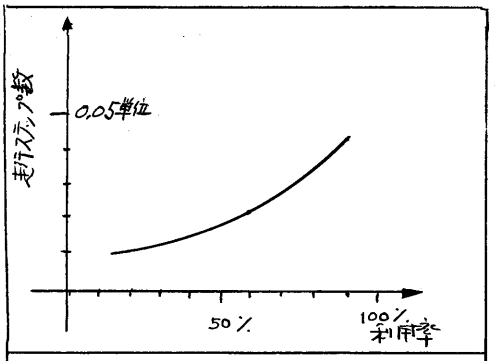


図4 GPSS 1資源当り走行ステップ数

いま、CPU と I/O の利用率がともに 60% であると仮定すると、図4より走行ステップ数は 0.02 単位である。また、シミュレーションは m 個のトランザクションについて行なうとすると、結局、下位モデルのシミュレーションに要する走行ステップ数は、

$$(2P+1) \times 0.02 \times m \quad \text{単位ステップ}$$

となる。

ここで上位のシミュレーションモデルの評価に要する走行ステップ数を、トランザクション当り、S単位ステップとして、以上の結果を図5にまとめる。

上位モデル	VS（単位ステップ）		
		チェック	
下位モデル		解析	シミュレーション
インタフェース	静	方式Ⅰ 1.0	方式Ⅲ (2p+1)×0.02×m
	動	方式Ⅱ 2V	———

単位：「単位ステップ」すなわち下位モデルのステップ数を1.0とした時の比
V：全完了トランザクション数
S：上位モデルの1トランザクション当りの走行ステップ数

図5. 各方式の走行ステップ数比較

全体の走行ステップを各方式について比較してみよう。便宜上図5の如く各方式を方式Ⅰ，Ⅱ，Ⅲと呼ぶ。

（a）方式Ⅰと方式Ⅱの比較

方式Ⅰのステップ数はVS＋1.0，方式ⅡはVS＋2Vであるから、方式ⅠとⅡの比は、下のようになる。

$$(Ⅰ)/(Ⅱ) = (VS+1.0)/(VS+2V) \simeq VS/(VS+2V) = S/S+2 \quad \cdots\cdots ③$$

すなわち，S≃1.0で、上位モデルの走行ステップ数が下位モデルと同程度ならば方式Ⅰは方式Ⅱの1/3の走行ステップ数で評価を完了できる。逆に，上位シミュレーションモデルで複雑なシミュレーションを行ない、S≃10となると、③は10/12＝83%となり、両者の差はほとんど無くなる。しかしながら，常にⅠの方がⅡより速い。

（b）方式Ⅱと方式Ⅲ

上記（1）における比較と同様にして、

$$(Ⅲ)/(Ⅱ) = (VS+(2p+1)\times0.02\times m)/(VS+2V) = (S+(2p+1)\times0.02\times m/V)/(S+2) \quad \cdots\cdots ④$$

ここで、上位シミュレーション，下位シミュレーションが安定するに要するトランザクション量V，mが、ほとんど同程度（通常10^3～10^5）とすると、

$$④ = (S+(2p+1)\times0.02)/(S+2) \quad \cdots\cdots ⑤$$

となり、トランザクション当りI/O使用回数 p＝24.5を境界にして、pがこれより小さければ方式Ⅲの方が走行ステップ数が短かくてすむ。方式Ⅲをつかって、下位モデルを、さらに複雑にしていき、1トランザクション当りの走行ステップ数が、2単位を超えれば，方式Ⅱより Ⅲの方が走行時間が長くなって（ま）が、そのかわりシミュレータの高いモデリング能力をひき出す事ができる。

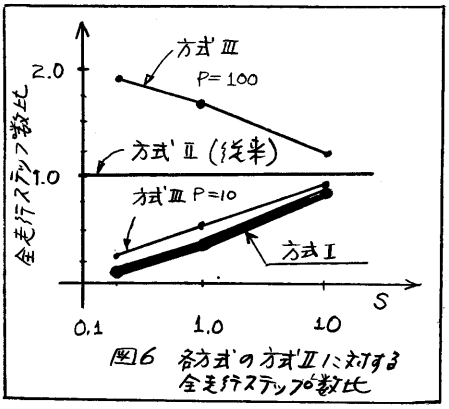


図6 各方式の方式Ⅱに対する全走行ステップ数比

③，④式を、上位シミュレーションモデルの1トランザクション当りステップ数Sについてプロットしたものを図6に示す。図6は方式Ⅱ（従来のハイブリッド方式）を1.0として、他の方式を比較したもので、1.0より上にあれば従来方式より遅く，下にあれば従来方式より速い事を意味する。

3・3 分散特有動作のモデル化支援機能

(1) Service Request = Function + Message (図9)

端末から入力される オンライントランザクション，あるいはTSSのコマンド あるいはバッチジョブ等、システムに対する処理要求は様々な名で呼ばれている。これらを以後Service Request，SRと総称する。ツール内であつかう処理要求は、全てSRであり、その他の形式は無い。分散システムでは、1個のSR内で、複数個のノードにまたがって処理するものが必らずある。(そうでなければ、分散システムではなく孤立したシステムが単に複数個有るにすぎない) このようなSRの性格をモデル化しやすいように、"Function"を作った。FunctionはSRの内部に有り、SRの構成要素である。FunctionはSRの内部に複数個あってよく，各Functionは、必らず1個のノード内で処理される。1個のFunctionのうち一部があるノードで処理され，他の部分が他のノードで処理されるような事は無い。しかし，SR内の複数のFunctionが、別々のノードで処理されてよい。Functionに入力されたり、Functionが出力したりする情報をMessageと呼ぶ。Functionは必らずMessageを入力し、必らず出力する。

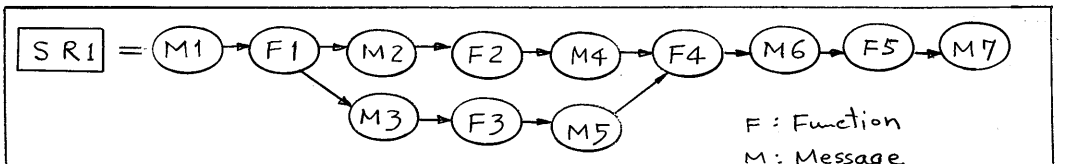


図7 SR = Function + Message

(2) Function Class / Region

分散システムを構成する各々のノードは、システムに入って来る処理要求の全てを処理できるわけではない。個々のノードで処理できる業務内容は決まっている。この事をモデル化しやすいように、各ノードをグループに分け、それらのグループ内のノードはどれも、共通のFunctionを処理できるが、それら以外のFunctionは処理できないものとした。このグループを"Functional Class"と呼ぶ。(図8)

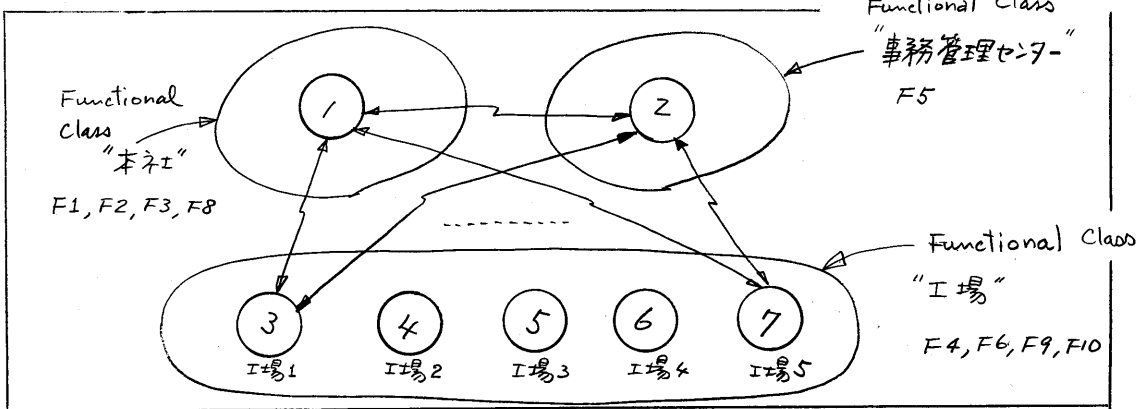


図8 Functional Class

また、ファイルが分散している場合など、そのSRが入って来たノード内のファイルを多く参照し、他ノードのファイルは少ししか参照しないといった事がよくおこる。例えば、全国に分散したデータベースで、大阪でおきた検索のトランザクションは、ほとんど関西圏内のファイル参照ですみ、少数が東京のファイルを参照するであろう。このような「ファイル参照の地域性」をモデル化するため"Region"の考え方を導入した。1つのRegionは、ノードの集合で、全ノードは必らずどれかのノードに属するものとする。各SR内のFunctionは、SRが到着したノード内で処理される率(ローカル率)、SRが到着したノード以外だが同一Region内のノードで処理される率(Region率)を指定しておく。シミュレーターは、この率に従って、Functionの処理を各ノードに配分していく。このとき処理すべきノードは、当然そのFunctionを処理できるFunctional Classに属していなければならない。

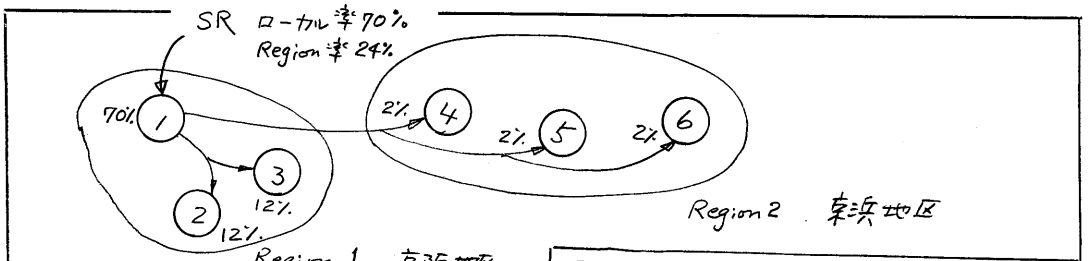


図9 Region

4. 評価例

以上述べてきた技法を用いて、仮想的な分散システムについて、性能評価を行なってみよう。システムは図8のように、本社、事務管理センター、工場とから成っている。本社ではオーダエントリーのトランザクションを受けつけ、それにともなうマクロ生産計画を立て、工場に生産を指令する。工場は指令にもとずきミクロ生産計画を立て、部品発注をし、生産を開始する。また、製品が完成に到るまで、その状態管理を行なうため、工場は独自の製品トラッキングを行なう。

SRは全体で12種類あり(SR1～SR12)そのうち、"オーダエントリ"と呼ばれるSR5は、次のような内訳となっている。

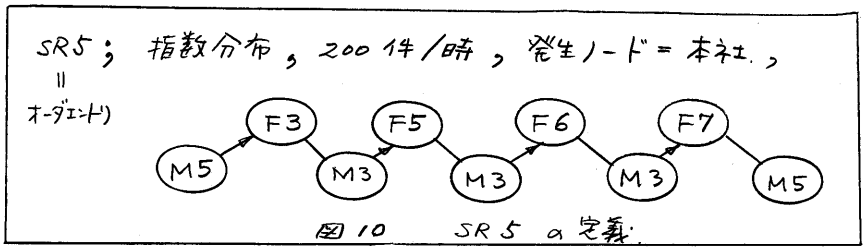


図10 SR5の定義

さらに、SR5内のMessage, Functionの定義をする。

Function;

F3 ; マクロセイサンケイカク, 処理ノードのFunctional class = 本社
使用資源 = CPU. 1個, 300 Kstep, 1点分布
DISK1, 10個, 10回アクセス, 1点分布.
メモリ 256 KB

F5 ; ジムカンリセンターレポート, 処理ノードのFunctional Class = 事務管理センタ
使用資源 = CPU 1個, 500 K step, 指数分布.
DISK2, 15個, 30回アクセス, 1点分布.
メモリ 512 KB

Message;

M5 ; メッセージ長 200バイト, 1点分布

M3 ; メッセージ長 4Kバイト, 1点分布.

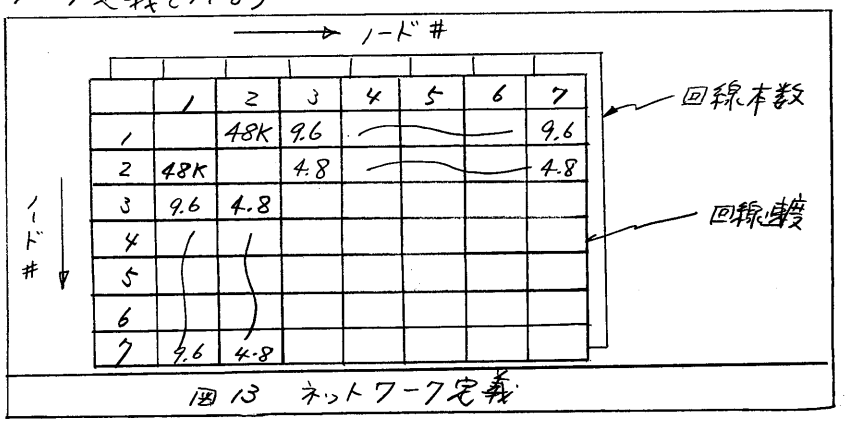
図11 SR5内のFunction, Message定義

次に、各ノード内にある資源の定義及びFunctional Classの定義を行なう。

ノード1; ホンシャ, Functional Class = ホンシャ, ノードRegion = 1,
CPU 6MIPS, メッセージ処理CPU負荷 = 3 Kstep.
DISK1, 30台, 30 msec/access
DISK2, 5台 50 msec/access

図12 ノード1(本社)の定義

最後にネットワーク定義を行なう



	1	2	3	4	5	6	7
1		48K	9.6				9.6
2	48K		4.8				4.8
3	9.6	4.8					
4							
5							
6							
7	9.6	4.8					

図13 ネットワーク定義

以上のような入力データに対し，以下のような結果を得た。（全CPU時間 8分10秒）

2. LINK UTILIZATION (%)

(ORIGIN NODE)

DS.N	1	2	3	4	5	6	7
1	0.0	4.179	2.480	1.380	1.412	1.708	1.566
2	13.901	0.0	4.118	3.980	3.840	3.261	3.095
3	23.170	34.804	0.0	0.0	0.0	0.0	0.0
4	26.169	28.775	0.0	0.0	0.0	0.0	0.0
5	16.461	14.168	0.0	0.0	0.0	0.0	0.0
6	16.456	28.733	0.0	0.0	0.0	0.0	0.0
7	17.613	32.017	0.0	0.0	0.0	0.0	0.0

表1
DYNAMIC MODEL（上位モデル）による回線利用率評価結果

表2 STATIC MODELによる利用率事前 check（100%以下）OK

3. RESOURCE UTILIZATION BY RESOURCE CLASS

RESOURCE CLASS (%)

NODE	1	2	3	4	5
1	10.514	33.808	2.917	36.808	0.039
2	1.897	13.250	0.0	0.0	0.007
3	60.331	0.810	48.276	71.425	1.169
4	60.331	0.810	48.276	71.425	1.169
5	60.331	0.810	48.276	71.425	1.169
6	60.331	0.810	48.276	71.425	1.169
7	60.331	0.810	48.276	71.425	1.169

表3 DYNAMIC MODELによる利用率評価結果

1. RESOURCE UTILIZATION BY NODE (%)

(RESOURCE CLASS)

NODE	1	2	3	4	5
1	15.372	34.800	4.500	70.500	3.636
2	2.016	14.400	0.0	0.0	0.322
3	57.075	1.200	53.100	59.850	28.312
4	55.500	0.750	47.700	62.550	28.186
5	56.925	1.050	48.450	64.350	23.119
6	60.375	0.750	46.950	73.050	27.591
7	58.200	1.050	46.950	68.400	27.451

CPU　　DISK

表4

完了個数　平均応答時間　応答時間標準偏差

3. SERVICE REQUEST PERFORMANCE DATA

SR.#	SERVICE REQUEST DESCRIPTION	NUMBER COMPLETED	MEAN TURNAROUND TIME (SEC)	VARIANCE OF TURNAROUND TIMES
1	ｾｲｾﾋﾝﾊｯﾁｭｳ	41	0.474	0.088
2	ﾊﾝﾊﾙ"ｲ ﾄｳｹｲ/ﾊﾝﾊﾙ"ｲ ｹｲｶ	41	0.474	0.088
3	ｾｲｻﾝｹｲｶｸ	41	1.440	0.325
4	ｸﾐﾀﾃ ｼ"ｭﾝｼ"ｮ ｹｲｶｸ	41	1.804	0.318
5	ﾌ"ﾋﾝ ﾊｯﾁｭｳ	131	4.304	0.912
6	ｾｲｾﾋﾝ ｽﾃｰﾀｽ	241	2.114	1.424
7	ﾋｷｱﾃ	2191	3.924	4.132
8	ｼｭｯｶ ｼｼ"	1556	0.968	0.032
9	ｾｯｹｲ ﾃﾞｰﾀﾃﾝｿｳ ｳ	41	1.609	0.232
10	ｹﾞﾝﾀﾝｲｾｯﾃｲ1/ｷｶ	2214	0.504	0.024
11	ｹﾞﾝﾀﾝｲｾｯﾃｲ2/ｷｶ	531	4.106	1.070
12	ﾌ"ﾋﾝﾃﾞ"ｰﾀﾍﾞｰｽ ﾒﾝﾃﾅﾝｽ	241	2.133	0.914

DYNAMIC MODEL（=上位モデル）によるSR応答時間評価結果

5. 結論

分散型計算機システムの性能評価ツールを開発した。この種のツールの課題は性能評価時間の短縮と、分散特有のモデリング機能支援の充実にある事から発し、時間短縮のためにSCHWETMANのハイブリッド方式をさらに高速化した方式を採用した。モデリング支援のため、トランザクションを複数の「ファンクション」と「メッセージ」で構成し、さらにノードをFunctional classと呼ばれるグループに分け、各々のFunctional Class内のノードが共通のファンクションを処理できるように定義する機能を設けた。これにより、トランザクション内の処理を、複数ノードで行なうようなモデルが簡単にできるようになった。

現在のところ、実稼動の分散システムの実測値により、本方式の結果の検証をしていない。また、ここで採用したハイブリッド方式の評価時間の測定と組織的には行なっていない。今後この二点を課題とした。

6. 参考文献

[1] Courtois, P. Decomposability, instabilities, and saturation in multiprogramming systems. Comm. ACM 18, 7 (July 1975), 371-377

[2] Schwetman, H. D.. Hybrid Simulation Models of Computer Systems. Comm. ACM 21, 9 (Sept. 1978) 718-723

[3] Chiu, W. and Chow, W-M, A performance models of MVS. IBM Sys. J. 17, 4 (1978) 444-463

[4] 田中, 通信ネットワーク計画・性能設計支援ツール SPACNET-Pの開発 第23回 情報処理大会 (予定)

[5] 中村, 通信ネットワークの性能設計 — SPACNET-Pによるアプローチ 第23回 情報処理大会(予定)

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

[7] 本山, 大町 ハイブリッド手法を用いた分散システム性能評価手法 第22回 情報処理大会

[8] 木下, 石沢 解析手法を取入れた計算機性能評価シミュレータの開発 第22回 情報処理大会