マルチメディア通信と分散処理 29－6
（１９８６．５．１５）

分散処理システム評価シミュレータを用いた具体的問題の適用例

下条真司　山口　英　宮原秀夫　高島堅助

大阪大学基礎工学部情報工学科

著者等は分散処理システムをハードウェア／ソフトウェア両面から統合的に性能評価を行うシミュレータSEDS(Simulator for Evaluation of Distributed Systems)の開発を行っている。SEDSにおいては分散処理システムを構成する諸機器およびそれらの間の接続形態、さらに実現される分散型アルゴリズムとをFormと呼ばれる簡単な書式でシミュレータへ入力することができる。想定したハードウェア構成を有する分散処理システムの上に特定の分散型アルゴリズムを実装した際のシステムの性能評価を行うことができる。本論文では特に、SEDSにおいて簡単な入力方法を実現しているFormについて詳しく述べる．さらに、SEDSを実際の分散処理システムに適用した例としてジョブディスパッチ問題をSEDSでシミュレートした例を示す．

Simulator for Evaluation of Distributed Systems

Shinji shimojo　suguru yamaguchi　Hideo Miyahara　Kensuke Takashima

Dept. of Information and Computer Sciences, Faculty of Engineering Science
Osaka University, Toyonaka Osaka 560, JAPAN

We have developed a Simulator for Evaluation of the Distributed Systems (SEDS) at the aim of evaluating not only distributed system (DS) architectures but also distributed algorithm. In our simulator SEDS, Through using simple format "forms" defined in SEDS, we can describe both the hardware configulation of a DS which we want to simulate and the distributed algorithm implimented on it. In this paper, we mention the detail of the 'Form' and apply the job dispatching problem on the DS to SEDS. Through the simulation of the problem by SEDS, we show the availability and applicability of the SEDS to the wide range of the problem around the DS.

分散処理システム評価シミュレータを用いた具体的問題の適用例

Simulator for Evaluation of Distributed Systems

下条真司 Shinji Shimojo　山口 英 Suguru Yamaguchi　宮原秀夫 Hideo Miyahara　高島堅助 Kensuke Takashima

大阪大学基礎工学部情報工学科
Dept. of Information and Computer Sciences, Faculty of Engineering Science
Osaka University, Toyonaka Osaka 560, JAPAN

1．はじめに

半導体技術の進歩によりVLSIの実用が可能となり、CPUやコンピュータ周辺機器の高機能・低価格化が実現された．これにより、従来単一の計算機で行われていた様々な処理を、複数の計算機で分散処理することにより効率向上を図ろうとするいわゆる分散処理システムの開発が進められるようになった．

この分散処理システムの効率は分散処理システムを構成する計算機の能力、個数、接続形体などのようなハードウェアの構成面だけでなく、ジョブを構成するプロセスの分散法などソフトウェアの構成面にも大きく依存する．ここに、分散処理システムにおける性能評価の必要性が生じ、しかも、分散処理システムの性能を論ずる際には、ハードウェア面からの評価だけでなく、その上で分散処理されるソフトウェア面をも考慮に入れて性能評価を行わなければならない．

そこで筆者らは分散処理システムを総合的にしかも容易に評価する為のシミュレータSEDS(Simulator for Evaluation of Distributed Systems)を作成した[1]．本論文では特に、SEDSにおいて簡単な入力を実現するFormについて詳しく述べる．さらに、SEDSを実際の分散処理システムに適用した例としてジョブディスパッチ問題をSEDSでシミュレートした例を示す．

2．SEDSの特徴

SEDSは次のような特徴を持っている．

1）ハードウェア／ソフトウェア両面の評価が可能

SEDSは分散処理システムのハードウェアのみを評価するためのものではなく、また、ソフトウェアのみを評価するためのものでもない．ある構成の分散処理システムの上に特定の分散型アルゴリズムを実装したときのシステムの性能評価を行う．つまり、実際にそのシステムを構築することなく、様々の分散処理システム形態と分散型アルゴリズムを評価できる．

2）分散型シミュレータ

SEDSはUNIXが持つプロセス管理およびプロセス間通信の両機能を用いてUNIX上に並行プロセスとして実現されており、分散処理環境があれば容易にSEDS自身を分散型シミュレータへと拡張可能である．また、シミュレータをいくつかのプロセスの集合として構成することにより、各部の独立性を高め、保守性を上げることができる．

3）Form

SEDSでは分散処理システムのハードウェアからソフトウェアにわたる構成要素のうち、性能評価を行う際に重要な要素のみを切り出し、Formと呼ばれる形式で分類し、モジュール化している．これにより各Formを独立して指定できる．Formではあらかじめ利用者が指定できる項目がまとめられており、利用者は指定したい項目を埋めていくだけでよい．また、別のシミュレーションで作成したFormを利用することも可能である．

3．シミュレーション実行環境(SEEDS)

SEDSはシミュレーションにまつわる一連の作業を複数のプログラム群(Managers)で分担して行う（図1）．まず、SEDSにはシミュレーションの実行制御のためのパラメータ、分散処理システムのシステム構成およびその上で実現される種々のソフトウェアなどをすべてFormの形で入力する．このFormを管理し、ユーザーによる対話的なFormの作成・編集を支援するのがSEDS-I/O Manager(SEDS/IO)である．次に、SEDS-Parameter/Generation Manager(SEDS/PG)がI/O Managerより渡されたFormを解釈し、必要に応じてソース／オブジェクトライブラリーであるBasic Program Package（SEDS/PP）を参照しながら、実際のシミュレーションを行うExecutable Module（SEDS/EM）を生成する．同時にFormよりパラメータを抽出し、EMの入力として与え、シミュレーションを実行する．得られた実行結果はSEDS/IOによりグラフ出力などを行える．

これらのManager群はシミュレートする分散処理システムのモデルであるFormを利用者が簡単に記述し、シミュレーションを実行、実行結果の管理も容易に行えるような環境を作り出す．このSEDSによる統合化されたシミュレーション環境をSEEDS(Simuration Environment for Evaluation of Distributed Systems)と呼ぶ．

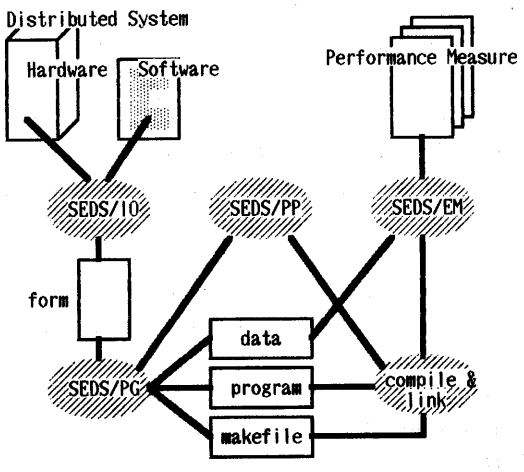

図1　ＳＥＥＤＳによる分散処理システムの評価

4　分散処理システムの要素

シミュレーションを行うためには対象とする分散処理システムを目的に従ってモデル化する必要がある．'SEDS'は性能評価を目的としているため分散処理システムから性能評価上重要な要素のみを基本的な構成要素として取り出している．利用者はこれらの構成要素を組み合わせることにより所望のモデルを構築する．これらの構成要素はハードウェア要素とソフトウェア要素の大きく二つに分かれる（図2）．

4．1ハードウェア要素

ハードウェア要素は分散処理システムを構成する物理的な機器とそれらを結合している様々な通信路からなっている．

・PE（Processing Element）

分散処理システム上で独立した一つの機器を表す．一つのPEは他のPEと独立して動作できる．PEはシミュレーションの目的により一つのプロセッサや一台の計算機、一つのVLSIチップなど様々な機器に対応する．各PEは複数のポート(port)を持ち、ポートに接続されたMediaを通して他のPEと接続される．PEは他のPEと一切の共有データを持たず、このportを通してのみ他のPEとデータのやりとりを行うことができる．

・Media

PE同士をポートを介して結合する通信路．Mediaでは複数のPEが共有する際の通信競合の管理、通信時間の確保などを行う．ただし、現在は通信方式及び通信路形態の種類別にシステムの標準Form(standard form)が用意されており、利用者はそのうちから選択すればよい．

4．2ソフトウェア

ハードウェア要素のうちPEの特性や実際の振舞いはソフトウェア要素としてPEごとに定義され、対応するPEから指定さ

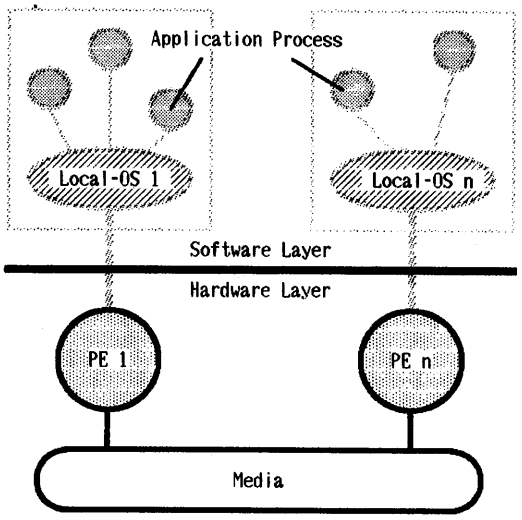

図２　分散処理システムの要素

れる．ソフトウェア要素はさらにプロセスの実行、他のプロセスとの通信等、PEの基本的な機能を提供するLocalOS要素とそれらの機能を組み合わせ実際のジョブを行うプロセスを定義するApplication Process要素に分かれる．

・LocalOS要素

PEの基本的機能を規定するソフトウェアの部分で、各PEに一つづつ対応している．LocalOSはApplication Process要素に対し、PEに割り当てられた複数プロセスの実行制御、IPCの管理などの機能を提供する．これらの機能はApplication Process に対し三つの基本命令(Primitive Operation: 後述)を実現する関数として提供される．プロセスの実行制御の方法（FIFO，ラウンドロビン、プロセッサ・シェアリングなど）によって幾つかのものが考えられ、標準Formが用意されている．

・Application Process

Application Process（以下単にプロセスと呼ぶ）は分散処理システムで行われるジョブを記述するもので、LocalOSの提供するprimitive operationの組み合わせでハードウェアとは独立に記述される．プロセスの記述にはＣ言語のシンタックスをそのまま用いることができる．ここでは分散処理システム上で行われる一つのジョブを比較的独立性の高い部分処理に分割し、それぞれ一つのプロセスとして記述する．各プロセスは一切の共有データを持たず、プロセス間通信を用いてデータのやりとりや同期をとる．これはいわゆるCSPモデル[2]に相当し、プロセス間の共有データを排除することによりプロセスの独立性が高まり、信頼性の高い分散型ソフトウェアを容易に組むことができる．また、プロセス間のデータのやりとりが単純になり、見通しのよい性能評価のモデルが構築できる．

４．３ primitive operation

SEDSでは分散処理システムの性能評価に注目している為、それ以外の要素はできるだけ排除し、利用者には見えない方がよい．プロセスを上記のCSPモデルで記述したとき、分散処理システムの性能に影響を与える動作はプロセスのPEでの実行とプロセス間通信である．これらの動作は共にPEやMediaなどの物理的資源を占有して行われる．その間他のプロセスはこれらの使用を制限される．これが分散処理システムの性能に大きな影響を与える．そこで、シミュレーションの実行に際し、シミュレーション時刻が進むのはこれらの動作を行っているときのみとし、その他の動作には時間がかからないとすることにより、分散型アルゴリズムの性能のみに注目したモデルの構築が行える．これらの動作は三つの基本命令(primitive operation)で表され、Ｃの関数としてLocalOSにより提供される．primitive operationは以下のものである．

exec(job_time)　CPUを利用してプロセスの実行を行う．複数プロセス実行時にはスケジューリング規約に従って実行される．

job_time プロセスの実行を行っている時間

send(pid, message, type)
receive(pid, message, type)　　CPUを利用してプロセス間通信を行う．プロセスはsend()関数を呼ぶことにより、LocalOSに通信要求を出す．LocalOSは通信の相手先プロセスが同じPE内にあれば、そのプロセスに、異なるPEにあれば、それらを結合しているメディアを通じてプロセス間通信を行う．

pid　　sendの場合は送信先、receiveの場合は送信元プロセスを表す．receiveの場合、送信元を指定しないANYが許され、任意の相手からのメッセージを受信できる．同報通信は行えない

message 送信する／受信したメッセージを指すポインタ．プロセス間通信によって授受される情報（Message）はその最大バイト数が固定されているほかはプロセスで内容を自由に使用できる．

type　　プロセス間通信の種別を表す．現在は以下の三種類を考えている．

BLOCKED 同期型．送受信側両方が待ち合わせを行い、同期がとられる．

BUFFERED 蓄積型．送信されたメッセージは受信側PEに取り込まれ、一旦内部のバッファに蓄積される．受信側プロセスのreceive()呼び出しはバッファにメッセージがあれば先頭のものを取り出す．メッセージがなければバッファにメッセージが来るまで待たされる．送信側プロセスは受信との待ち合わせを行わない．非同期通信の一種と考えられる．

NONBLOCKED 非同期型．送受信側プロセスとも待ち合わせを行わない．送信されたメッセージは一旦受信側に取り込まれる．受信側ではバッファにメッセージがあればそれが取り込まれるが、なければそのまま待ち合わせをせず、次に進む．

これらの型を用意することによって様々な分散型アルゴリズムを記述できる．

5　Form

SEDSへ利用者が入力すべきパラメータはハードウェア階層からソフトウェア階層まで多岐に渡るため、それらを簡単に扱えるよう、フォームを用いたパラメータ指定法を導入する．フォームとはパラメータを指定するための便宜上の書式であり、利用者は必要に応じてフォームの所望の項目を埋めていけばよい．SEDSでは必要なパラメータの指定を各要素ごとにそれぞれ独立ないくつかのフォームで指定できるようにした．SEDSで必要なフォームを表1に示す．

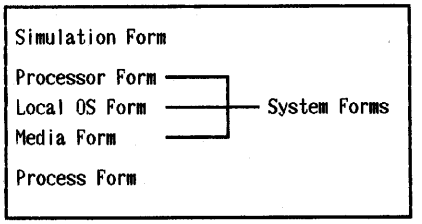

表1　SEDSにおけるフォーム

5．1　各フォームの特徴

ここでは各フォームの特徴について述べる．

Simulation Formはシミュレーションの実行管理を行うFormであり、シミュレーション実行による統計量の収集、出力のタイミングを指定する．シミュレーションの実行によって得られた評価測度はあるシミュレーション実行時間の平均値として出力される．統計処理の方法には二種類（set/time）ある．

その他のFormはそれぞれ、前出の分散処理システムの各要素に対応しており、それぞれに対応した項目を指定する．

Process formはMessage部、Statistics部とprogram部からなる．Message部ではApplication Processの間で実行時にやりとりされるメッセージの構造を定義できる．Statistics部はシステムが統計出力時に出力する評価測度以外に出力を希望する変数の宣言を行う．これにより利用者が自由に所要の出力統計量の作成が可能となる．分散処理システムで実行されるアルゴリズムの記述はProgram部に置かれる．

Local OS formおよびMedia Formの二

つに関してはいくつかの標準フォームが用意されており、利用者はパラメータのみをPE form内で変更できる．

５．２ parameters

SEDSではFormの記述に従い、シミュレーションを行うプログラムを生成し、コンパイルすることにより実行形式にする．したがって、一般に各Formの変更は実行形式であるSEDS/EMの再生成をうながす．ところが、一般には同じシミュレーションプログラムをいくつかのパラメータを変更して実行することが多く、パラメータを変更する毎にSEDS/EMを作り変えることは効率的ではない．このためSEDSではParameterと呼ばれる変数のクラスを用意している．ParameterはLocalOS Form，Process Form，Media Formで宣言可能である．これらの各FORMにおいて任意の変数をParameterとして宣言しておくことによりPE Formにおいてその変数を初期設定することが可能となる．実際には、PE Formに指定された各パラメータ変数の初期値はデータファイルに切り出される．シミュレーション実行時にSEDS/EMはこのデータファイルより各パラメータ変数の初期値を読み込みセットする．これにより各パラメータの値のみの変更はデータファイルのみの変更ですみ、シミュレーションプログラムを再生成する必要はない．

６　実際の問題への適用例

ここでは、SEDSを用いた分散処理システムの評価例としてジョブディスパッチ問題を取り上げ、その適用例を示す．

６．１　ジョブディスパッチャー問題

このモデルはマルチプロセッサ・システムである種の画像処理を行う場合に相当する[3]．[3]などの画像処理専用マルチプロセッサ・システムでは一画面を生成する際、均等ないくつかの小画面に分割し、それらを各プロセッサに分配して小画面毎に生成する．画面の分割及び分配は専用のプロセッサが行い、これをディスパッチャー（dispatcher）と呼ぶ．その他のプロセッサはディスパッチャーから配られた小画面の処理を行う．ここではこれらのプロセッサをエグゼキュータ（executer)と呼ぶ．したがって、このモデルにおけるマルチプロセッサ・システムは一つのディスパッチャーと複数のエグゼキュータからなり、ディスパッチャーは個々のエグゼキュータと通信路で結合されている（図3）．分割された各小画面の処理は独立して行えるためエグゼキュータ同士での通信はない．ディスパッチャーは一度に一つの小画面をエグゼキュータに渡し、その処理を依頼する．エグゼキュータはその受け取った小画面の処理が完了すると、その旨をディスパッチャーに知らせる．このときディスパッチャーにまだ未処理の小画面があれば、その中の一つを再びそのエグゼキュータに渡す．全ての小画面の処理が行われるまでこの手順で処理が行われる．このときエグゼキュータに小画面を渡すのに必要な時間をここではディスパッチャにおける処理時間として捉えている．

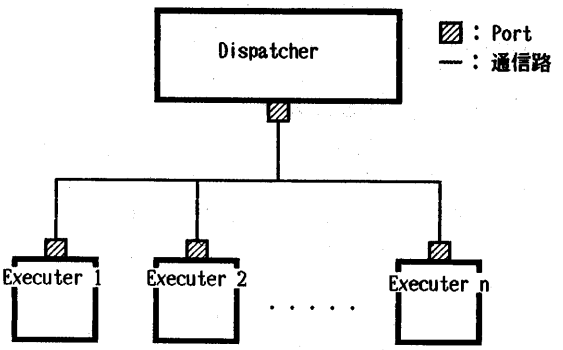

図3　ディスパッチャの構成

このような処理を行うとき、一つの画面をいくつの小画面に分解して処理するのが最も効率的かという問題が生じる．つまり、多くの小画面に分割すればするほど並列性は向上するが、小画面の分配にかかるオーバーヘッドにより画像全体の処理時間は増加する．一方、少ない数の小画面に分割するとオーバーヘッドは

小さくなるが、並列性は低くなりエグゼキュータの有効利用が計れない．したがって、一つの画像全体の処理時間を最小にするための最適な分割数が存在するはずである．これをジョブディスパッチャー問題と呼び、SEDSを用いて最適な分割数を求めてみる．

６．２　SEDSによるジョブディスパッチャー問題のモデル化

前節で説明したジョブディスパッチャー問題をSEDSの上にモデル化する．モデル化に際し次のような仮定を置く．

１）画面の分割に要する時間は無視する．
２）分割された各小画面の生成時間の平均は分割数に反比例する．
３）各小画面の分配に要する時間は平均１／μ_dの指数分布に従う．

ジョブディスパッチャー問題のパラメータを表2にまとめる．ディスパッチャーおよびエクゼキュータの処理は１つのapplication processとして表わし、それぞれ１つのPEに割り当てる．各PEには１つのapplication processが割り当てられ、各LocalOSはシングルタスクの処理を行う．
これらの定義をまとめ、Formの形で表したものを付録１に示す．

表2　ジョブディスパッチ問題のパラメータ

ジョブの分割数	N＝10-100
ディスパッチ時間	$1/\mu_d$=2.5
ジョブ実行時間	$1/\mu_e$=10000／N
executer数	M=4

６．３　シミュレーション結果

前節で説明したジョブディスパッチ問題について付録１のFormに基づきシミュレーションを実行した．

画面生成時間（すべての小画面の生成が終了する時間）と画面の分割数の関係を図4に示す．ここでエグゼキュータが小画面生成にかかる時間は指数分布に従うとし、図にはディスパッチャーの処理時間（小画面の分配時間）が2および5の場合を示している．図から画面生成時間を最小にする分割数が存在することがわかる．つまり、分割数を増加していくと、最初エグゼキュータに負荷が分散することにより画面生成時間は減少していくが、除々に小画面を分配する時間がオーバーヘッドとなり、やがて画面生成時間は増加し始める．したがって、小画面の分配時間が小さい（$1/\mu_d$=2）場合の最適な分割数は分配時間の大きい場合に比べ大きくなる．M=6の場合はこの生成時間の

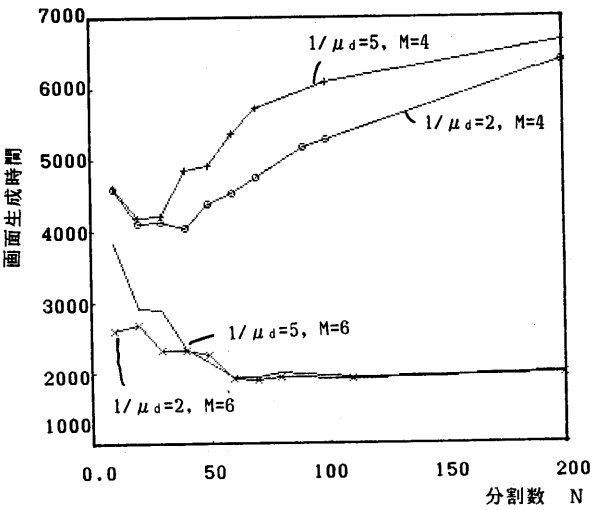

図４　画面生成時間（指数分布）

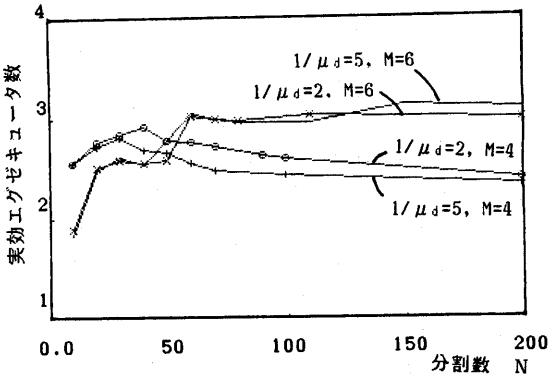

図５　実効エグゼキュータ数（指数分布）

増加がM=4程ではない．図5には実効エグゼキュータ数（平均して働いているエグゼキュータの数）を示しており、M=4の場合、実効エグゼキュータ数が減少していくのに対してM=6の場合あまり減少しないことがわかる．

以上の例では小画面の生成時間を指数分布に従うものとした．したがって、分割数が増加するに従い、小画面の生成時間の分散は平均とともに減少していく．ところが、画像処理アルゴリズムとして[3]のような視線探索法を用いた場合、処理する画面の領域を小さくしても、場合によっては全画面を探索することにり小画面生成時間の分散は分割数を増やしても余り小さくならない．そこで、次に小画面生成時間を正規分布に従うものとし、平均は分割数に従って小さくなるが分散は一定であるとして、シミュレーションを行った．図6はこのときの画面生成時間を示している．指数分布の場合と比較すると、分散が一定であることにより、長い処理時間を有する小画面が全体の画面生成時間を引き上げ、画面生成時間の増加の傾向が著しい．σ=1000の場合にこのことが顕著に表れている．図7には実効エグゼキュータ数を示しているが、実効エグゼキュータ数自体はあまり減少していない．このことはエグゼキュータ数を増やしたり、ディスパッチャーの処理時間を小さくしたとしても画面生成時間を引き下げる効果は薄いことがわかる．したがって、画面生成時間をさらに小さくするには小画面の生成時間の分散を小さくすることが効果的であり、実際[3]でも画面の分散を小さくする工夫を加えている．

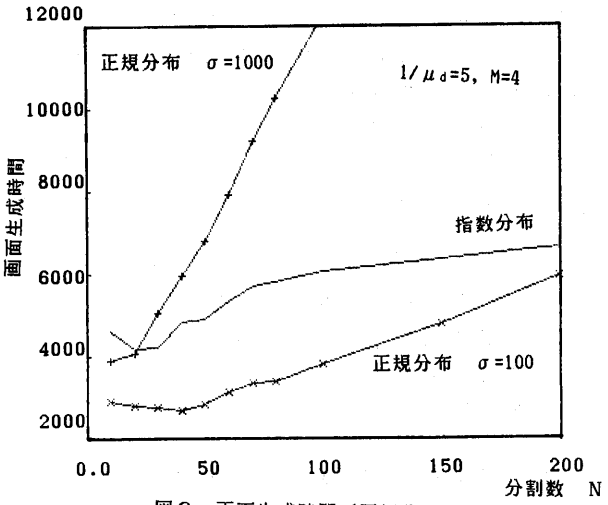

図6　画面生成時間（正規分布）

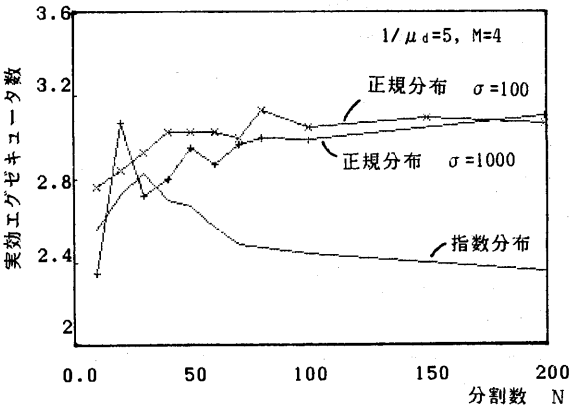

図7　実効エグゼキュータ数（指数分布）

7．おわりに

分散処理システム評価シミュレータSEDSを実際の画像処理専用マルチプロセッサシステムにおけるジョブディスパッチ問題に適用した例を示した．シミュレートする分散処理システムの記述にFormを用いることにより、短期間にモデルを構築できた．さらに、SEDSを適用できる分散処理システムにまつわる問題として以下のものを考えている．

・LANにおける上位レイヤーを含めた性能評価

・画像処理専用マルチプロセッサシステムの単位コンピュータの設計

謝辞　本研究を行う際に貴重な助言を頂いた大阪大学工学部電子工学科寺田研究室ＣＧグループの皆様に深く感謝致します．

参考文献

[1]:山口、下条、宮原、高島、”分散処理システム性能評価シミュレータ”、信学技報　情報ネットワーク研究会、IN85-110、1986年３月.

[2]:Hoare, C. A. R., "Communicating Sequential Processes," C. ACM. vol. 21, no. 8, pp. 666-677. AUG. 1978.

[3]:西村，出口，辰己，河田，白川，大村，"コンピュータグラフィックシステムLINKS-1における並列処理の性能評価"，信学会 論文誌(D)，Vol. J68-D, No. 4, pp. 733-740, (1985.4).

付録１　'Form'によるジョブディスパッチ問題の記述

a)　Simulation Form

```
/* ジョブディスパッチ問題のシミュレーション */
Simulation Form  JOB DISPATCHER
    Type of Simulation Run: set /* set指定 */
        Total Set: 40        /* 40setで統計量を
        Interval Sets: 40    出力して終了する    */
```

b)　PE Form

```
/* ディスパッチャーの記述 */
PE Form DISPATCHER

    /* LocalOSの指定 */
    Local OS: Round_Robin        /* 標準Form */
        parameters
            slice_time=9999     /* シングルタスクの
                    ため、スライス時間は無限大 */

    /* 割り当てるプロセスの指定 */
    Processes: dispatcher
        parameters
            TJOB = 10000     /* 全ジョブ処理時間 */
            DJOB = 5     /* ディスパッチャーの
                        処理時間 */
            NJOBS = 100 /* 分割数 */

    /* Mediaとの接続関係 */
    Ports
        a: BUS1              /* メディア名 */

/* エグゼキュータの記述１～４ */
PE Form PE1
    Local OS: Round_Robin   /* シングルタスク処理 */
        parameters
            slice time=9999

    Processes: executer

    Ports
        a: BUS1 /* Mediaに対するparameterの指定 */
            parameters
                elasped_time=0
```

c)　Process Form

```
/* ディスパッチャー・プロセスの記述 */
Process Form DISPATCHER
    /* 開放するパラメータの宣言 */
    Parameters
       double TJOB, DJOB, NJOBS;

    /* プロセス間でやりとりされるメッセージの型 */
    Message
        int  no;     /* パケットを送ったエグゼキュータの番号 */
        double  average;  /* 割り当てるジョブの処理時間の平均 */

    /* 実際の処理の記述 */
    Program
dispatcher()
{   PROCESS ex[5], find();
    MESSAGE *mes;
    double exp rand();
    int loop, i;

/* エグゼキュータpidの取り出し */
ex[1] = find( "executer1" );
ex[2] = find( "executer2" );
ex[3] = find( "executer3" );
ex[4] = find( "executer4" );

mes->average = TJOB / NJOBS;    /* 小画面の生成時間の平均 */

for(i = 1; i <= 4; i++) /* １つめのジョブの割り
    send( ex[i], mes, BLOCKED);      付け */

for(loop = 4; loop < NJOBS - 4 ; loop++){
    recieve( ANY, mes, BLOCKED);    /* 小画面生成終了待ち */
    exec( exp rand( DJOB ));    /* ディスパッチの処理 */

    /* 次のジョブの割り付け */
    send( ex[mes->no], mes, BLOCKED);
}

/* 各エグゼキュータから最後の小画面生成の終了を受ける. */
for(loop = 0; loop < 4 ;loop++)
    recieve( ANY, mes, BLOCKED);

/* 小画面の終了を各エグゼキュータに伝える. */
mess->no = 999;     /* 999が最後を表わす. */
for(i = 1; i <= 4; i++)
    send( ex[i], mes, BLOCKED);
}


/* エグゼキュータの処理の記述１～４ */
Process Form executer1
    Program
executer(){
    PROCESS dispatcher;
    MESSAGE *mes;
    double exp rand();

    /* ディスパッチャーpidの引き出し */
    dispatcher = find( "DISPATCHER" );

    while( TRUE ){
        recieve( dispatcher, mes, BLOCKED);
                                /* 小画面の受取 */

    if( mes->no == 999 ) break; /* 処理の終了 */

    exec( exp rand( mes->average ) );   /* 小画面の生成処理 */

    mes->no = 1   /* 1 is process number. */ (注)
    send( dispatcher, mes, BLOCKED);
                                /* 生成終了   */
    }
}
```

注) エクゼキュータ２～４までのProcessor Form はエクゼキュータ１と同様であるため省略した.