The University of Osaka
Institutional Knowledge Archive

Title	分散処理システムの性能評価に関する研究
Author(s)	下條, 真司
Citation	大阪大学, 1986, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/1526
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

分散処理システムの性能評価に関する研究

Studies on
Performance Evaluation of
Distributed Systems

昭和６１年２月

下条真司

内容梗概

本論文は分散処理システムにおいてプロセス間通信を考慮に入れてシステムの性能評価を行う手法について述べている．本論文で扱う分散処理システムとしては、単一のプロセッサと独自のローカルメモリーからなる各処理要素（PE）が共通の通信路を介して結合されているものとする．分散処理システム上で実行される一まとまりの仕事をジョブと呼び、ジョブは分散処理可能な幾つかの部分処理、すなわちプロセスから構成されるとする．各プロセスは別々のPEで並列に実行可能であるが、これら各プロセス間での処理の同期およびデータの相互転送が必要であり、その結果、各プロセス間でプロセス間通信が行われる．

分散処理システム上でプロセスを各PEに割り当て実行する場合、同一のPEに割り当てられたプロセス間同志ではプロセッサの使用に関し競合が生じ、また一方、異なるPEに割り当てられたプロセス間同志ではプロセス間通信による通信路の競合が生じ、これらがシステムの性能を大きく左右する要因となる．例えばプロセスを割り当てるPE数を減少させると同一PE内のプロセッサの競合が増大し、また一方プロセスを割り当てるPE数を増加させると異なるPEに割り当てられたプロセス同志のプロセス間通信による通信路の競合が増大する．したがって、分散処理システムにおいてあるジョブを実行する場合、その上で分散して処理されるプロセスのPEへの割り当て方法によってジョブの実行が完了する時間は異なって来る．そこでここでは分散処理システム上であるジョブの実行が開始されてからその実行が終了するまでの時間の平均、すなわち平均ジョブ実行時間を評価測度としてとり、これを最小にするようなプロセスの割り当てを求める問題を考えた．

まず、任意にプロセスを割り当てた場合に対し、そのときの平均ジョブ実行時間を待ち行列網理論を用いて解析的に求める手法を提唱した．次に平均ジョブ実行時間を目的関数とするプロセス割り当て問題を整数計画問題として定式化した．このうち、均質な（実行時間と他のプロセスとの通信確率が等しい）プロセスに対する

プロセス割り当て問題に関し、平均ジョブ実行時間を最小にするPE数を解析的に求めた．これにより、均質プロセスの割り当て問題においてすべての可能な割り当てを探索することなく容易に求めることができた．また、一般の（実行時間および通信確率が各プロセス毎に異なる）プロセス割り当て問題に対しても、各PEにおけるプロセスの実行時間の総和を等しくする（ロードバランス）とともに各PE間の通信量がほぼ等しくなるような割り当てにより平均ジョブ実行時間を最小化できるという発見的手法を導入し、これに基づく二つの近似解法を提唱した．中でも分枝限定法に上記の発見的手法を導入することにより、解の探索時間を大幅に改善することができた．またいくつかの数値例に本手法を適用し、厳密解と比較することにより、この手法で得られる近似解の有効性を示した．その結果、プロセスを割り当てるPE数が増加すると平均ジョブ実行時間は大幅に減少するが、最適PE数を越えると、それ以降プロセス間通信によるオーバーヘッドにより平均ジョブ実行時間が徐々に増加するということが明らかになった．

また、解析では扱うことが困難な複雑なシステムの性能評価を行い、解析で導入した近似の妥当性を検討するために、分散処理システムにおいてソフトウェア・ハードウェアを統合的に評価できるようなシミュレータＳＥＤＳを開発した．

このように、本論文ではプロセス間通信による通信競合を含めた分散処理システムの性能評価手法を数値解析、シミュレーションの両面について提唱するとともに、この評価手法に基づき分散処理システムの設計に際して問題となるプロセス割り当て法に対する近似解法を示し、分散処理システムにおいてその上で実行されるジョブによるプロセス間通信を考慮に入れて性能評価する手法を確立した．

関連発表論文

A. 論文誌関係

1. 下条，宮原，高島，"通信競合を含めたマルチプロセッサにおけるプロセス割り当て問題",信学会 論文誌(D), Vol. J68-D, No. 5, pp. 1049-1056, (1985,5).

2. Shinji Shimojo, Hideo Miyahara, and Kensuke Takashima, "Process Assignment on Distributed System with Communication Contentions," Proc. of Inter. Seminar of Comput. Network and Perf. Eval., pp. 11-5, SEP 1985.

3. 下条，宮原，高島，"分散処理システムにおけるプロセス割り当て問題に対する近似解法",信学会 論文誌(D),（投稿中）

B. 専門研究会関係

1. 下条，西田，宮原，高島，"通信競合を含めたプロセス割り当て問題とその近似解法",信学技報 電子計算機研究会, Vol. EC84, No. 47, pp. 25-35, (1984,12).

2. 下条，宮原，高島，"分散処理システムにおけるプロセスの割り当て法の比較 検討",信学技報 電子計算機研究会, Vol. EC85, No. 16, pp. 21-28, (1985,8).

【目次】

『分散処理システムの性能評価に関する研究』

１．　序論　1

２．　通信競合を含めたプロセス割り当て問題　6

　２．１　分散処理システムとプロセス　6

　２．２　プロセス割り当て問題の定式化　8

３．　処理とＩＰＣの同時実行を行わないプロセス割り当て問題　9

　３．１　プロセスと分散処理システムのモデル　9

　３．２　問題の定式化　10

　３．３　平均外部プロセス通信時間の導出　11

　３．４　均質プロセスに対するプロセス割り当て問題　13

　　３．４．１　モデル　13

　　３．４．２　問題の定式化および最適ＰＥ数　14

　３．５　実行時間が異なるプロセス割り当て問題への拡張　17

　３．６　考察　21

　　３．６．１　均質プロセスに対する割り当て問題　21

　　３．６．２　要求処理量が異なるプロセス割り当て問題　27

４．　処理とＩＰＣの同時実行を考えたプロセス割り当て問題　30

　４．１　分散処理システムの待ち行列網によるモデル化　31

４．２　プロセスの同期　33

４．３　プロセス割り当て問題の近似解法　34

４．３．１　割り当て問題の定式化　35

４．３．２　分枝限定法による解の探索　36

４．３．３　発見的手法　37

４．３．４　目的関数値の下限の導出　38

４．３．５　ｄの決定方法　39

４．４　考察　41

５．　分散処理システム評価シミュレータ（ＳＥＤＳ）　45

５．１　ＳＥＤＳの必要性　45

５．２　ＳＥＤＳの特徴　46

５．３　シミュレータの動作　47

５．４　分散処理システムの要素　48

５．５　実行型式モジュール（SEDS/EM）の構成　51

５．５．１　SEDS/EMの内部構成　51

５．５．２　ＳＥＤＳにおけるシミュレーション時刻管理　53

５．６　ＳＥＤＳの入力フォーム　54

５．６．１　フォームの種類　54

５．６．２　各フォームの型式　56

５．７　ＳＥＤＳのパフォーマンス・メジャー　58

５．７．１　基礎となる統計量　58

５．７．２　ＳＥＤＳの評価測度　60

５．８　ＳＥＤＳの実験例（ジョブディスパッチ問題）　62

５．８．１　dispatcher/executerモデル　62

５．８．２　ＳＥＤＳによるＤ／Ｅ問題のモデル化　63

５．８．３　シミュレーション結果　65

６．　結論　68

謝辞　72

参考文献　74

付録１　平均外部通信時間の導出　78

付録２　フォームによるＤ／Ｅ問題の記述　81

1．　序論

半導体技術の進歩によりVLSIの実用が可能となり、CPUやその他の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．

初期には多数のCPUを通信路を介して密に結合したマルチプロセッサ・システムが多数開発された[1],[2],[3]．マルチプロセッサ・システムにおいては、与えられた問題を命令レベルまで分割して個々のCPUの処理能力を最大限利用しようとするため、一般的な問題に適用しうる汎用マシンの開発はやや困難があるように思われる．なぜなら、問題をあまり細かく分割しすぎると、分割した処理単位同志による同期やデータの通信などのオーバーヘッドが増大し、期待する程に効率は上がらないという事実が存在するためである．個々のCPUの処理能力が増加するに従い、オーバーヘッドが全体の処理時間において占める割合が大きくなる．さらに、マルチプロセッサ・システムでは問題に含まれている並列性を記述する適当な方法がなかったことも汎用のマルチプロセッサ・システムを構築する上で大きな障害となった．したがって、マルチプロセッサ・システムは、比較的並列度の高い特定の問題に対し、最大限に効率を上げるようハードウェア、ソフトウェアの両面から最適化した専用化の道を歩むことになり、ベクタ・マシン、アレープロセッサなどとして実現された．

一方、半導体などデバイス技術の進歩は、通信装置の低価格・高機能化をももたらし、通信コストを低下させることにより、大規模なコンピュータ・ネットワークからＬＡＮ（Local Area Network）までの通信網の発達を促した．これにより、複数の計算機をＬＡＮなどを通じて疎結合し、それらの間で負荷分散や機能分散を行

おうといういわゆる分散処理システムが構築されつつある[4].

さらに、主としてオペレーティング・システム（ＯＳ）などにおいて必要とされる並列事象の記述機能を備えた言語も出現した．Concurrent Euclid[5], Occam[6]などがそれである．これらの言語においては使用者が意識して並列性を見出し記述しなければならなかったが、使用者が意識することなく、問題に内在する並列性の記述が行える手法も登場した．CSP[7]、オブジェクト指向言語（smalltalk,Concurrent prologなど）、アクター理論[8]などがそれに相当する．これらを用いて記述することの特長は、問題を極度に分割することなく、ある程度直列に処理する部分を残したまま、適当な大きさに分割しうる点にある．このようにソフトウェアの記述法が整備されるにつれて、より汎用で柔軟性に富む分散処理システム構築の基盤が整いつつある．

分散処理により解決される問題の範囲もＦＦＴ[9]や連立方程式の求解[10]、グラフ理論における様々な問題[1],[11]、ソーティング[12][13]などからＡＩ（Artificial Intelligence）[14],[15]まで広範囲に及んでいる．また、具体的に分散処理システム上に並列処理言語で記述した問題を適用した例として、ネットワークプロトコルの各階層の実現[16]やグラフに対する諸問題[17]等がある．

ここで、分散処理システム上で処理される一まとまりの仕事をジョブと呼ぶことにする．ジョブは先のOccam, Concurrent Euclid, アクターモデル等で記述され、分散処理可能な幾つかの部分処理、すなわちプロセスから構成されている．ところが、ジョブを複数のプロセスに分割して分散処理する際には、各プロセス間での同期、データのやりとりなど、プロセス間通信（IPC：InterProcess Communication）の必要性を生み、そのために必ずしも分散化による効率向上が望めないという問題がある．この分散化によるＩＰＣのオーバーヘッドは、ジョブとそれが実現される

分散処理システムの構造に大きく依存し、また、分散処理システム上でのプロセスの分散の仕方によっても異なって来る．

つまり、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問題が存在し、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．

ところが、このような統合的な性能評価手法はいまだ開発されていない．専用VLSIによる処理デバイスの性能評価は特定のアルゴリズムとハードウェアごとに組み合わせやグラフ理論等の手法を用いて行われて来た[12][13][18]が、ここでの手法をより汎用な分散処理システムの性能評価に拡張することは困難である．また、マルチプロセッサ・システムが種々提案されるにしたがい、それらのハードウェア面からの評価もなされてきたが、それらは、主として多数の計算機による共通メモリー競合を持つシステムの評価や計算機同志を相互に接続する通信ネットワークの性能評価がマルコフ過程や待ち行列網理論を用いて行われているに止まっている[19][20][21]．つまり、従来行われて来た分散処理システムの評価手法においては、プロセスの処理やそれらに依存したプロセス間通信がシステムの性能へ与える影響を考えていない[22]．さらに分散型アルゴリズムの評価も、分散処理システムの形態に依存したプロセス間通信の影響を考えておらず、ソフトウェア、ハードウェア両面を含めて分散処理システムを論じたものはほとんどない．

そこで、本論文ではプロセス間通信による通信競合をも含めた分散処理システムの性能評価の手法を示す．すなわち、分散処理システムにおいて実行される一つの

ジョブをプロセスとしてシステム上に分散し、プロセス間の依存度をプロセス間通信としてとらえる．これらのプロセスを実行する分散処理システムを待ち行列網としてモデル化し、平均ジョブ実行時間を求めることにより、分散処理システムをソフトウェア、ハードウェア両面から統合的に評価する手法を提唱した．ここでジョブ実行時間とは、そのジョブに属するすべてのプロセスの実行が終了する時間である．各プロセスが割り当てられたプロセッサにおいて処理を完了するのに要する時間をプロセス終了時間とすれば、ジョブに属するプロセスのプロセス終了時間のうちで最大のものがジョブ実行時間となる（図1.1）．分散処理システム上で実行されるジョブの構成（ジョブを構成する各プロセスの実行時間とプロセス間通信確率）と分散処理システム形体がシステムの性能に与える影響は、プロセス間通信による通信競合を考慮することにより初めて評価することが可能になる．

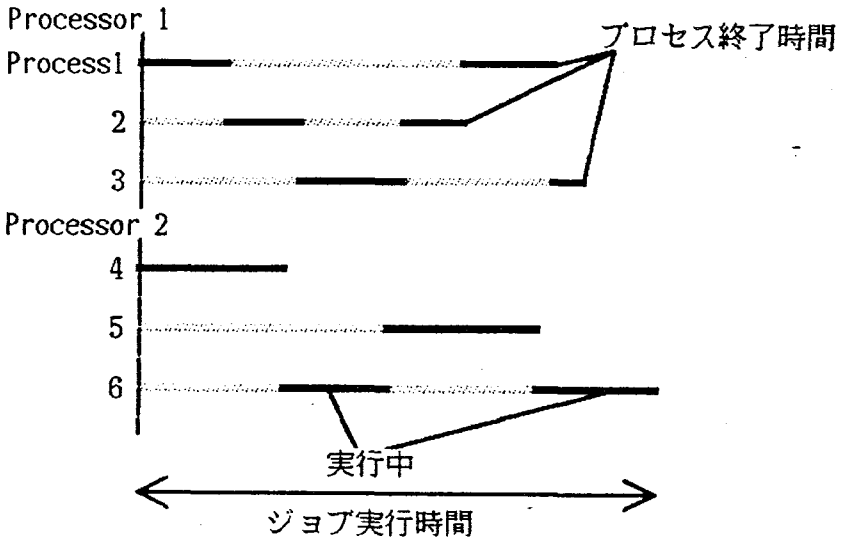


図1.1　ジョブ実行時間

プロセス間通信を考慮した分散処理システムの性能評価手法を利用して、本論文では通信競合を考慮したプロセス割り当て問題を考える．つまり、ハードウェア・ソフトウェアの構成が固定されると、分散処理システムの性能はジョブを構成する各プロセスを分散処理システム上にどのように割り当てるかによって大きく異なってくる．ここでは先のジョブ実行時間を最小にする割り当てを求める近似解法を示

している．プロセス割り当て問題[23]のうち、プロセッサ間の通信を考えず、プロセスの実行順序のみを扱ったものとしては、ジョブのスケジューリング問題として古くから考えられている[24][25]．また、文献[26][27][28]においてはプロセス間の依存関係を一定のコストに置き換えて議論している．しかし、プロセス間通信による競合も含めた議論は殆どなされていない．

本論文で行った性能評価では、解析においていくつかの近似を導入し簡単化を行っている．そこで、近似の正当性を検証するため、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評価するための専用シミュレータを開発した．このシミュレータにより、先の解析手法の有効性を評価できるとともに、解析では扱えない複雑な分散処理システムの評価が可能になる．

本論文は次のような構成になっている．第２章では分散処理システムにおけるプロセス割り当て問題について明らかにするとともに、整数計画問題として定式化する．第３章と第４章では待ち行列網を用いた分散処理システムの２種類のモデル化について述べ、分散処理システム上で実行されるプロセスのモデルについても言及する．第３章ではプロセスの処理とＩＰＣの同時実行を行わないもとでの分散処理システムのモデルについて解析法を示すとともに、近似を導入することにより実行時間とプロセス間通信確率が等しい均質なプロセスに対する最適プロセッサ数を求めた．さらに、均質なプロセスに関する条件を緩める手法についても述べる．第４章ではプロセスの処理とＩＰＣの同時実行を行うモデルについて解析法を示すとともに、ここでは分枝限定法を用いて一般のプロセス割り当て問題に対する解法を示した．第５章では本論文で扱っているプロセス割り当て問題を始めとする分散処理システムにおける様々な問題をソフトウェア、ハードウェアの両面から総合的に評価するために筆者が作成したシミュレータについて、その概要を述べる．

2．　通信競合を含めたプロセス割り当て問題

2．1　分散処理システムとプロセス

ここでは、分散処理システムを図2.1に示されるような構造を持つものと仮定する．すなわち、処理要素(PE: Processing Element)は、単一のプロセッサと独自のローカル・メモリー(LM: Local Memory)からなり、各PEは相互通信網(CN: Communication Network)を通じて互いに結合されている．CNとしては共通バスやクロスバー・スィッチのようなものから、LANのようなものに至るまで種々考えられる．

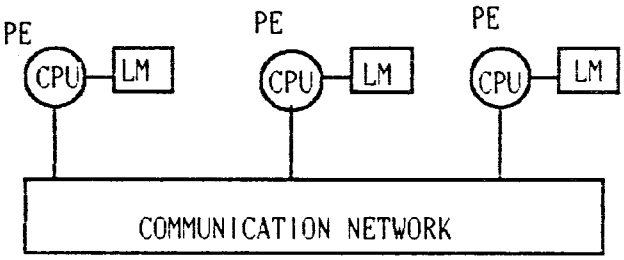


図2.1　分散処理システム

このような分散処理システム上で処理される一まとまりの仕事をジョブと呼ぶ．ジョブは複数のプロセスからなり、これらのプロセスは各々別々のPEで並列に実行することが可能である．しかし、同一のジョブに属するプロセスは互いに関連性があり、プロセス間通信(IPC: InterProcess Communication)によって互いのデータの授受、同期などを行う

各PEはプロセスの実行やIPCを管理するためのOS（オペレーティング・システム）を備えている．一つのPEには複数プロセスの割り当てが可能であり、これらのプロセスはラウンド・ロビン方式によって処理される．同一のPEに割り当てられたプロセス同志ではLMを通じてIPCを行う（内部IPC）が、異なるPEに割り当てられたプロセスはCNを通じてIPCを行う（外部IPC）．分散処理システムにおいては一つのジョブに属するプロセスをどのように割り当てるかによって効率は大きく異なって来る．

例えば、一つのジョブに属する複数のプロセスを分散処理システム上に分散配置

する際、各プロセスはそれらの実行に際し、次のような二つの制約を受けることになる．

1) 同一PEに割り当てられたプロセス間でのPEの競合

2) 異なるPEに割り当てられたプロセス間通信における通信路の競合

上記の制約から、PEの使用率、プロセス、しいてはジョブ全体の実行時間は各プロセスをどのようにPEに割り当てるかによって大きく左右される．多くのPEにプロセスを割り当てると、異なるPEに割り当てられたプロセス同士のプロセス間通信量は増大し、したがって、通信路でのPE間競合が大きくなる．一方、割り当てを行うPE数を減少させると、同一のPEに割り当てられたプロセス間では通信のための通信路の競合を避けることができるものの、同一PEに割り当てられたプロセス間でのPE競合が増大することになる[23]．したがって、分散処理システムの構築時における大きな問題は、互いに依存性のある処理をどのようにシステム上に分散化するかということであり、ここでは、これをプロセス割り当て問題と呼ぶことにする．

ここでは、プロセス間通信における通信路の競合をも考慮して分散処理システム上でのプロセス割り当て問題を取り上げる．システムの性能を評価するための測度の一つとして、ジョブの処理時間を考える．すなわち、各PEに割り当てられたすべてのプロセスの実行が終了する時間を最小にする割り当てを求めることにする．この全てのプロセスが実行を終了する時間の平均値を平均ジョブ実行時間と呼ぶ．そのためにまず、このプロセス割り当て問題に待ち行列網理論を適用するため、ここで対象としている分散処理システムに対し二種類のモデル化を行う．第一のモデルは比較的簡単な分散処理システムをモデル化したものであり、第二のモデルではより大きな分散処理システムを考えており、第一のモデルでは考慮していなかったIPCとPE内部での処理の同時実行の効果を導入している．つまり、前者ではあるPEがIPCを実行している間は他のプロセスの処理を行わないとしたが、後者ではこれを許している．分散処理システムをモデル化した後、待ち行列網理論を用いて平均

ジョブ実行時間を求める方法について述べる．

２．２　プロセス割り当て問題の定式化

分散処理システムにおけるプロセス割り当て問題を0/1整数計画問題として定式化する．プロセスの割り当てを割り当て行列Ｘを用いて次のように表現する．

$$x_{ij} = \begin{cases} 1 & \text{プロセスｉがＰＥｊに割り当てられている．} \\ 0 & \text{プロセスｉがＰＥｊに割り当てられていない．} \end{cases} \qquad (2.1)$$

ある割り当てＸに対するPE数をn_pとする．このとき、プロセス割り当て問題は

条件

$$\sum_{j=1}^{n_p} x_{ij} = 1 \qquad i=1,\ldots,M \qquad (2.2)$$

x_{ij}＝0または1

の下で

Ｔ（Ｘ）　　を最小化せよ

となる．Ｔ（Ｘ）は割り当てＸのときの平均ジョブ実行時間を表す．以下では待ち行列網理論を用いてＴ（Ｘ）を求める方法について述べる．上式の整数計画問題はＴ（Ｘ）が非線型となるため、一般には直接解くことは難しい．したがって、なんらかの近似手法を導入して解く必要がある．第３章では待ち行列網の漸近的特性を利用した近似を導入し、第４章では分枝限定法を用いて近似解を求める手法について言及する．

3．　処理とIPCの同時実行を行わないプロセス割り当て問題

3．1　プロセスと分散処理システムのモデル

一つのジョブの全プロセス数をMとする．各PEの能力は等しく、一単位の処理を終えるのに必要な時間を以下単位時間とする．各プロセスはa_i(i=1...M)単位の処理（要求処理量）を必要とする（図3.1）．各PEには複数のプロセスが割り当てられるが、一時には高々一つのプロセスしか実行されず、PEに割り当てられた各プロセスはプロセッサ・シェアリング（PS）方式で処理されるものとする．

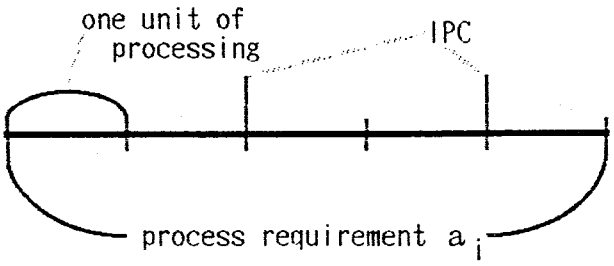


図3.1　プロセスｉのパラメータ間の関係

ここでは、各プロセスは一単位の処理を終えた直後にIPCを行い、プロセスｉが一単位の処理を終えた後にプロセスｊとプロセス間通信[+]を必要とする確率をp_{ij}とする．ただしプロセスはIPCを行わずそのまま次の処理を続けることもあるので、

$$\sum_{i=1}^{M} p_{ij} < 1 \qquad j=1...M \qquad (3.1)$$

となる．このとき、あるプロセスが同時に二つ以上のプロセスと通信することはないものとし、また、各プロセスの順序関係は考えず、プロセス間相互の通信のみを考える．PEは、あるプロセスがIPCを行なっている間は、別のプロセスの処理は行

一般にプロセス間通信といった場合、プロセス iが jに要求を出し、それに対する応答を jから受け取るといった具合に相方向の通信となるが、ここではプロセス間における同期を考慮していないため、 i→ jと j→ iとを別々のトラヒックと考えている．

なわない．つまり、プロセス間IPCと処理の同時実行はないものとする．

３．２　問題の定式化

分散処理システムを待ち行列網として解析し、平均ジョブ実行時間Ｔ（Ｘ）を求める．

PE j に割り当てられたプロセス i が、一単位の処理を終えたときに他のプロセスと通信する確率を σ_i、そのうちCNを介して通信する確率を γ_i とすると、

$$\sigma_i = \sum_{\substack{j=1 \\ j \neq i}}^{M} p_{ij} \qquad i=1,\ldots,M \qquad (3.2)$$

$$\gamma_i = \sum_{j=1}^{n_p} \sum_{k=1}^{M} x_{ij} (1 - x_{kj}) p_{ik} \qquad i=1,\ldots,M \qquad (3.3)$$

となる．これから、PE j が、一単位の処理を終えたときにCNを介して通信する確率 λ_j およびローカル・メモリーを介して通信する確率 λ'_j は、

$$\lambda_j = \frac{\sum_{i=1}^{M} a_i x_{ij} \gamma_i}{\sum_{i=1}^{M} a_i x_{ij}} \qquad j = 1 \ldots n_p \qquad (3.4)$$

$$\lambda'_j = \frac{\sum_{i=1}^{M} a_i x_{ij} (\sigma_i - \gamma_i)}{\sum_{i=1}^{M} a_i x_{ij}} \qquad j = 1 \ldots n_p \qquad (3.5)$$

である．

あるプロセスがCNを介して他のプロセスと通信する場合（外部IPC）、これに要する時間は、そのプロセスがCNを占有する時間+の他に、CNを占有するまでの待ち時間が加わることになり、ここでは、これらの和の平均値をR_c（平均外部プロセス間通信時間）として表わすことにする．一方、あるプロセスが、ローカル・メモリーを介して他のプロセスと通信する場合（内部IPC）、これに要する時間は、外部プロセス間通信におけるようなCNでの競合がないと考えて、平均値$1/\mu_{lm}$の指数分布に従うとする．したがって、PE j に割り当てられた全てのプロセスの実行を終了するのに必要な平均時間R_jは、

$$R_j = \sum_{i=1}^{M} a_i x_{ij} (1 + \lambda_j R_c + \lambda'_j 1/\mu_{lm}) \qquad j = 1 \ldots n_p \qquad (3.6)$$

となる．評価基準として、平均ジョブ実行時間

$$T(X) = \max_{1 \leq j \leq n_p} (R_j) \qquad (3.7)$$

を用い、これを最小にする割り当てXを求める．

3．3　平均外部プロセス通信時間の導出

PE内部でのプロセスの処理は、図3.2-aのように進むと考えられる．PE j に割り当てられるプロセスは、一時に一つしか処理されず、一単位の処理を終えると、PE j は確率λ_jでCNに、確率λ'_jでローカル・メモリーに通信要求を出す．通信要求がなければ、そのまま次の処理を開始する．PE jの単位時間当たりのCNへのアクセ

+ 実際のモデルでは、この時間はCN上でのデータ伝送時間であり、データ長およびCNの伝送速度で決まる値である．

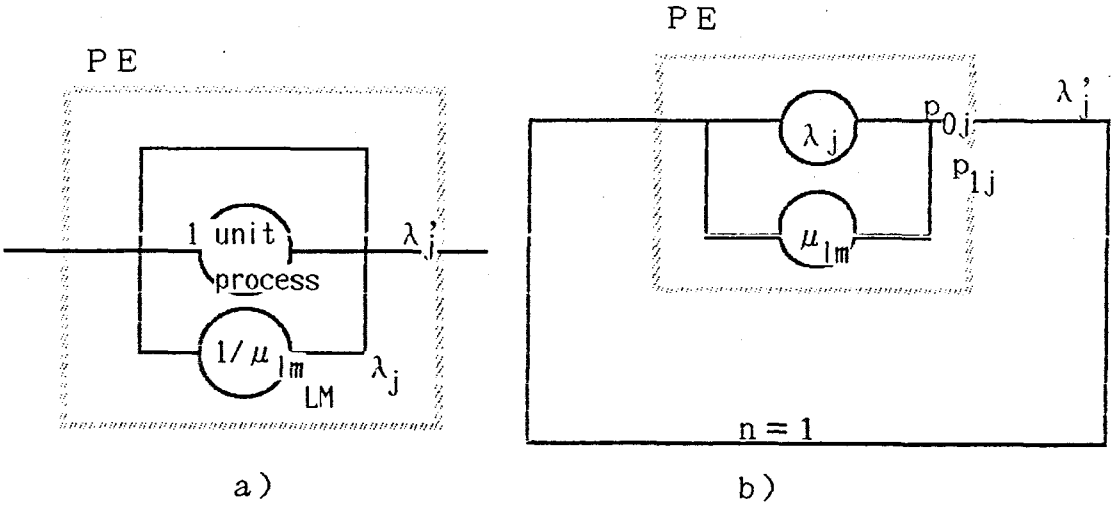


図3.2　ＰＥにおける内部処理のモデル

ス率を求めるために、図3.2-bのような閉じた待ち行列網のスループット$\lambda^{PE}{}_j$を求めると、

$$\lambda^{PE}{}_j = \lambda_j \mu_{lm} / (\lambda'_j + \mu_{lm}) \qquad j = 1 \ldots n_p \qquad (3.8)$$

となる．このとき、PE jは、処理率が$\lambda^{PE}{}_j$で与えられる単一指数サーバーとみなせる[30]．$\lambda^{PE}{}_j$をまとめてアクセス・ベクトル

$$\Lambda = (\lambda^{PE}{}_1, \lambda^{PE}{}_2, \ldots \lambda^{PE}{}_{np}) \qquad (3.9)$$

とすると、R_cはΛの関数であり、それを$R_c(\Lambda)$と書く．したがって、CNへの通信要求の到着は、n_p個の独立したソースからの重畳であり、その過程はポアソン分布に従うから、CNを一つのコンポジット・キューとみなせば、システム全体は待ち行列網理論に於けるセントラル・サーバー・モデルと考えられる．つまり、客数n_pで、到着率がΛで与えられるM／G／1／／n_pとなり、解析的に$R_c(\Lambda)$を求めることが可能である[31]．

ここで一例として、CNが単一共有バスであり、また、外部プロセス間通信における各プロセスのCN占有時間は、平均$1/\mu_{cn}$の同一の指数分布に従うものとする．

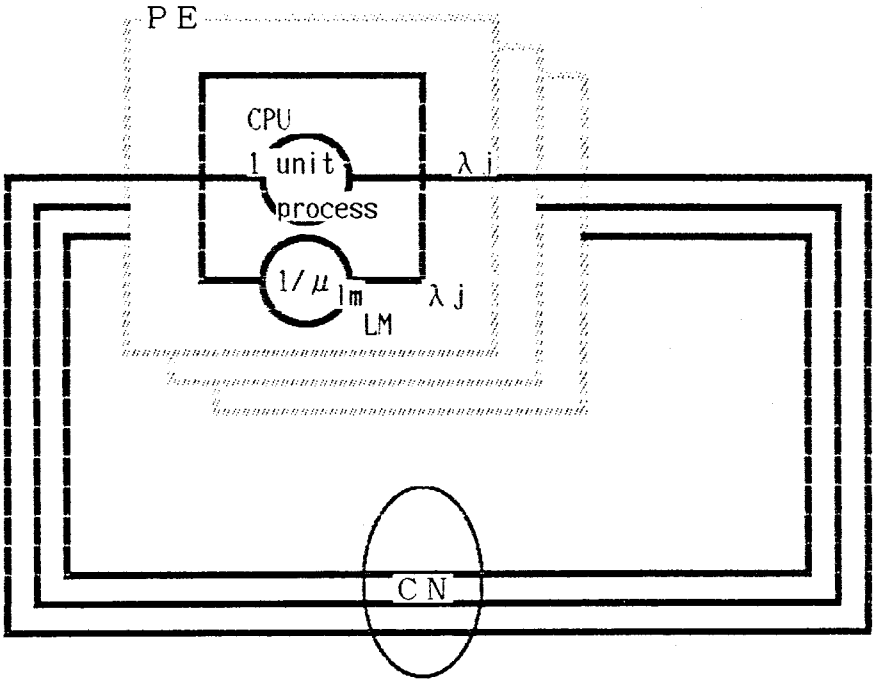


図3.3　分散処理システムの待ち行列網モデル

このとき、系全体は複数クラスを持つM／M／1／／n_pとなり（図3.3）、積形式解が得られ平均外部プロセス間通信時間$R_c(\Lambda)$を得る（付録1）．しかし、$R_c(\Lambda)$はΛに関して非線形であるため、前節の割り当て問題は、非線形0/1整数計画問題となり、ＮＰ完全であることが知られており[29]、したがって、直接解を得ることは困難である．そこで特別な場合として、以下のものを考え、解析的に解が得られることを示す．

３．４　均質プロセスに対するプロセス割り当て問題

３．４．１　モデル

ここでは、均質なプロセスに対する割り当て問題を考える[32][33]．つまり、各プロセスはすべて等しい要求処理量ａを持ち、他のプロセスと等しい確率ｐで通信するとする．このような均質なプロセスモデルとして表現されうるものとして，グラフの諸問題に対するdiffusing algorithm[17]や線形方程式系の解法のプロセス

表現[10]等が挙げられる．[17]では，グラフの辺の数がある程度大きくなると，ノードに対応した各プロセスはほぼ等しい確率で通信すると考えられ，その処理時間も各ノードについてほぼ等しい．[10]でも，方程式における0でない係数が増えるに従い同様のことが言える．

プロセスは全て均質であるとすると、どのプロセスをどのPEに割り当てるかではなく、各PEに割り当てられるプロセス数だけでジョブ全体の処理時間は決定される．しかも、各PEに不均等に割り当てるよりも、なるべく均等に割り当てる方がジョブ全体の処理時間を小さくできることから、このPEへのプロセス割り当て数は、使用するPE数によって一意に決定できる．したがって、前述のように通信量最小の割り当て（$n_p=1$）と、1PE当たり1プロセス割り当て（$n_p=M$）との間に、(3.7)式のジョブ全体の処理時間を最小にする最適なPE数が存在することが予想できる．このような考察に基づき、均質なプロセスの場合に於ける最適PE数を求める手法を以下に述べる．ここでは平均外部プロセス間通信時間$Rc(\Lambda)$を漸近線で近似することにより、問題を簡単化した．

3．4．2　問題の定式化および最適PE数

各PEにプロセスを均等に割り当てることにし、そのときのPE数をn_pとすると、一PE当たりのプロセス数m_pは全てのPEについて等しく、

$$m_p = [M / n_p] \qquad (3.10)$$

[x] はxを越えない最大の整数

で与えられる（実際には，n_pがMの約数でない場合，すべてのプロセスを均等に割り当てることは出来ないが，Mが大きくなると，この差は無視できると思われる．従って，以下では近似値として(3.10)式の値を用いている．この事については，より詳しく3.6 考察で触れる）．このとき、一単位の処理を終えたときにCNにアクセスする確率λ_iおよびローカル・メモリーにアクセスする確率λ'_iは、全てのPEについて等しく(3.4),(3.5)式より、

$$\lambda_j = \lambda = (M - m_p)\ p \qquad j = 1 \ldots n_p \qquad (3.11)$$

$$\lambda'_j = \lambda' = (m_p - 1)\ p \qquad j = 1 \ldots n_p \qquad (3.12)$$

これらから、全てのPEについて割り当てられた全プロセスの平均処理時間R_jは等しく、また、$R_c(\Lambda)$はn_pの関数として$R_c(n_p)$と書けるから、

$$T = \max(R_j) \qquad j = 1 \ldots n_p$$

$$= m_p a\ (1 + \lambda R_c(n_p) + \lambda' 1 / \mu_{lm}) \qquad (3.13)$$

となる.

ここで、$R_c(n_p)$に対して近似を導入する. 図3.2のセントラル・サーバー・モデルにおける遅延は漸近的に次のようになる[31].

$$R_c(n_p) = 1 / \mu_{cn} \qquad n_p \leqq n^*$$

$$n_p / \mu_{cn} - \lambda^* \qquad n_p > n^* \qquad (3.14)$$

n^*は[31]において、飽和点として定義されている値であり、

$$n^* = 1 + \mu_{cn} / \lambda^* \qquad (3.15)$$

と与えられる. ここで、λ^*はセントラル・サーバーへの到着率であり(3.8)式のλ^{PE}_jに相当し、PE数によって単調に増加するが、これを外部プロセス間通信確率の最大値で置き換える. したがって、

$$\lambda^* = \max(\lambda) = (M - 1)\ p \qquad (3.16)$$

$n_p \leqq n^*$、$n_p > n^*$のそれぞれの場合についてTを最小にするPE数n^{opt}_A、

$n^{opt}{}_B$を求めてみる．

$n_p \leqq n^*$のとき、(3.14)式より、平均外部プロセス間通信時間$R_c(n_p)$はn_pに対して一定となり、平均ジョブ実行時間Tは、

$$T = Ma / n_p \left(1 + \lambda \left(1 / \mu_{cn}\right) + \lambda' 1 / \mu_{lm}\right)$$

$$\frac{Ma\{((M-\eta)\rho+1)n_p + M(\eta-1)\rho\}}{n_p^2} \qquad (3.17)$$

ただし、

$\rho = p / \mu_{cn}$　　$\eta = \mu_{cn} / \mu_{lm}$

(3.17)式より、明かにTはn_pに関して単調減少で、Tを最小にするPE数$n^{opt}{}_A$は、

$$n^{opt}{}_A = \min(n^*, M) \qquad (3.18)$$

で与えられる．

次に、$n_p > n^*$のとき、平均外部プロセス間通信時間$R_c(n_p)$は、(3.14)式よりn_pに対して線形に増加し、平均ジョブ実行時間Tは、

$$T = \frac{a\{M\phi n_p^2 - ((M+\eta)\phi+1)n_p + M(\eta\phi+M)\}}{(M-1)n_p^2} \qquad (3.19)$$

但し、

$\phi = M(M-1)\rho$

これから、最小値を求めるために(3.19)式を微分し0 と置くと、

$$n^{opt}{}_B = \frac{2M(\eta\phi+M)}{(M+\eta)\phi+M} \qquad (3.20)$$

上式より平均ジョブ実行時間Tは、$n_p \leqq n^*$では$n^{opt}{}_A$、$n_p > n^*$では$n^{opt}{}_B$に

従う．したがって、n^{opt_B}がn^*より小さければ、n^*が最適解となる．最適PE数は結局、

$$n^{opt} = \min(\max(n^*, n^{opt_B}), M) \qquad (3.21)$$

$$n^* = 1 + M/\phi \qquad (3.22)$$

となる．特に、$\eta = 1$の場合、つまり、内部プロセス間通信時間と外部プロセス間通信におけるCN占有時間の平均が等しいときには、$n^{opt} > n^*$なるρが存在し、それをρ^*とすると、最適PE数は、

$$n^{opt} = \begin{cases} M & \rho \leqq \rho^* \\ \dfrac{2M((M-1)\rho+1)}{(M^2-1)\rho+1)} & \rho > \rho^* \end{cases} \qquad (3.23)$$

$$\rho^* = 1/(M-1)^2$$

3．5　実行時間が異なるプロセス割り当て問題への拡張

前節でプロセスの要求処理量および通信確率が等しいときの最適割り当てを求めたが、ここではその拡張として要求処理量のみが異なり各プロセス間の通信確率が等しいとしたときの最適割り当てを求める近似アルゴリズムを提案する[34]．ここでプロセスｉがa_iの要求処理量を持つとする．(3.11)、(3.12)式より

$$\lambda_i = (M - m_i)p \qquad (3.24) \qquad \lambda'_i = (m_i - 1)p \qquad (3.25)$$

となる．ここでm_iはPE ｊに割り当てられているプロセス数である．平均ジョブ実行時間を最小にするために、まず、外部プロセス間通信確率をできるだけ等しくし、ロードバランスを計る．(3.24)、(3.25)式よりプロセス間通信確率λ_i、λ'_iはPE ｊ

に割り当てられたプロセス数m_iのみに依存するからすべてのPEに割り当てるプロセス数を等しくすることを目標とする．

今かりにすべてのPEにおいて割り当てられたプロセス数を等しくできたとすると、その結果λ_j、λ'_jはすべてのPEにおいて等しくなり、(3.6)式の（）内、すなわち一単位の処理を終える平均時間はすべてのPEにおいて等しくなる．このとき平均ジョブ実行時間T（X）をできるだけ小さくするには各PEに割り当てられたプロセスの要求処理量の総和をできるだけ等しくすればよい．これらを同時に実現するため次の補助問題を考える．

＜補助問題＞

条件

$$\sum_{j=1}^{n_p} x_{ij} = 1 \qquad i=1...M$$

$$\left(\max_j \left(\sum_{i=1}^{M} a_i x_{ij}\right) + \max_j \left(\sum_{i=1}^{M} x_{ij}\right)\right) \qquad (3.26)$$

を最小化せよ

この補助問題は直接解くことが困難であるので、次のような近似アルゴリズムを用いる．

＜近似アルゴリズム＞

1．a_iを降順に分類し番号の付け替えを行う．

$$a_1 \leqq a_2 \leqq a_3 . . . \leqq a_M \qquad (3.27)$$

2．PE j において現在までに割り当てられているプロセスの要求処理量の総和をb_j

$$b_j = \sum a_i \qquad (3.28)$$

（和は現在PE j に割り当てられているプロセスすべてについて行う）

とする．このb_jを昇順に分類し、

$$b_{j1} \leqq b_{j2} \leqq b_{j3} \cdot \cdot \cdot \leqq b_{n_p} \qquad (3.29)$$

とする．

3．PE_{j1}にはプロセス1、PE_{j2}にはプロセス2というようにn_p個のプロセスを順に各PEに割り当てて行く．すなわち、現在割り当てられているプロセスの要求処理量の総和のできるだけ小さいものに大きな要求処理量を持つプロセスを割り当てる．

4．2、3をすべてのプロセスが割り当てられるまで繰り返す．

この方法は単純であるが、各プロセスの要求処理量の分散がさほど大きくないときには補助問題に対するかなりよい近似を与えると思われる．

ここまでは、割り当てるPE数n_pは与えられているものとしてきたが、次にこのn_pを決定する方法について述べる．ここで全プロセスの要求処理量の平均値を$\bar{a}$、

$$\bar{a} = \sum_{i=1}^{M} a_i / M \qquad (3.30)$$

とすれば、与えられた全プロセスがすべて等しい要求処理量$\bar{a}$、プロセス間通信確率pを持つ均質プロセスに対する割り当て問題を考える．このようなプロセス割り当て問題に対して(3.21),(3.22)式を適用して最適PE数n^{opt}を求めることができる．上で示した近似アルゴリズムはあるn_pに対して、各PEに割り当てられたプロセス数が等しく、要求処理量の総和も等しいような割り当てを与えるから、$n_p = n^{opt}$のときに平均ジョブ実行時間は最小値をとると考えられる．つまり、要求処理量が異なるプロセス割り当て問題に対する近似アルゴリズムは次のようになる．

＜要求処理量が異なるプロセス割り当て問題に対する近似アルゴリズム＞

1．(3.30)により全プロセスの要求処理量の平均$\bar{a}$を求め、プロセス数M、

プロセス通信確率pである均質プロセス割り当て問題を(3.21),(3.22)式を用いて解き、最適PE数n^{opt}を求める.

2．$n_p = n^{opt}$として補助問題を近似アルゴリズムによって解き、割り当てXを求める.

3．6　考察

3．6．1　均質プロセスに対する割り当て問題

ここでは、(3.21),(3.22)式を用いて求めた最適プロセッサ数(apprx) と、待ち行列網モデルによって求めたRc(Λ) を(3.7) 式に代入しながら、逐次探索によって求めた最適プロセッサ数(exact)とを比較する．システムの負荷を表わす測度としてρを用いる．

いくつかの負荷に対して、近似モデルによる最適解n^{opt}と待ち行列網理論による厳密解から求めた最適解をまとめたのが表3.1である．その他のパラメータは以下のとおりである．

プロセス要求処理量　$a = 10$

CN占有時間の平均　$1 / \mu_{cn} = 1/0.5$

$\eta = 1$

表3.1から、プロセス数M＝30、100 に関して、二つの方法により求めた最適プロセッサ数と、そのときの平均ジョブ実行時間は、ほぼ一致していることが分かる．(3.23)式より、最適プロセッサ数がプロセス数より小さくなるρは、M＝30(100)についてそれぞれ、ρ＞0.00119,(0.000102)である．そのためM＝30(100)のときρ＝0.001(0.0001)ではn^{opt}＝Mが最適解となっている．

表3.1　最適PE数

M=30				
ρ	n^{opt}		T	
	apprx	exact	apprx	exact
0.001	30	30	10.29	11.03
0.007	10	10	59.19	58.53
0.03	4	5	233.93	227.35
0.1	3	3	721.03	697.69

M=100				
ρ	n^{opt}		T	
	apprx	exact	apprx	exact
0.0001	100	100	10.10	10.78
0.0003	51	50	29.61	29.59
0.001	20	20	97.22	95.49
0.005	6	6	456.10	448.14

図3.4-a,b,cに、M＝30、ρ＝0.001,0.03、0.1 のときの平均ジョブ実行時間Tを、近似解、厳密解両方について、プロセッサ数n_pに対してプロットしたものを示す．ρが大きくなると、端数のためプロセッサが必ずしも均等に割り当てられなくなる影響が大きくなり、厳密解では、Mがちょうどn_pの倍数になっていないときの遅延が極端に大きくなり、近似モデルと合わなくなる．この傾向はρが大きくなるほど増える．しかし、それでも最適プロセッサ数はよく一致することが分かる．図3.4からわかるように、プロセッサ数をn^{opt}まで増やすことによって、平均ジョブ実行時間は飛躍的に改善されるが、それ以上増やしても、もはやあまり改善されない．ρのn^{opt}に対する影響を、種々のηに対して示したのが図3.5である．ηはプロセッサ間通信によるオーバーヘッドを表わし、η=0のときオーバーヘッドは最大となる．図3.5から見る限り、M=30,100 のそれぞれの場合について、ηによる最適プロセッサ数n^{opt}への影響はほとんどない．これはηがMに比して非常に小さい($0 \leq \eta \leq 1$)ことから、(3.20)式よりわかる．一方、ρに対する平均ジョブ実行時間を、幾つかのηに対して示したのが図3.6-a,bである．このように、ηによって最適PE数はほとんど影響を受けないが、平均ジョブ実行時間は大きく影響を受ける．図3.7は、プロセス数Mのときのn^{opt}によって得られた平均ジョブ実行時間を、Mに対して示したもので、ρが0.03まではプロセス数を増やしても、平均ジョブ実行時間はそれほど増加しないが、ρ＞0.03になると平均ジョブ実行時間はプロセッサ数に対して急激に増加する．以上、考察より明かになったことをまとめると以下のようになる．

・PE数をn^{opt}まで増やすことによって、平均ジョブ実行時間は飛躍的に改善されるが、それ以上増やしても、もはやあまり改善されない．

・オーバーヘッドηによる最適PE数n^{opt}への影響はほとんどないが、平均ジョブ実行時間は大きく影響を受ける．

・ρが0.03までは、プロセス数を増やしても平均ジョブ実行時間はそれほど

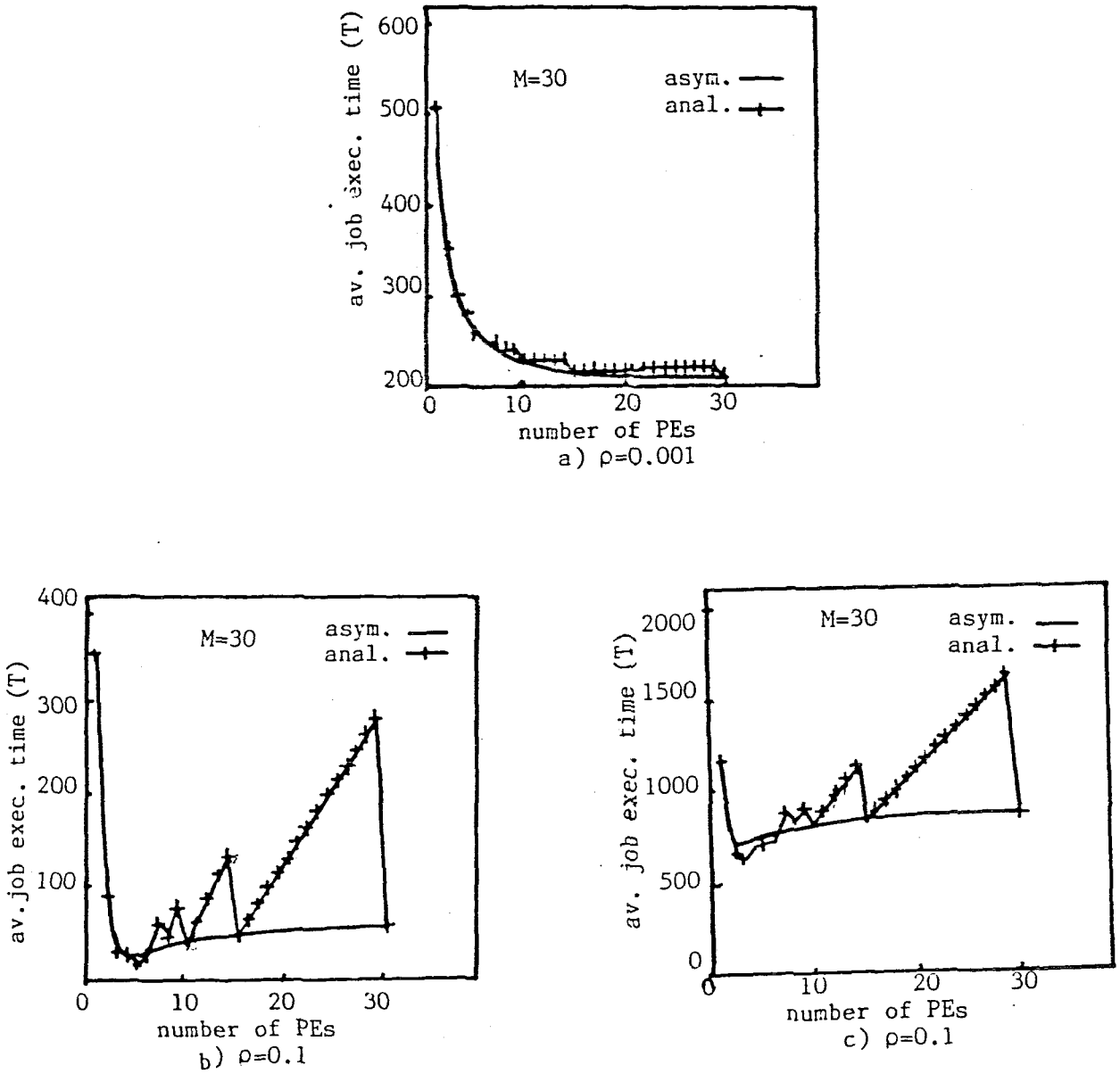


図3.4　ＰＥ数に対する平均ジョブ実行時間の変化

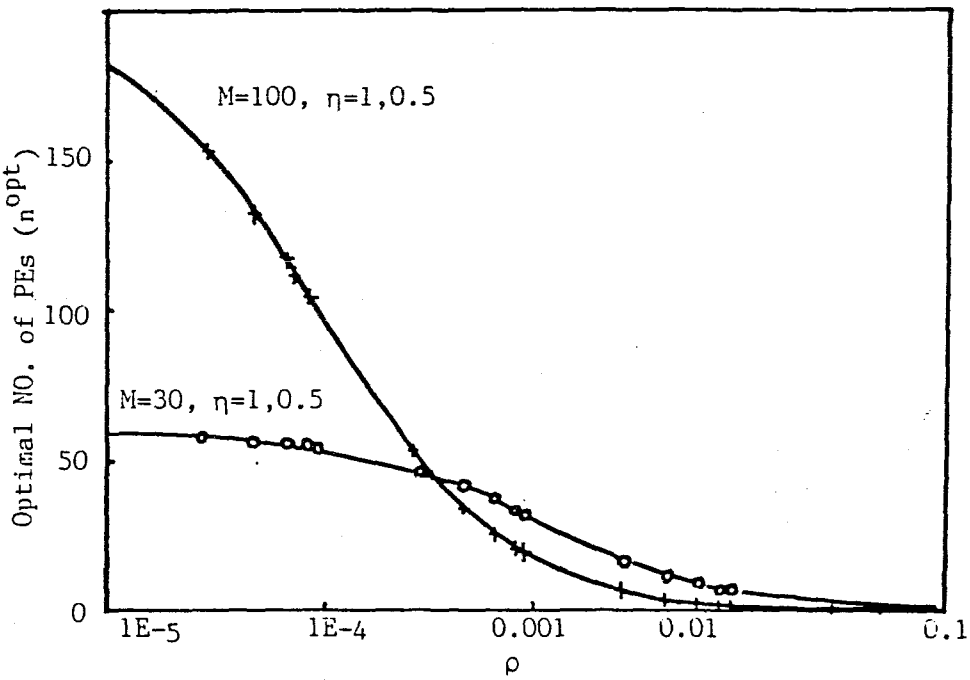

M=100, η=1,0.5
M=30, η=1,0.5
Optimal NO. of PEs (n^{opt})
150
100
50
0
1E-5
1E-4
0.001
0.01
0.1
ρ

図3.5　最適ＰＥ数

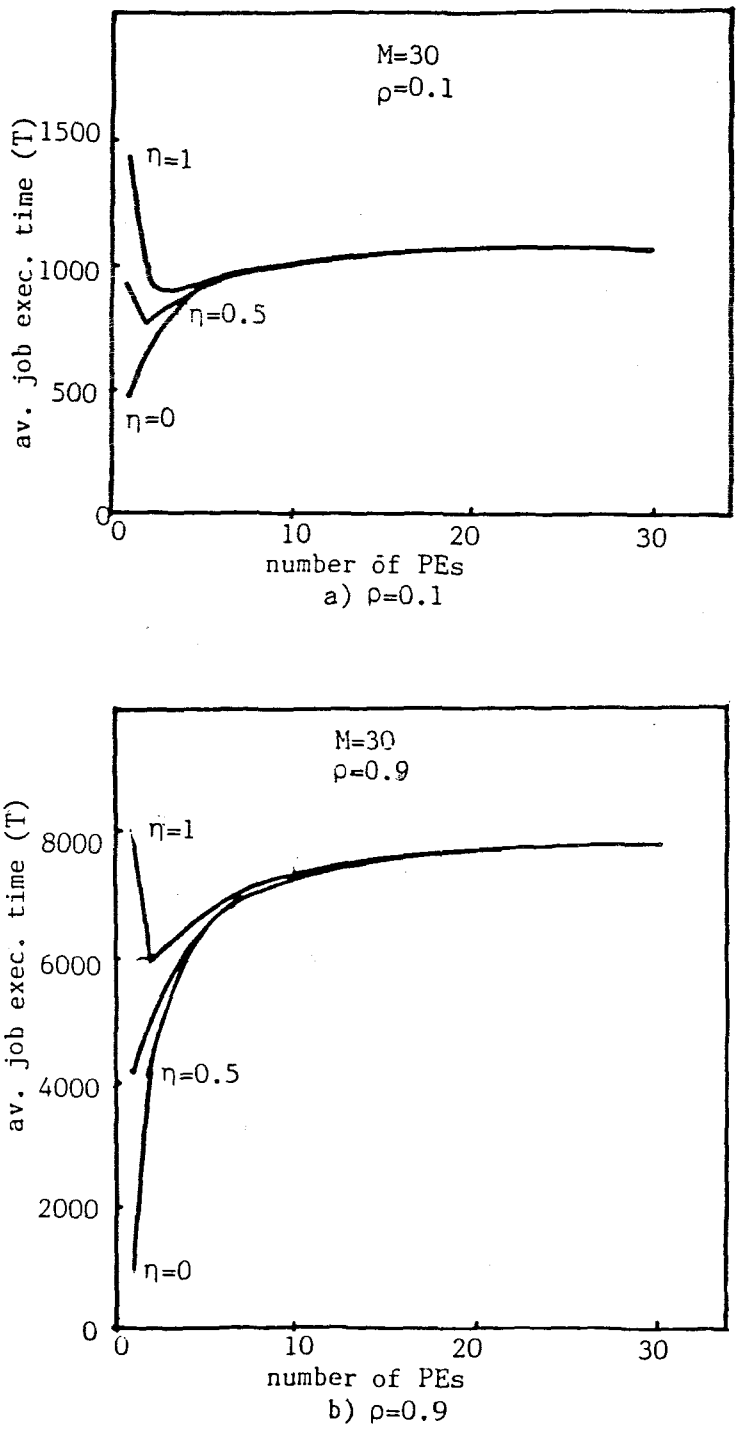


図3.6　ＩＰＣが与える影響

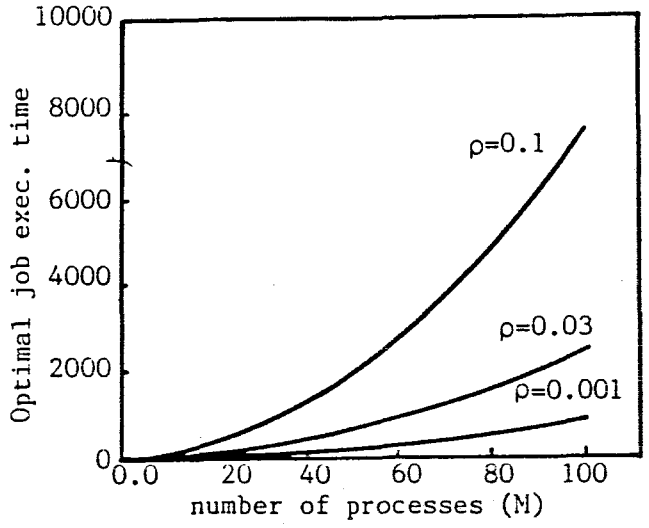


図3.7　最適ＰＥ数の時の平均ジョブ実行時間

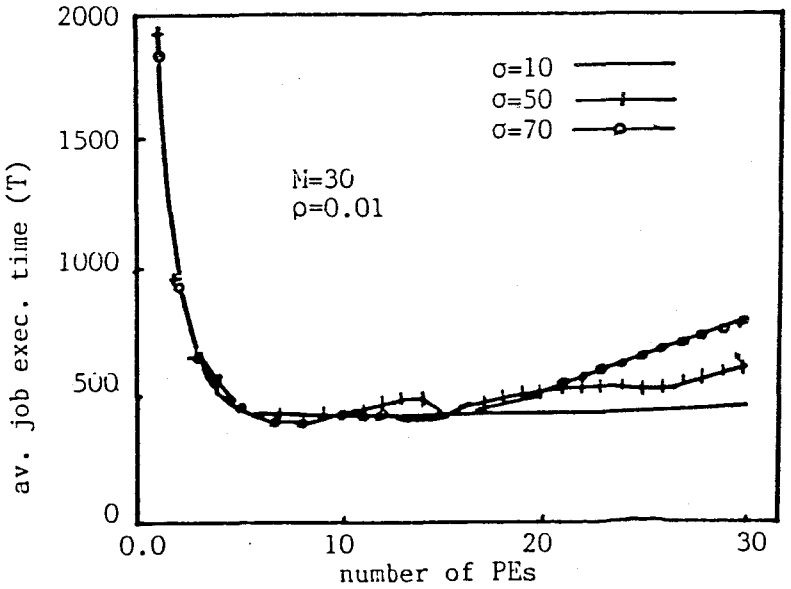


図3.9　平均ジョブ実行時間（均質でないプロセス割り当て）

増加しないが、$\rho > 0.03$になると平均ジョブ実行時間はプロセッサ数に対して急激に増加する．

３．６．２　要求処理量が異なるプロセスの割り当て問題

表3.2は要求処理量のみが異なるプロセスに対して3.5で示した近似アルゴリズムにより得られた近似解のPE数（n^{opt}）とそのときの平均ジョブ実行時間（T(X)）を示している．ここで各プロセスの要求処理量は平均50、分散処理システム10、50、70の正規分布より発生させる．その他のパラメータは前節の均質プロセスの場合と同じである．

表3.2　実行時間が異なるプロセス割り当て問題の最適ＰＥ数と平均ジョブ実行時間

M=30			
ρ=0.01		n^{opt}	T
σ=10	proc1	7	431.43
	proc2	8	415.66
σ=50	proc1	7	401.12
	proc2	8	396.38
σ=70	proc1	7	434.38
	proc2	7	

M=100			
ρ=0.001		n^{opt}	T
σ=10	proc1	19	508.84
	proc2	83	372.34
σ=50	proc1	19	491.09
	proc2	53	462.66
σ=70	proc1	19	522.65
	proc2	57	466.21

ρ=0.03		n^{opt}	T
σ=10	proc1	4	1097
	proc2	5	1081
σ=50	proc1	4	1064
	proc2	5	1055
σ=70	proc1	4	1150
	proc2	5	1139

ρ=0.005		n^{opt}	T
σ=10	proc1	5	2182
	proc2	51	2026
σ=50	proc1	5	2237
	proc2	53	2047
σ=70	proc1	5	2344
	proc2	56	2087

proc1: local optimum when np=nopt
proc2: grobal minimum from np=1 to np=M

表3.2には二つの手続き（Proc1, Proc2)によって求めた値を示している．図3.8を用いてこの二つの手続きを説明する．図3.8の点線は要求処理量が(3.30)式のａに等しい均質プロセスのPE数に対する平均ジョブ実行時間の漸近的特性を表している．この均質プロセスに対して$n_p = n^{opt}$のとき、平均ジョブ実行時間は最小値をとる．Proc1は3.5の近似アルゴリズムに相当し、$n_p = n^{opt}$のとき

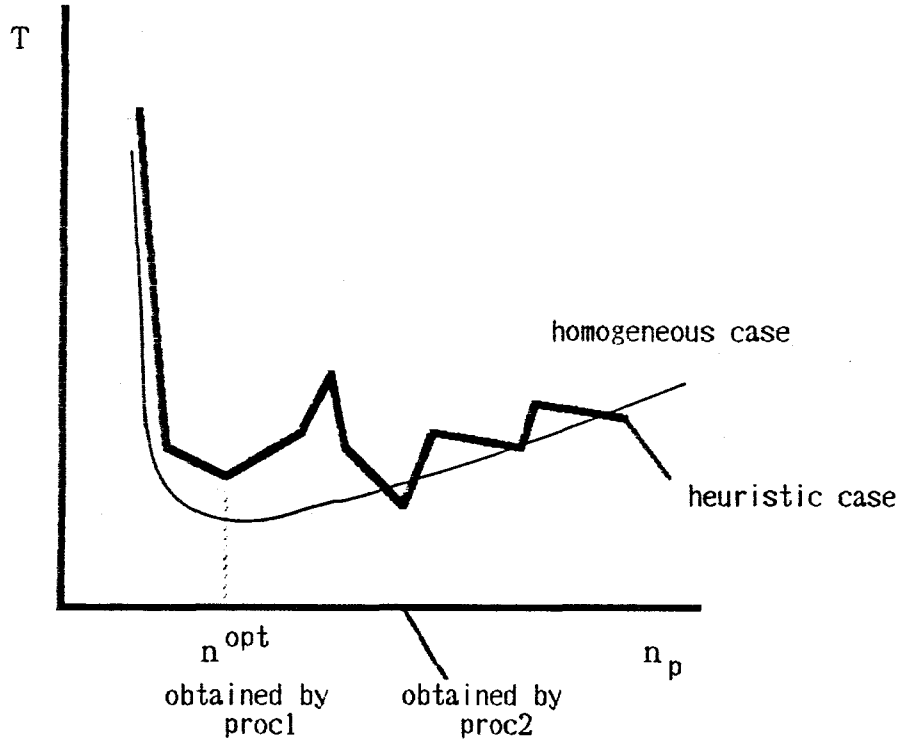


図3.8　手続き1,2の説明

の補助問題の解を近似解としている．3.5の近似アルゴリズムを拡張しn_pを1からMまで変化させて補助問題を解き、得られた割り当てから求めた平均ジョブ実行時間を示したのが図3.8である．Proc2はこのうちの最小値をとる割り当てを近似解とする．Proc1は実線上の一つの局所最適解を表し、Proc2は広域最適解を表している．表3.2にはProc1,2より得られた近似解のPE数と平均ジョブ実行時間を示している．表からM＝30の場合にはProc1、つまり3.5の近似アルゴリズムはほぼ最適解を与えていることがわかる．しかし、M＝100の場合には要求処理量の分散の影響が大きくなり、必ずしも$n_p = n^{opt}$のときに平均ジョブ実行時間は最小値をとらない．これは均質プロセスの場合、外部IPC時間を最大値で近似しているため（3.28）式の最適PE数は小さく見積もられているためである．しかし、この場合でも平均ジョブ実行時間TはProc2で得られた最小値と大きな差異はなく、近似解として充分用い

ることができる．

図3.9に$\rho=0.01$、M＝30および$\rho=0.001$、M＝100の場合にn_pを1からMまで変化させたときの3.5の近似アルゴリズムによって求めたTの値を示す．どちらの場合も3.4の均質プロセスに対する近似解とほぼ一致しており、3.5の近似アルゴリズムの有効性が示された．

4．　処理とＩＰＣの同時実行を考えたプロセス割り当て問題

ここでは、プロセスに関してより一般的なモデル化を行う[35]．先のモデルに対し次のような拡張を行う．３章のモデルでは一定のIPC間隔を仮定していたが、ここでは平均IPC間隔は可変とする．

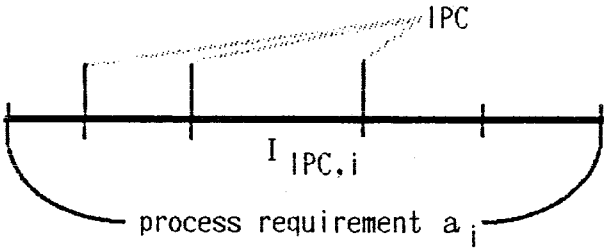


図4.1　プロセスｉのパラメータ間の関係

$$\sum_{i=1}^{M} p_{ij} = 1 \qquad i=1...M \qquad (4.1)$$

とする．したがって、プロセスｉにおいて次の値が知られているとする（図4.1）．

- 平均処理時間　I_i
- 平均IPC回数　a_i
- 平均IPC間隔　$I_{IPC,i}$

ただし、これら三つの量の間には

$$I_i = I_{IPC,i} \times a_i \qquad (4.2)$$

という関係があるとする．したがって、プロセスｉにおけるIPCから次のIPCまでの平均処理時間（$1/\mu_{CPU,i}$）は

$$1/\mu_{CPU,i} = I_{IPC,i} \qquad (4.3)$$

となる．

分散処理システムにおいて、一つのPEに割り当てられた複数のプロセスはプロセ

ッサ・シェアリング(PS)で処理される．プロセスｉのIPC間隔は平均１／$\mu_{CPU,i}$の指数分布に従うとする．ここでは、各プロセスはIPCを行うまで、CPUで処理される．LMまたはCNを通じてのIPCに必要な時間はそれぞれ平均１／μ_{lm}、１／μ_{cn}の指数分布に従うとする．あるプロセスがIPCを行っている間、CPUは他のプロセスの処理を行うことができる．つまり、IPCは内部IPC、外部IPC共にCPUにおける処理とオーバーラップして行われる．

パラメータをまとめると表4.1のようになる．

表4.1　分散処理システムパラメータ

プロセス数	M
プロセス間通信確率	pij
平均ＩＰＣ回数	ai
割り当て行列	xij
LMを介しての平均ＩＰＣ時間	1/μlm
CNを介しての平均ＩＰＣ時間	1/μcn
CPU処理時間	1/μcpu,i

４．１　分散処理システムの待ち行列網によるモデル化

ある割り当てＸに対するPE数をn_p、PEｊに割り当てられたプロセス数m_jとすると、

$$m_j = \sum_{i=1}^{M} x_{ij} \qquad j=1...n_p \qquad (4.4)$$

ここで分散処理システムを閉じた待ち行列網としてモデル化する．各PEにおいて、CPUをPSの指数サーバー、LMをFCFSの指数サーバーとする．PEｊにおいてプロセスｉが内部IPCを行う確率$p_{l,i}$は（以下、プロセスｉの割り当てられているPEｊは一意に決定できるので特に必要ない限りはｊを省略する）、

$$p_{l,i} = \sum_{k=1}^{M} p_{ik} x_{ij} x_{kj} \qquad i=1...M \qquad (4.5)$$

一方、PE j にあるプロセス i が外部IPCを行う確率 $p_{E,i}$ は

$$p_{E,i} = \sum_{k=1}^{M} p_{ik} x_{ij} (1 - x_{kj}) \qquad i=1...M \qquad (4.6)$$

となる．(4.1)式より、$p_{I,i} + p_{E,i} = 1$ である．プロセス i が処理を終えてシス

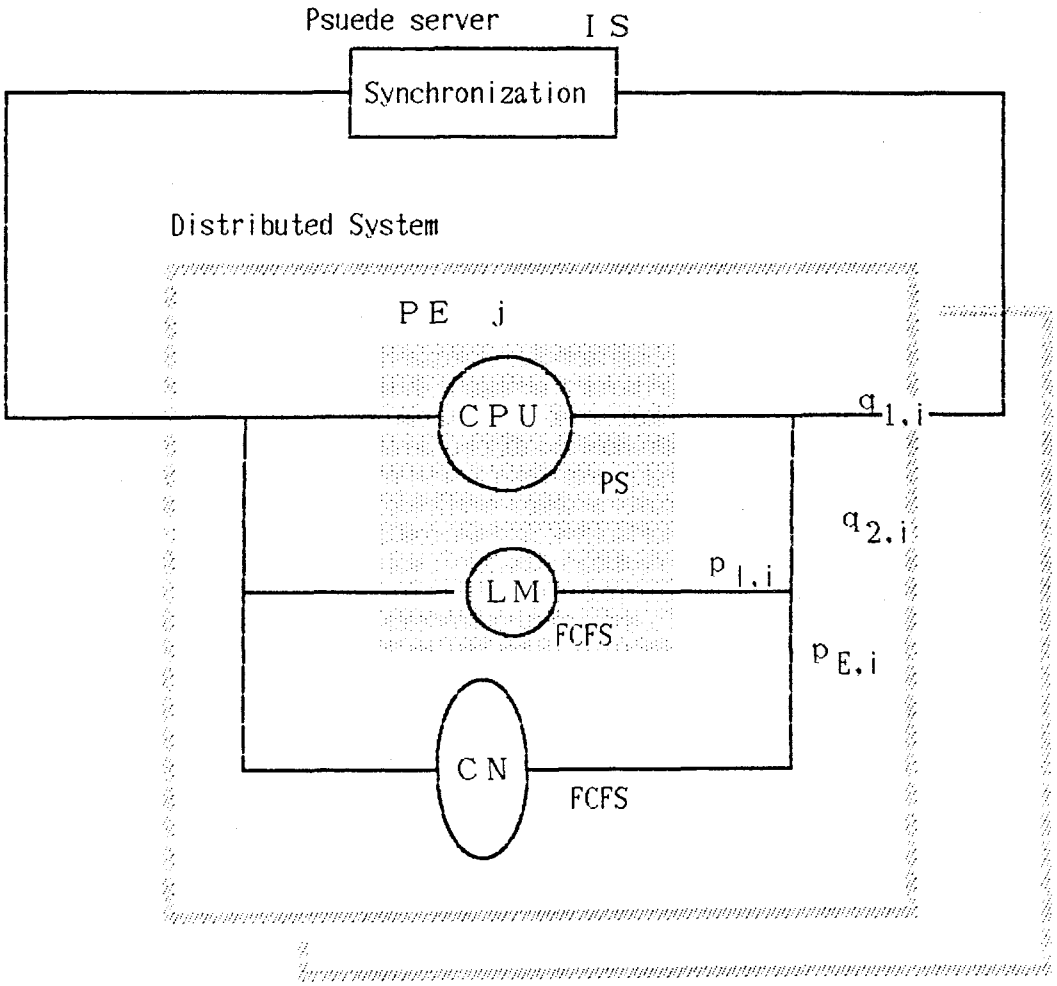


図4.2　分散処理システムの待ち行列モデル

テムから出て行く確率$q_{1,i}$および処理の続行を受ける確率$q_{2,i}$は

$$q_{1,i} = 1 / a_i \qquad i=1...M \qquad (4.7)$$

$$q_{2,i} = 1 - q_{1,i} \qquad i=1...M \qquad (4.8)$$

したがって、分散処理システムの待ち行列網モデルは図4.2のようになる（同期部分については次節で説明する）．ここでは分散処理システムに割り当てられるプロセスを客と考え、各々の客が別々のチェインをなすとして、単一のチェインだけを示している．PE j におけるCPUおよびLMはそこに割り当てられたm_j個のプロセスによって共有され、CNはM個の全プロセスによって共有されている．

4．2　プロセスの同期

一つのジョブに属するM個のプロセスは実際には同時に図4.2の分散処理システム内に到着し、並行に処理されシステムから出る．ジョブの終了は最後のプロセスがシステムから退去することによって起こる．このような特殊な客の分散処理システムにおける応答時間を求めるために、[36]で用いられている疑似サーバーの手法を導入した．

プロセスの同期は分散処理システムの外に疑似サーバーを置くことにより実現される．すなわち、到着において、各プロセスはこの疑似サーバーから同時に分散処理システムに到着する．処理を終え分散処理システムから退去したプロセスは一旦疑似サーバーに入り、他のすべてのプロセスが終了するまで待たされる．最後のプロセスが処理を終え疑似サーバーに入ると、再びすべてのプロセスが分散処理システム内に入ることができる．今、プロセス i の分散処理システムにおける在中時間を確率変数R_iで表すと、すべてのプロセスの終了時間すなわちジョブ実行時間D_0は、

$$D_0 = \max(R_1, R_2, \ldots, R_M) \qquad (4.9)$$

で与えられる．ここで、R_1，R_2．．．．，R_Mが平均r_1，r_2．．．．，r_Mの指数分布に従うとすれば、D_0の期待値、d_0、は

$$d_0 = E(D_0) = \sum_{i=1}^{M} 1/r_i - \sum_{i<j} 1/(r_i + r_j) + \cdots + (-1)^{M-1} \sum_{i_1<i_2<\ldots<i_M} 1/(r_{i_1} + r_{i_2} + \ldots r_{i_M}) \quad (4.10)$$

となる．したがって、プロセスｉの疑似サーバーでの待ち時間の期待値d_iは

$$d_i = d_0 - r_i \qquad i=1\ldots M \quad (4.11)$$

となる．疑似サーバーのサービス規約を無限サーバー(IS)とすると、図4.2はBCMPクラスの閉じた待ち行列網となり、ISサーバーにおける待ち時間の初期値d_i(i=1...M)を適当に与えれば、図4.2の待ち行列網モデルを解くことにより、プロセスの平均在中時間r_i(i=1...M)を求めることができる．すると、(4.10),(4.11)式より新たなd_i(i=1...M)が求まる．このようにして、繰り返しにより平均ジョブ実行時間d_0を求めることができる．

4．3　プロセス割り当て問題の近似解法

ここでは、IPC確率p_{ij}、平均IPC回数a_iおよび平均IPC間隔$1/\mu_{cpu,i}$がプロセスごとに異なる一般の場合におけるプロセス割り当てについて考える．この場合に先に述べたジョブ実行時間の平均値を目的関数とし、これを最小にするプロセス割り当てを見出すことはきわめて困難である．そこでここでは新たに次のような割り当て問題を考える．

(H.1)　PEに割り当てられるプロセスの平均IPC回数a_iの総和に、すべてのPEに対

し共通の上限dを設ける．すなわち、

$$\sum_{i=1}^{M} a_i x_{ij} \leqq d \qquad j=1...n_p$$

これを満たす下で、

(H.2)　異なるPEに割り当てられたプロセス間のIPCを最小化する．

PEに割り当てられたプロセスの平均IPC回数の総和に共通の上限dを設けることにより、単一のPEにプロセスが集中するのを制限することができる．つまり、各PEごとにロードバランスが実現できることになる．上記の割り当て問題から求められた割り当てを元の平均ジョブ実行時間を最小にする割り当てに対する近似解として採用する．この問題は比較的簡単な整数計画問題として定式化でき、分枝限定法を用いることにより元の問題よりはるかに簡単に解くことができる[29]．

したがって、ここではまず、(H.1)，(H.2)を同時に満たす割り当てを求める問題を整数計画問題として定式化する．次にこの整数計画問題に対する最適解を分枝限定法を用いて求める手法を示す．さらに近似解を改良する手法について述べる．

4．3．1　割り当て問題の定式化

まず、この場合のプロセス割り当て問題を0/1整数計画問題として次のように定式化する[29]．

変数

$x_{ij}=0$　　プロセスiをPEjに割り当てる．

　　1　　その他　　(4.12)

目的関数

$$\min \quad f(X) = \sum_{k<l} \sum_{i \neq j} \left(\sum \sum t_{kl} x_{ki} x_{lj} \right) \qquad (4.13)$$

制約条件

x_{ki}は0／1変数 (4.14)

$$\sum_{i=1}^{n_p} x_{ij} = 1 \quad \text{for } 1 \leqq i \leqq M \qquad (4.15)$$

$$\sum_{i=1}^{M} a_i x_{ij} \leqq d \quad \text{for } 1 \leqq j \leqq n_p \qquad (4.16)$$

但し、

$$t_{kl} = \begin{cases} p_{kl} / \mu_{cpu,k} + p_{lk} / \mu_{cpu,l} & k < l \\ 0 & \text{その他} \end{cases} \qquad (4.17)$$

d：PEあたりの平均ＩＰＣ回数の総和の上限

(4.17)式のt_{kl}はプロセスkとプロセスlの間の単位時間あたりのプロセス間通信確率を表している．ここで、p_{kl}はIPCあたりであるので、これを平均IPC間隔$1/\mu_{cpu,k}$で正規化している．このようにすると目的関数(4.13)により、(H.2)すなわち、プロセス間の通信量の総和を最小にすることが実現できる．dを変えることにより、プロセスのPEへの集中度を調節することができる．dの決定方法については後述することとし、以下dが与えられたとして議論を進める．

４．３．２　分枝限定法による解の探索

(4.12)-(4.17)の0/1整数計画問題を分枝限定法を用いて解く．分枝限定法における解の探索において、プロセスを一回に一つづつ割り当てて行く．探索は深さ優先で行う．例えば三つのプロセスを二つのPEに割り当てるとすると探索木は図4.3

のようになる．ここで木の頂点は割り当てを行うプロセスを、木の枝がそのプロセスを割り当てるPEを示している．中間の頂点は一部のプロセス割り当てのみが完了した状況を現しており、これを部分割り当てと呼ぶ．一方、葉頂点はすべてのプロセスを割り当てた一つの実行可能解を現し、これを完全割り当てと呼ぶ．プロセス割り当てはこのような木をたどって行くことにより、以下のように進んで行く．

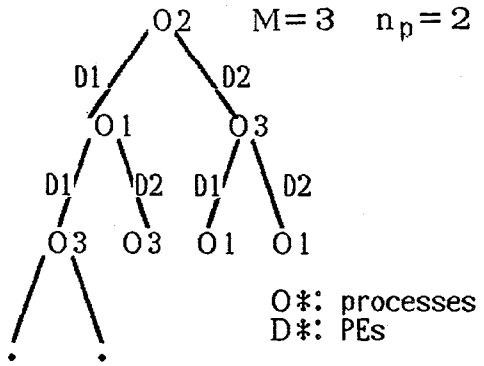


図4.3 割り当ての探索木

1） 割り当てるプロセスの選択

まだ、割り当てられていないプロセスを一つ選ぶ．このプロセスをｉとする．

2） PEの選択

まだ、テストされていないPEを選ぶ．このPEをｊとする．

3） プロセスｉをPEｊに割り当てた後の部分割り当てＶによる目的関数値の下限ｇ（Ｖ）を計算し、完全割り当てによって現在までに得られた最良の目的関数値ｚと比較する．

$$z \leqq g(V) \qquad (4.18)$$

ならば以下の探索を中止し、別の選択を行う．

４．３．３ 発見的手法

探索において候補となるPEおよびプロセスの選択に発見的手法を導入することにより最適解を早く求めることが期待できる．そのため、PE、プロセスの選択は次のように行う．

(H.3) プロセスの選択

まだ割り当てられていないプロセスのうち、

$$a_i = \max_k (a_k) \qquad (4.19)$$

すなわち、平均IPC回数が最大となるプロセスｉを選ぶ．

(H.4)　PEの選択

$$\max_j \left(\sum_{l \in S_j} t_{il} \right) \qquad (4.20)$$

なるPEｊを選ぶ．ここで、

$S_j = \{ k \mid k \neq i$、部分割り当てX_iのうちPEｊに割り当てられているプロセスの添字の集合$\}$

すなわち、割り当てようとするプロセスｉと現在までにPEｊに割り当てられているプロセスとの通信量の総和が最大となるPEｊを選ぶ．

(H.3)はビンパッキング問題のFFD(First Fit Decreasing)アルゴリズムに相当し、ロードバランスを実現しながら、使用するPE数を最小にできる[29]．また、(H.4)を適用することにより、プロセスｉを割り当てることによる目的関数値の上昇を最小限に押さえることができる．

４．３．４　目的関数値の下限の導出

現在の部分割り当てより派生するすべての完全割り当てに対する目的関数の下限を計算することにより、探索する範囲を限定することができる．部分割り当てＶに対する目的関数値の下限として

$$g(V) = f(V) + \sum_{i \in U} \left(\sum_{j} \sum_{l \in S_j} t_{il} - \max_j \left(\sum_{l \in S_j} t_{il} \right) \right) \qquad (4.21)$$

を用いる．ここで、

U＝｛ｉ｜まだ割り当てられていないプロセスの添字の集合｝

すなわち、(4.16)式の制限を考えずに(H.4)を適用して、残りのすべてのプロセスを割り当てたときの各プロセスによる目的関数の増加量と部分割り当てＶによる目的関数値の和である．

４．３．５　ｄの決定方法

ここまでｄの値は、与えられたものとして進めて来た．ここでは各PEに割り当てられるプロセスの平均ＩＰＣ回数の総和の上限であるｄを定める方法について述べる．(4.12)-(4.17)の整数計画問題は与えられたｄに対し通信量最小の割り当てを求める．ｄの範囲は次のようになる．

$$d_{min} = \max(a_i) \leqq d \leqq \sum_i a_i = d_{max} \qquad (4.22)$$

すなわち、ｄは最小値d_{min}において最大数のPEを使用し（＝Ｍ）、最大値d_{max}において最小数のPEを使用する（＝１）．(4.22)内のそれぞれのｄの値について先の分枝限定法により整数計画問題(4.12)-(4.17)を解くことができ、ロードバランスを実現しながら外部通信量最小の割り当てを得ることができる．ただし、このようにして得られた割り当てはあくまでも近似解であり、平均ジョブ実行時間を最小にするとは限らない．したがって、いくつかの適当なｄに対して整数計画問題(4.12)-(4.17)を解き、これらの割り当てのそれぞれに対し、4.2章の待ち行列網モデルを適用して平均ジョブ実行時間を計算し、それらの中から最小の平均ジョブ実行時間を与えるものを採用することにより、近似解を改良することができる．

したがって、一般的なプロセス割り当て問題の近似解法を以下のようにまとめることができる．

1) 調べるdの数をnとする．dのきざみΔdを

$$\Delta d = (d_{min} - d_{max}) / n \qquad (4.23)$$

とする．dの初期値を$d = d_{min}$とする．

2) 与えられたdに対して整数計画問題(4.12)-(4.17)を上記の分枝限定法を用いて解く．

3) 2)で求まった割り当てに対し、4.2章の待ち行列網モデルを適用し平均ジョブ実行時間を求める．

4) $d = d + \Delta d$

5) 2)-4)を$d = d_{max}$まで繰り返し、平均ジョブ実行時間を最小にする割り当てを近似解として採用する．

nを大きくすることにより調査する候補数が多くなり、近似解は良くなるが計算時間は増大する．したがって、要求される近似解の精度に対して適当なnの値を選ぶことができる．

4．4　考察

4.3章で示した一般の場合のプロセス割り当て問題に対する近似解法の得失を論じる．ここでは、すべてのプロセスの平均IPC間隔が等しいと仮定し（$1/\mu_{cpu,i} = 1/\mu_{cpu}$ for i=1...M）、プロセスｉがプロセスｊとIPCを行う確率は、平均0.5、標準偏差0.5の正規分布より発生させ、(4.1)式を満たすように正規化される．また、プロセスの要求処理量a_iは平均20、標準偏差20または2の正規分布より発生する．分散処理システムのパラメータは以下のとおり．

$\mu_{cpu}=0.1$

$\mu_{lm}=0.5$

$\mu_{cn}=0.15$または0.5

近似解法において探索する候補の数ｎを20とした．結果は分散処理システムがIPC依存型（$\mu_{cn}=0.15$）である場合と内部処理依存型（$\mu_{cn}=0.5$）である場合の二つについて行う．IPC依存型の場合、CNにおける処理時間がLMにおけるそれと比べて大きいため少ないPEを用いた割り当てが望まれ、一方要求処理量依存型の場合、CNにおける処理時間とLMにおけるそれが等しいため負荷をできるだけ分散させることが望まれる．さらに、それぞれの場合について、平均IPC回数の分散が大きい場合（高分散）と小さい場合（低分散）を比べ、平均IPC回数の分散の影響を考察する．高分散の場合、負荷を均等に分散するのが難しくなり、一方低分散の場合、均質なプロセスの割り当て問題により近づく．

表4.3　平均ジョブ実行時間（M=5）

a）ＩＰＣ依存型

		高分散	低分散
近似解	平均ジョブ実行時間	1283.1	797.7
	訪問ノード数	1597	2790
厳密解	平均ジョブ実行時間	1281.7	797.7
	訪問ノード数	15	51
最悪訪問ノード数		3125	

b）内部処理依存型

		高分散	低分散
近似解	平均ジョブ実行時間	984.9	544.7
	訪問ノード数	1597	2790
厳密解	平均ジョブ実行時間	984.9	544.7
	訪問ノード数	12	50
最悪訪問ノード数		3125	

表4.3にＭ＝5における4.3章で述べた近似解法および厳密解によって求めた割り当てに対する平均ジョブ実行時間と探索木における訪問頂点数を示す．厳密解は平

均ジョブ実行時間を目的関数として分枝限定法により求めた．表から近似解法によって求めた割り当てはIPC依存型で平均ジョブ実行時間の分散が大きい場合でも厳密解で求めた割り当てによる平均ジョブ実行時間に近い解が得られていることがわかる．低分散の場合および内部処理依存型の場合には近似解法によって最適解が得られている．表4.5には参考のためにｄによる制限を加えない場合の分枝限定法による訪問頂点数の最悪値を示している．近似解法においてはｎ＝20回も探索を繰り返しているにもかかわらず、ｄの上限および下限による限定操作により訪問頂点数が相当少ないことがわかる．厳密解では訪問頂点数はかなり少ないが一つの頂点を訪問する度に4.2章で示した待ち行列網モデルを解く必要があり、計算時間では近似解法のほうがはるかに少ない．例えば、Ｍ＝5で、IPC依存型高分散の場合、VAX750/unix上で近似解法46秒に対して厳密解は5分33秒かかる．低分散の場合で近似解法1分15秒に対して厳密解は約7分かかっている．表4.4にはＭ＝8の場合の近似解法による結果を示している．この場合、近似解法による訪問ノード数の削減効果はＭ＝5に比べてはるかに大きく、Ｍが大きくなるほど効率がよくなっていくことが分かる．また、平均IPC回数の分散が大きいほど上限ｄのために限定されるノード数は多い．計算時間は低分散の場合でもVAX750/unix上で2～3時間程度ですむが、厳密解を求めるのはほとんど不可能である．また、比較のために（）内にランダム・プロセス割り当てを行ったときの平均ジョブ実行時間を示している．これはランダムな10個の割り当てから求めた平均である．これから、ランダムに割り当てるよりも近似解法を用いることによって2割程度平均ジョブ実行時間が改良されていることがわかる．その効果はIPC依存型のようにCNに対する負荷が

表4.5　平均ジョブ実行時間（Ｍ＝8）

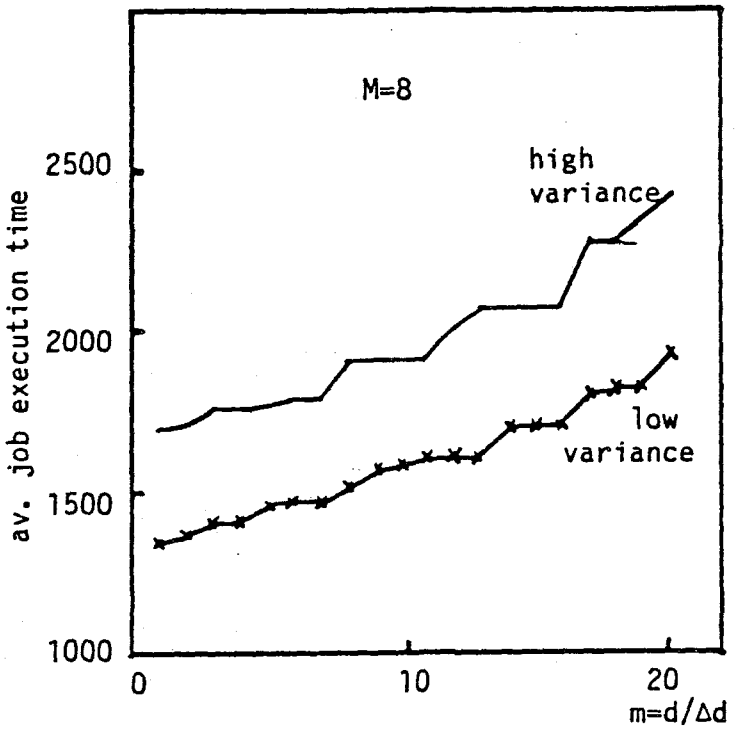
	高分散	低分散
ＩＰＣ依存型 平均ジョブ実行時間	2000.4 (2229.9)	1176.4 (1265.3)
訪問ノード数	35697	221641
内部処理依存型 平均ジョブ実行時間	1400.4 (1543.4)	719.2 (889.9)
訪問ノード数	35697	221641
最悪訪問ノード数	16777216	

()はランダム割り当てによる平均ジョブ実行時間

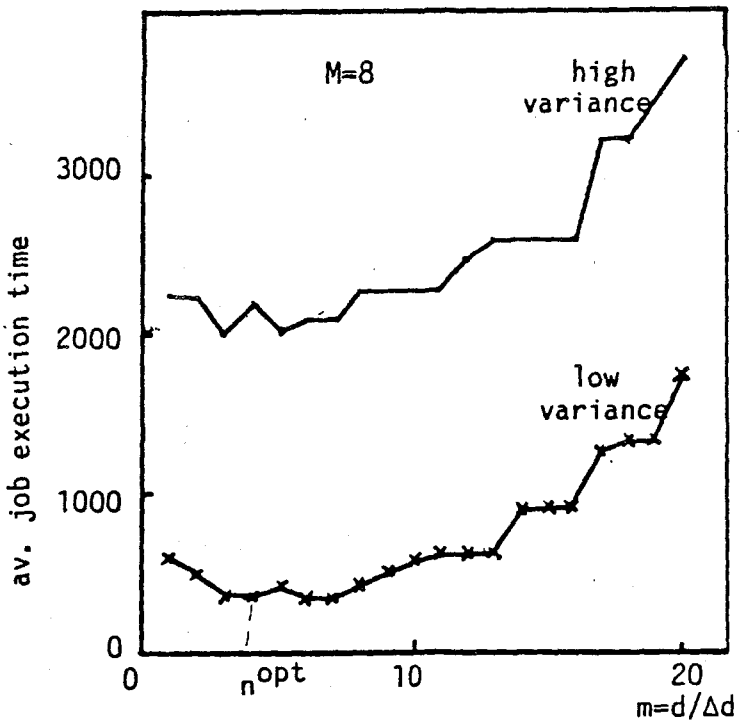
高いほど、また、各プロセスの平均IPC回数の分散が大きいほど高い．　図4.4-a,bにはM＝8、内部処理依存型、IPC依存型それぞれの場合の(4.16)式の平均IPC回数の上限の効果を表している．つまり、図は各PEに割り当てられるプロセスの平均IPC回数の総和の上限dを(4.22)式の範囲すなわち、d_{min}からd_{max}までΔdずつ変化させて各dの値について得られた近似割り当てに対する平均ジョブ実行時間を示している．ただし、横軸は各dをΔdで正規化した値m、すなわち、

$$m = d / \Delta d \qquad d_{min} \leqq d \leqq d_{max} \qquad (4.24)$$

を示しており、n＝20と(4.23)よりこの場合$0 \leqq m \leqq 20$である．dを増加させていくと、外部IPCを最小にすることを目的関数としているため、プロセスの割り当てられるPE数は減少して行く．内部処理依存型の場合（図4.4-a）、CNに対する外部IPCによる負荷が小さいためにPE数の大きいところ（dの小さなところ）で平均ジョブ実行時間は最小になる．特に、低分散の場合、各PEに一つづつプロセスを割り当てたものが最適解となっている．一方、外部IPC依存型の場合（図4.4-b）内部処理依存型よりも少ないPE数（dの大きなところ）で平均ジョブ実行時間は最小になる．このとき低分散の場合の最適PE数はn_p＝4である．探索する候補数nを増やせばそれだけ近似解を厳密解に近づけることができるが、計算時間も増加する．したがって、できるだけnを少なくすること、あるいは候補を調べる間隔Δdをできるだけ大きくとることが望まれる．図がdの変化に対して比較的滑らかであることからΔdを2～3倍して調べる候補数を減らしても、そこから得られる最適解はそれほど悪くならない．したがって、nの適当な値としては6～10程度で充分であることが言える．また、平均ジョブ実行時間の変化はdに対してほぼ凸であり、mを増やしながら局所最適解が求まった時点で探索を打ち切ってもよいことがわかる．



a) low load of CN case.



b) High load of CN case.

図4.4　平均ジョブ実行時間の上限dに対する変化

５．　分散処理システム評価シミュレータＳＥＤＳ

本章では分散処理システム評価シミュレータＳＥＤＳ（Simulator for Evaluation of Distributed System）について述べる[37].

５．１　ＳＥＤＳの必要性

先に述べたように分散処理システムを構築する上で，プロセッサを始めとする各ハードウェア要素の結合形体や分散型OSの処理方式，処理の並列化や分散法，分散された処理間での同期や通信方法など様々な問題がある．これらの問題はプロセッサの結合形体などのハードウェア的性質を含むものととその上で実現される処理方式などソフトウェア的性質のものとががからみあっており，したがってこの両面からの評価が必要である．

前章までに示した解析的手法はある特定の問題に対しては正当な論理に基づいた深い洞察を与えてくれる．また適当な近似を導入することにより、計算時間を短くすることができる．しかし、解析的に扱えるモデルにはおのずと限界があり、複雑なシステムを直接扱うことは難しい．

一方で、実際のシステムの動作を知る方法としてシミュレーションによる方法がある．シミュレーションを用いることで容易にシステムの特性を把握することができ、また近似などを導入せずに生のシステムの特性を知ることができる．一般にシステムをシミュレートする場合、なんらかの方法でシステムのモデルをシミュレータ上に展開する必要がある．これには、i)C,FORTRANなどの汎用言語でシミュレートするシステムのモデルを構築する．ii)GPSS、SIMSCRIPTなどシミュレーション専用言語を用いる．iii)そのシステムのための専用シミュレータを構築するなどの方法がある．i)は専用の言語を必要とせず、また既に開発されている汎用のパッケージを利用することができるが、汎用言語のため、プログラムの開発に時間がかかるという欠点がある．ii)は、専用の言語パッケージを必要とするが、これらの言語はあらかじめシミュレーション特有の処理を内蔵しているため、使用者が、例えば時刻管理などを気にせずプログラムを開発できる．しかし、言語ごとにある程度の対象領域を限定しており、シミュレートするモデルの構築法は固定されている

（GPSSのqueue，SIMSCRIPTのentity）ため、複雑なシステムの構築が難しい．特に、本論文で取り扱っているようなプロセスの割り当て問題の場合、扱う問題がハードウェアの構成法からソフトウェアの分散法まで多岐に渡るため、複雑なモデルと高い汎用性が要求されるが、それをシミュレーション専用言語で実現することは汎用言語で構築するのと同程度に難しい．iii)の方法は専用シミュレーション・システムを開発するために労力を必要とするが、シミュレートする問題領域に合わせた汎用性を実現することができ、また、問題領域固有の概念を用いモデルの構築が可能であるため、複雑なモデルを構築が容易である．さらに、一旦構築がなされた後は、モデルの変更が容易であり、効率のよいシミュレーションが行える．そこで我々はiii)の方法をとることにし、分散処理システム評価シミュレータＳＥＤＳを作成した．ここではその概要を述べる．

５．２　ＳＥＤＳの特徴

ＳＥＤＳは分散処理システムにおける種々の問題、すなわち分散型アルゴリズムの実現法、プロセスの割り当て方法、OSの処理方式などを評価するための高度な汎用性とユーザーインターフェースを持った専用シミュレータである．ＳＥＤＳは次のような特徴を持っている．

・高い汎用性を持つ．

ハードウェアの構成法、プロセッサの数やプロセッサ相互の結合形体、分散型OSの処理方式、分散処理システム上で実行される処理まで使用者が変更可能である．また変更を必要としない部分についてはシミュレータが提供しているものを用いることにより分散処理システムのモデルが容易に開発できる．

・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、各部の独立性を高くし、保守性を上げることができる．

・フォーム駆動型シミュレータ

一般にこの種のシミュレーションにおいて、使用者が指定しなければならない項目はきわめて多岐に渡り、複雑である．したがって、ＳＥＤＳではシミュレータへの入力指定を効率的かつ容易に行えるようにＳＥＤＳでは以下に述べるようなフォームを用いてパラメータを指定する方法を採っている．ここでフォームとはパラメータを効率よく指定するための便宜上の書式である．システムでは大きくPE form, LocalOS form, Media form, Process formの４つのフォームがあり、各フォームの指定はそれぞれ独立して行える．使用者はフォームの各項目を必要に応じて埋めて行くだけでＳＥＤＳを起動するためのパラメータの指定が完了する．このフォームの作成・編集を対話的に支援するツールとしてSEDS/10（後述）がある．

・ノンパラメトリック・シミュレータ

分散処理システムをハードウェアからソフトウェアまで種々のレベルでのシミュレーションを可能にするため、他の専用シミュレータ（PLANS,HASS）とは異なり高度の抽象化と汎用性を必要とする．つまり、分散処理システム上で行う処理を変更する場合や分散型OSの処理方式の変更などのように単なるパラメータの指定だけでは済まない要素を含んでいる．そのため、これら、分散処理システムの構造や接続形態、OSの処理方式、分散処理システム上で実現する並行プロセスの記述などを直接シミュレータの入力とする．

５．３　シミュレータの動作

ＳＥＤＳには分散処理システムおよびその上で実現される種々のソフトウェアを記述したもの、分散処理システムのシステム構成、シミュレーションの実行制御のためのパラメータなどをすべてフォームの形で入力する．ＳＥＤＳはフォームの記述に従い、対応する分散処理システムのシミュレーションプログラムを生成、フォームにあるパラメータを入力してシミュレーションを実行する．実行結果として、種々のパフォーマンス・メジャーを出力する．

ＳＥＤＳは次のようなプログラム群から構成される（図5.1）．

☆SEDS-I/O Manager　　(SEDS/10)

ＳＥＤＳの入力となるフォームを管理し、ユーザーによる対話的なフォームの作成・編集を支援する．また、シミュレーションの出力をExecutable Moduleより受け、グラフ出力などを行うプログラム群である．

☆SEDS-Parameter/Generation Manager (SEDS/PG)

I/O Managerより渡されたフォームを解釈し、各階層のプログラムを必要に応じてBasic Program Package (SEDS/PP) より選択しそれらを結合して実際のシミュレーションを行うExecutable Module (SEDS/EM) を生成する．また、パラメータを抽出し、EMの入力として与える．

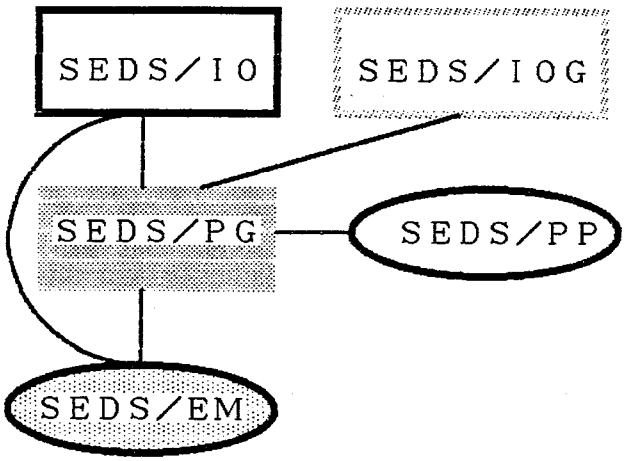


図5.1　ＳＥＤＳの構成

☆SEDS-Basic Program Package(SEDS/PP)

ＳＥＤＳの基本プログラム群を収めたライブラリー．ただし、これらのプログラム群はソースイメージで組み込まれるものとオブジェクトイメージで組み込まれるものとがある．

☆SEDS-Executable Module(SEDS/EM)

ＳＥＤＳの実行形式プログラム．EMの生成はまずＣのソースレベルで行われ、コンパイル、リンクして実行形式ができる．

５．４　分散処理システムの要素

ＳＥＤＳを駆動して分散処理システムをシミュレートするためには、シミュレートする分散処理システムをモデル化してフォームの形で記述する必要がある．以下でそのフォームの記述要素となるものについて説明する（図5.2）．

☆ハードウェア

・PE (Processing Element)

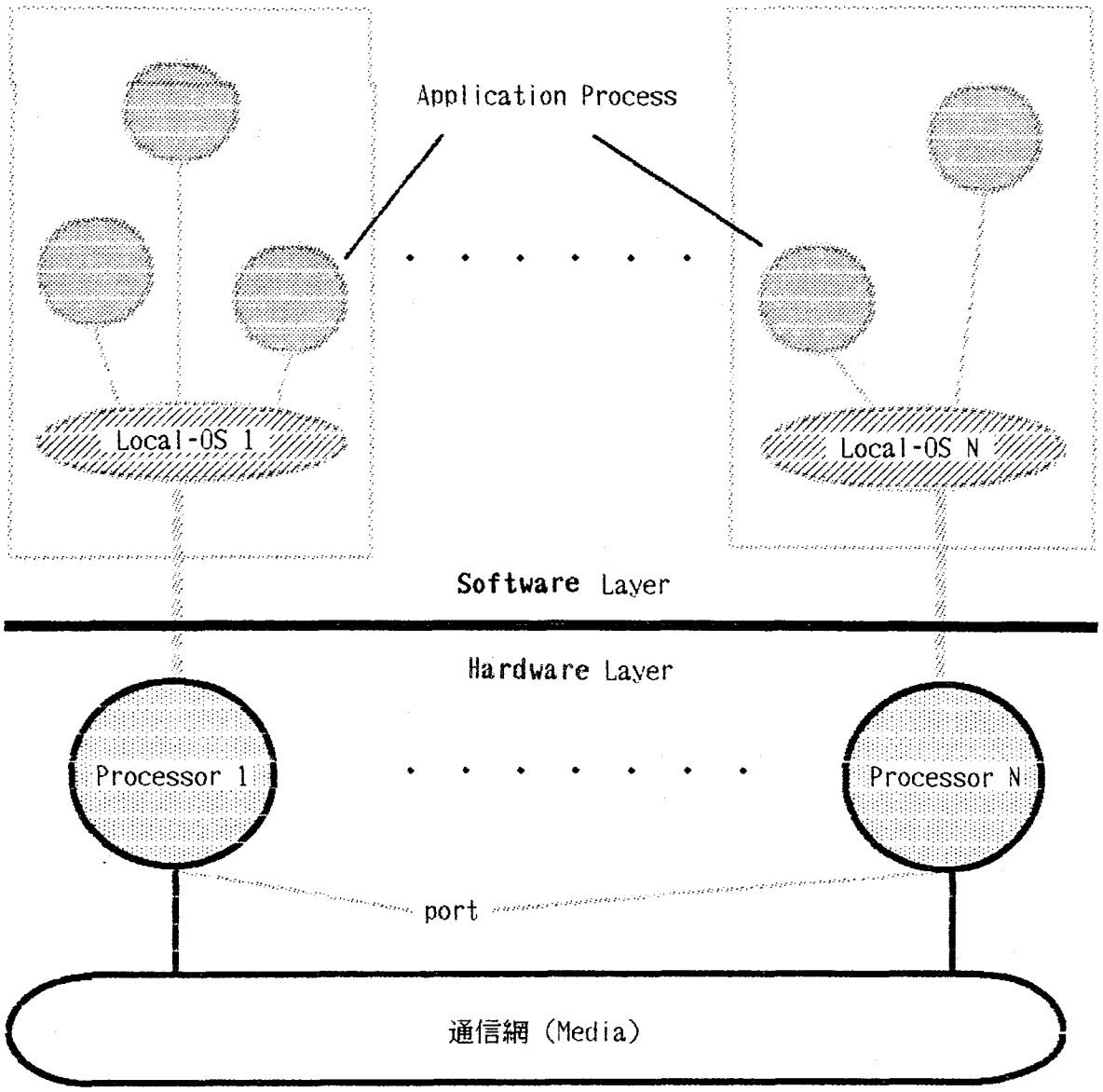


図 5.2　分散処理システムの要素

分散処理システムで実行されるユーザーの定義したソフトウェアであるアプリケーション・プロセスを実行する主体．各PEは複数のポート(port)を持ち、ポートに接続されたメディアを通して他のPEと接続される．

・Media

PE同志をポートを介して結合する通信路．ISOのOSI（Open System Interface）プロトコルのレベル２（Data Link Layer)までの機能を持つものとし、プロセス間のメッセージ転送が行われる．通信競合の管理、通信時間の確保などを行う．ただし、現在はメディアのアクセス方式は開放されておらず、システム備え付けのものがいくつか用意されており、ユーザーはこの中の一つを選ぶ．

☆ソフトウェア

・LocalOS

各PEに一つづつのLocalOSがあり、PEに割り当てられた複数プロセスの実行制御、IPCの管理などを行っている．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る．ただし、LocalOSの処理方式の実現まで開放せず、ユーザは提供されている処理方式の中から一つを選択することができる．

・Application Process

分散処理システム上で実行される処理を並行プロセス [5][6] の形で記述する．つまり、分割して各PEに割り当て可能な処理単位であるプロセスの集合とそれらの間のプロセス間通信によって処理を記述する．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．また、LocalOSとのインターフェースのために次のような関数が用意されている．

exec(job_time)

CPUを利用してjob-timeだけの処理を行う．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．

send(destination, message, type)

receive(destination, message, type)

CPUを利用してプロセス間通信を行う．

・プロセス間通信

並行プロセス間の情報の授受および同期はプロセス間通信を通して行われる．アプリケーション・プロセスはsend()関数を呼ぶことにより、LocalOSにIPC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．プロセス間通信の種類は現在のところBlocking send，Non-blocking sendの二種類が用意されている．

・Message

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アプリケーション・プロセスで内容を自由に使用できる．

５．５　実行形式モジュール（SEDS/EM）の構成

ＳＥＤＳの扱う分散処理システムにおいて、基本的に各PEは互いに独立して動作できる．また、メディアの管理も他の部分とは独立して動作できるなど高い並列性を含んでいる．したがって、ＳＥＤＳを単一のプログラムとして構成するのではなく、独立した部分を並行プロセスとして実現することにより、ＳＥＤＳ全体が捉え易く、また各部の独立性が高くなることにより開発・保守が容易になる．さらに、将来分散処理環境が構築された際にはこれらの並行プロセスを複数の計算機に分散することにより処理効率を上げ、計算時間を短縮することができる．

このような理由からＳＥＤＳはUNIX上で分散型シミュレータとして開発されており、ＳＥＤＳを構成する並行プロセスの管理およびプロセス間通信はUNIXのそれを利用している．

５．５．１　SEDS/EMの内部構成

ＳＥＤＳの実行形式であるSEDS/EMは大きく次の三つのプロセス群が階層的に構成されてできている．（図5.3）

・Simulation Management Process　　(SMP)

・PE Management Processes　　(PMP)

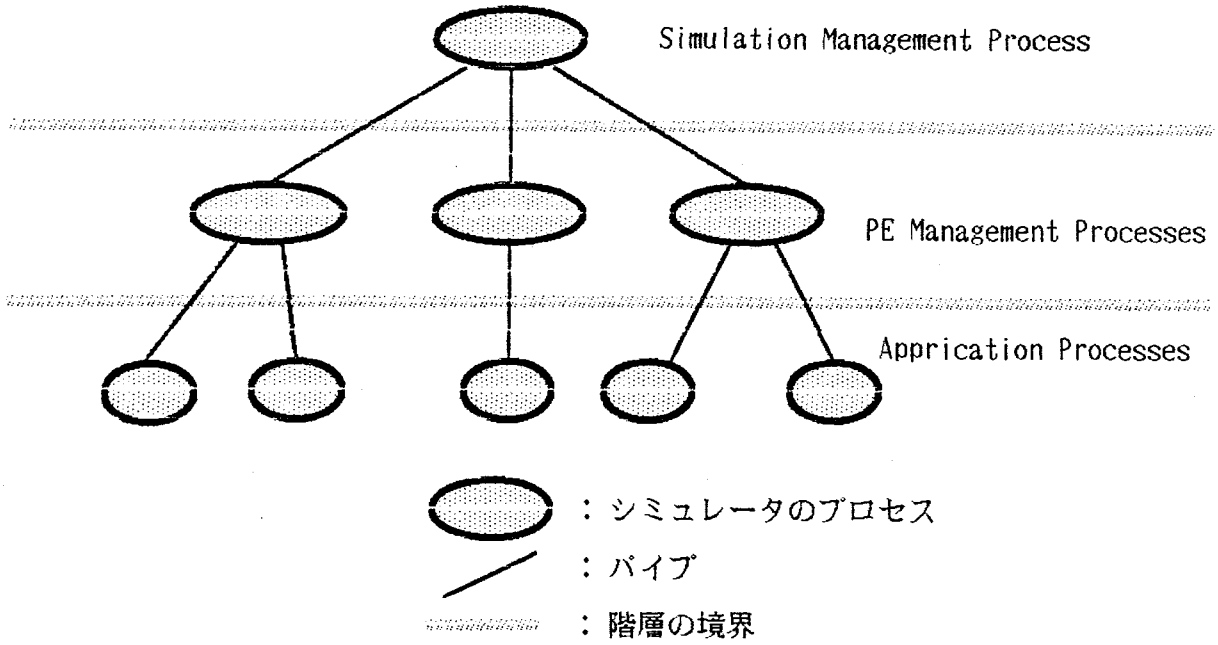


図5.3　分散処理システム評価シミュレータの構造

・Application Management Processes　　(AMP)

i)Simulation Management Process (SMP)

SMPは下位のPMPを統括し、シミュレーション全体の時刻情報管理、実行管理、統計情報収集、初期パラメータの分配などを行う．その内部にメディア管理プログラムを含んでおり、メディアに結合されているPE間でのローカル・クロックの同期をとると同時にPE間でのメッセージ転送を行う．

ii)PE Management Processes (PMP)

これは、個々のPEに一対一に対応したプロセスであり、PMPの内部にはLocalOS処理プログラムが含まれている．PEに割り当てられている複数プロセスのスケジューリングなど実行管理およびIPC管理とそれに伴うローカル・クロックの管理を行っている．

iii)Application Management Processes (AMP)

ユーザーが定義した分散処理システムにおけるプロセスはそれ自体そのままＳＥＤＳにおけるプロセスを構成する．LocalOSとのインターフェースであるsend(),recieve(),exec()は一組のPMPとSMPの間の通信となり、対応する要求をPMPに送りそれに対する返答をPMPより受け取るという形で実行される．

５．５．２　ＳＥＤＳにおけるシミュレーション時刻管理

ＳＥＤＳではシミュレータを構成する各プロセスができるだけ並列に動作できるようにするため、単一の論理時刻をもつのではなく、各PMPが別々のローカル・クロックを持って非同期に動作し必要に応じて他のプロセスとの同期をとるという形をとっている．

各PEのシミュレーション時刻はPEがプロセスの実行および同一PE内部のプロセス間通信を行っている間は非同期に進行している．PEが他のPEとプロセス間通信を行う、すなわち、あるPEに割り当てられているプロセスが別のPEに割り当てられているプロセスと通信をした時に初めてシミュレーション時刻の同期が取られる．このPE同志の時刻の同期はPMPのメディア管理プログラムで行われる．

時刻管理はメディアごとに行われる．すなわち、SMPにおけるメディア管理プログラムはあるメディアに接続されているすべてのPEから通信要求が到着するのを待つ．各PMPはアプリケーション・プロセスから外部のプロセスに対してsend()あるいはrecieve()を行ったときにこれを通信要求としてSMPにメッセージの形で伝送する．メディア管理プログラムはすべての通信要求が出揃ったら、その中の時刻の最も早いものを処理し、現在の論理時刻を決定する．SMPは現在の論理時刻をメッセージとして各PMPに送信し、再びすべての通信要求が出揃うのを待つ．各通信要求を出したプロセスはその通信要求がSMPによって満たされるまで待たされる．

５．６　ＳＥＤＳの入力フォーム

ＳＥＤＳへユーザー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．ここではこれらシミュレーションのパラメータを指定するフォームについて解説する．

５．６．１　フォームの種類

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、ユーザー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階層化して捉え、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5.1に示す．各フォーム間の階層関係を図5.4に示す．また、各フォームと分散処理システムとの関係は図5.5のようになる．

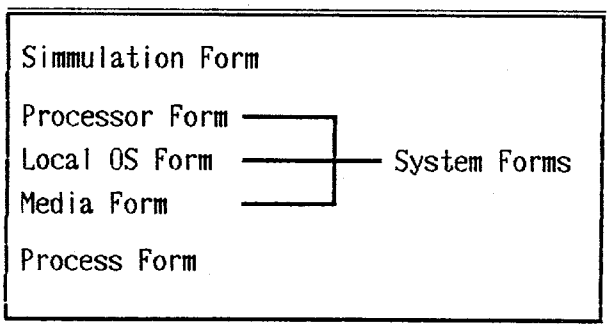


表5.1　ＳＥＤＳにおけるフォーム

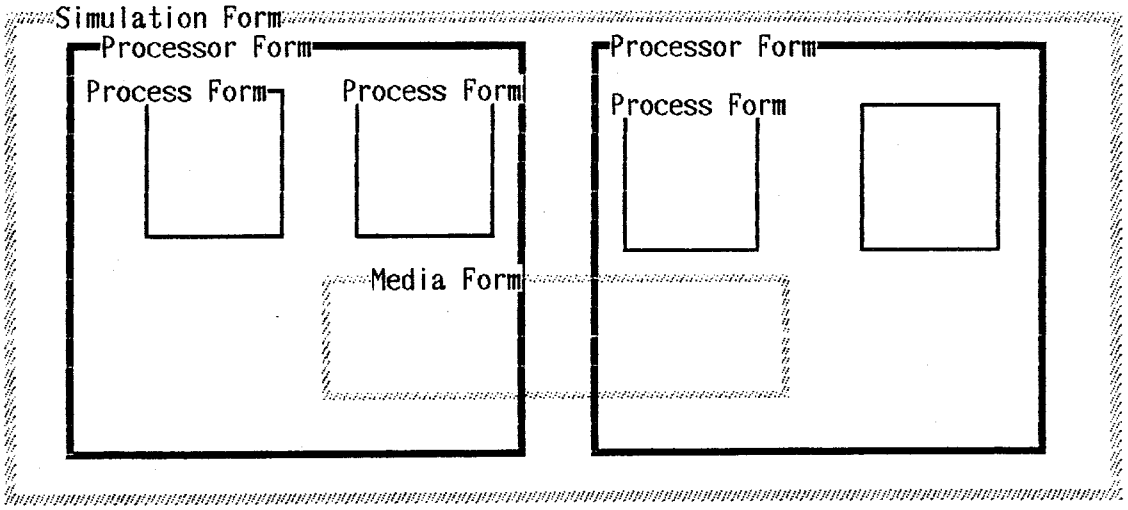


図5.4　各フォームの階層関係

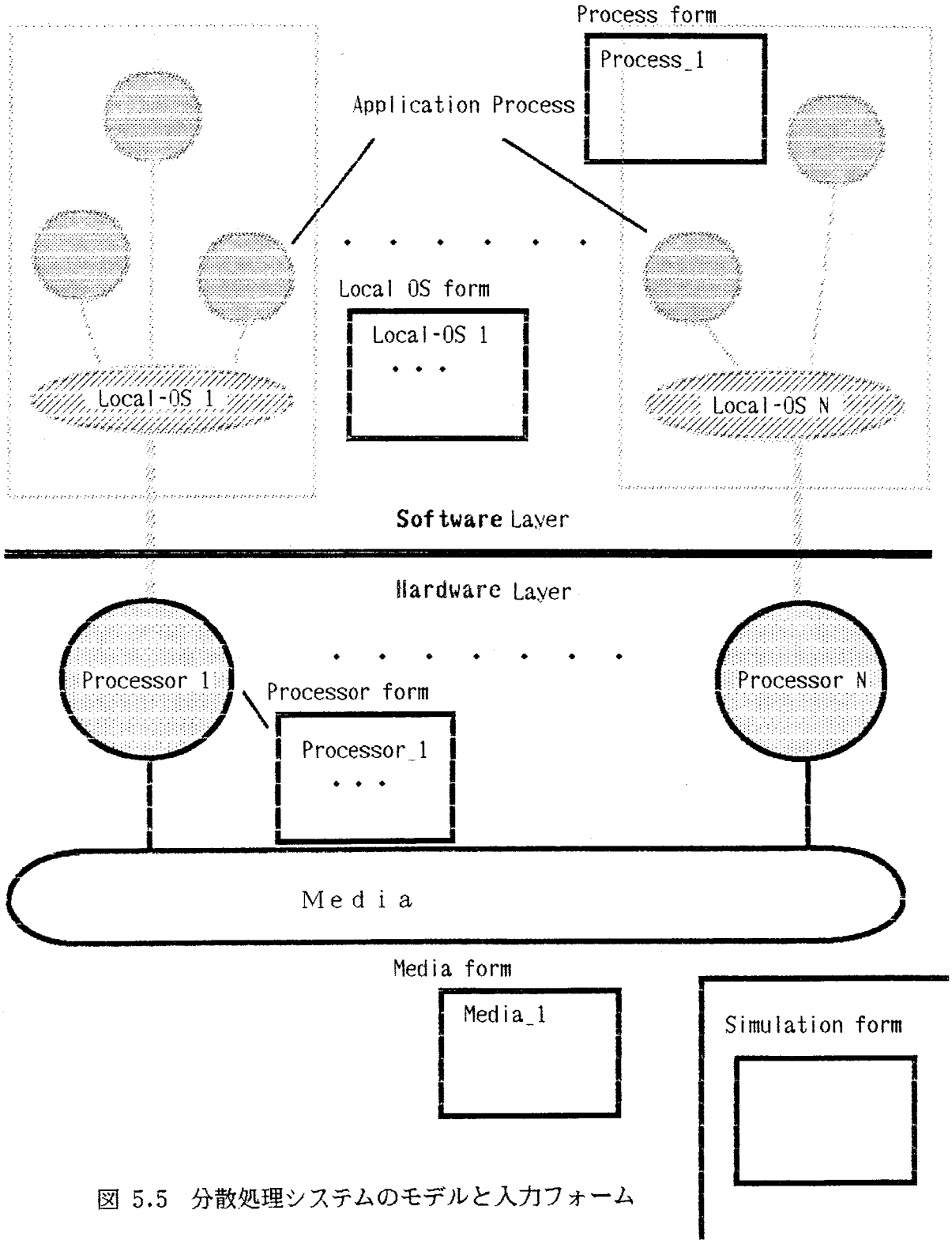


図 5.5　分散処理システムのモデルと入力フォーム

5．6．2　各フォームの形式

ここでは各フォームで指定できるパラメータおよび形式について述べる．フォームの各項目に対して必要なものだけを指定する．不必要な項目は指定を省略できる．

☆　Simulation Form

シミュレーションの実行に関するパラメータや下位のフォームが書かれているファイル名などを指定する（図5.6）．

```
Simulation Form  system_name
    System Form File: form_file_names
    Type of Simulation Run: <set/time>
      set                    time
      Total sets:            start time:
      Interval Sets:         interval time:
                             end time:
```

図5.6　Simulation Form

Simulation Formは一つの分散処理システムのシミュレーションの記述全体を統括するフォームである．一つのシミュレーションに関するフォーム群はいくつかのファイルにまたがってもかまわないが、その際関連するフォームの格納されているファイル群をSystem Form Fileの項に列挙する必要がある．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．set指定ではプロセスの起動から全てのプロセスの終了までを１セットと考え、Interval set数ごとに統計情報の平均を出力しセット数がTotal set数を越えるまでシミュレーションを行う．一方、time指定の場合、クロックがstart timeを越えるまで統計量を採らず、越えてから集計してInterval timeごとにend timeまで統計出力を出す．

☆　PE Form

PEごとに、プロセスを管理するLocalOS名、そのPEに割り当てられるプロセス名、ポートに接続するメディア名を指定する（図5.7）．プロセスは複数を指定でき、各プロセスを同様の形式で指定して行く．LocalOS,Processes,PortにおけるPara-

```
PE Form PE_name
    Local OS: localOS_name
        Parameters
            パラメータの指定
    Processes: process_name
        Parameters
            パラメータの指定
        Ports
            a: media_name
                Parameters
                パラメータの指定
            b:
```

図5.7 PE form

meterssには、LocalOS Form, Process form, Media formで設定したパラメータを与える（詳しくは後述）.

☆ Process Form

これはプロセスのパラメータおよびアルゴリズムを記述するフォームである. シミュレーション実行前に変更可能なParameters部、アルゴリズムを記述するPrograms部、統計出力時に出力するユーザー定義の変数の宣言を行うStatistics部、メッセージの内容、型をユーザーが定義するMessage部からなる. Parameters部は外部にパラメータとして開放する変数の宣言を行う. シンタックスはC言語と同じである. ここで宣言するとProcessor Formで指定可能になり、実行モジュール（SEDS/EM）を変更することなく、パラメータのみの変更が可能となる. 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. これによりユーザーが自由に所要の出力統計量の作成が可能となる. Message部は、プロセス間通信によってプロセス間で渡されるメッセージの構造を宣言する部分である. 一連のシミュレーションに対しては共通のものであるため、複数のProcess Formの内どこか一箇所で宣言しておけばよい. 分散処理システムで実行されるアルゴリズ

```
Process Form process_name
    Parameters
        解放するパラメータの宣言

    Statistics
        統計出力時に出力する変数の宣言

    Message
        ユーザーの定義するメッセージの形
                                    内容

    Program
        アルゴリズムの記述
```

図5.8 Process form

ムの記述はProgram部に置かれる．
LocalOSとインターフェースをとる特定の関数とＣ言語のシンタックスを用いて記述される．

```
Media form single_common_bus
    parameters
        type: [BLOCKED/NONBLOCKED]
        elasped time:[EXPO/NORM]
            average:
```

図5.9 Media formの例

☆　その他のフォーム

Local OS formおよびMedia Formの二つに関してはいくつかの定義済みフォームが用意されているだけでユーザーには開放されていない．ユーザーはパラメータのみをPE form内で変更できる．図5.9.5.10にそれぞれの例を示す．

```
Local OS Form Round_Robin
    slice time:
```

図5.10 LocalOS formの例

５．７　ＳＥＤＳのパフォーマンス・メジャー

ＳＥＤＳは分散処理システムおよびその上で実行される処理を記述したフォームを入力し、シミュレーションを行った結果として分散処理システムの種々の評価測度を出力する．ここではＳＥＤＳの出力する評価測度について述べる．

５．７．１　基礎となる時間量データ

ここでは評価測度を計算するもとになる種々の時間量について説明する．各時間量はある期間における累積を表しており、これと全シミュレーション時間との比をとることにより統計量の区間平均がとれる．

1) ET_i (Execution Time of process i)

　　プロセスiのCPUを使用していた全時間．

2) IPC_i (Inter Process Communication time of process i)

　　プロセスiのプロセス間通信に使用した全時間．

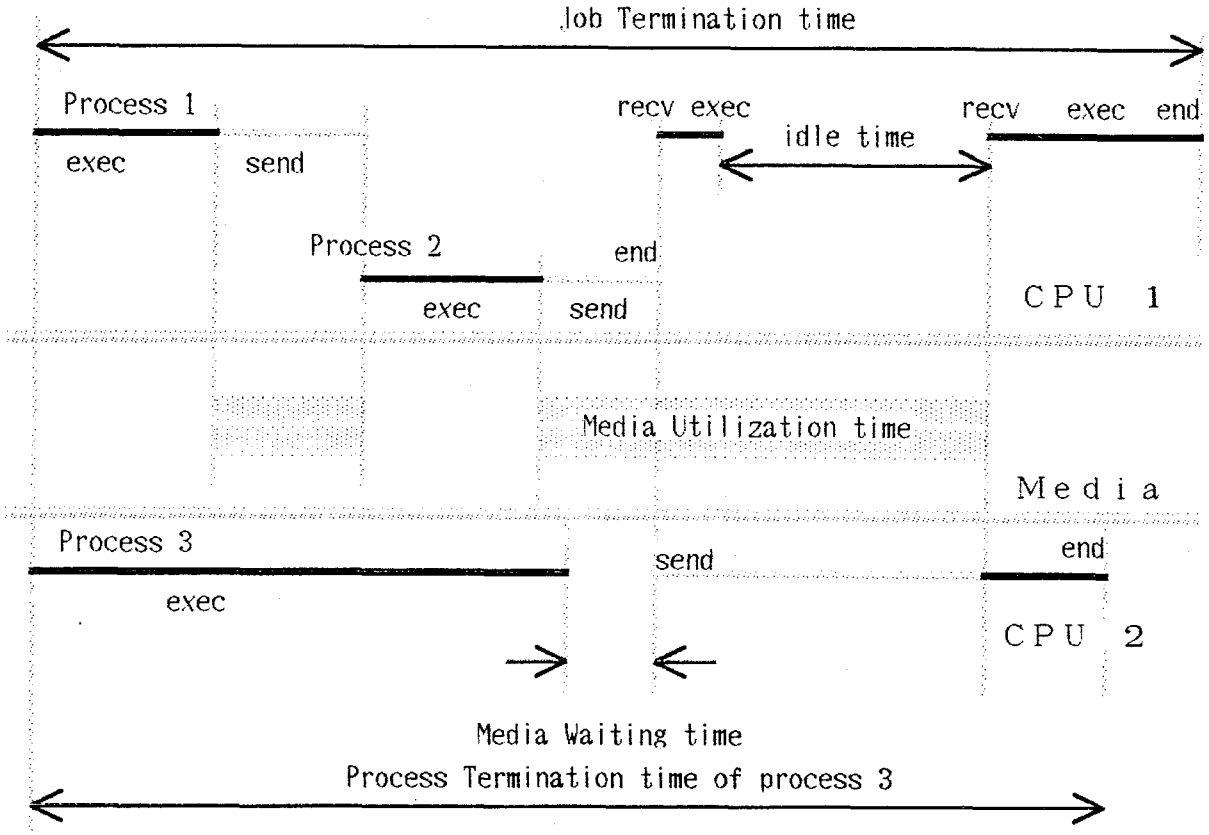


図1.1 シミュレータの出力するデータの関係

3) $PSWC_i$ (Process Switch Count of process i)

プロセスiのスイッチされた回数.

4) PT_i (Process Termination time of process i)

プロセスiに割り当てられた処理が終わるのに要した時間.

5) IDT_k (Idle Time of Local-OS process k)

Local-OS kがIdleの状態にあった時間. LocalOSはアプリケーション・プロセスの受信待ち、メディアが他のPEによって使用中のために起こる送信待ちのときに起こりうる. (図5.11参照)

6) MU_j (Media Utilization Time of media j)

media jの通信のために使用している全時間.

7) MWT_j (Waiting Time of media j)

あるプロセスがメディア j を使用しようとした時から実際にメディア使用し始めるまでの時間の総和.

1)-4)までは、LocalOS,アプリケーションの各管理プロセスが収集する. 5)については、LocalOSのプロセスのみ、6)-7)については、各メディアごとに管理プロセスが収集する. これらの関係を図で表したものが図5.11である.

5．7．2 SEDSの評価測度

5.7.1で説明したデータをもとに種々の評価測度を導出する過程について述べる. ここで説明する評価測度は、評価する対象によって大きく分けて３つに分かれる.

1) Application Processに対するもの.

1: AR_i (Active Ratio of process i)

$AR_i = (ET_i + IPC_i) / PET_k$

但し、アプリケーションプロセスi は、LocalOS kに割り当てられているとする.

ここで $PET_k = \max(PT_j)$

j:for all application process assigned to Local-OS k

2: AER_i (Active Execution Ratio of process i)

$AER_i = ET_i / PET_k$

但し、アプリケーションプロセスi は、LocalOS kに割り当てられているとする.

2) LocalOSに対するもの.

3: IR_k (Idle Ratio of Local-OS process of k)

$$IR_k = IDT_k / PET_k$$

3) システム全体に対するもの.

4: JT (Job Termination time)

$$JT = \max(PET_k)$$

k:for all Local-OS k

5: E (Effectiveness)

$$E = \frac{JT * np}{(\Sigma(ET_i + IPC_i)) * 1}$$

i:for all application process

最後のEffectivenessは、並列性を表す評価測度として捉えている. 分母は1つのPEを使用して、1つのジョブを処理するのに必要な時間であり、分子は、N_p個のPEを使用してそのジョブを処理するのに要した時間を表していると考えられる. そこで、この値が小さいほど処理の並列性が高いと考えることができる.

５．８　ＳＥＤＳの実験例（ジョブディスパッチ問題）

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・モデルを取り上げ、その適用例を示す．

５．８．１　dispatcher/executerモデル

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[38]．[38]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均等な領域のいくつかの部分画面に分割し、それらを各プロセッサに分配して実行する．この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部分画面の処理を行う．ここではこれらのプロセッサを特にこれをエグゼキュータ（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5.12）．分割された各部分画面の処理は独立して行えるためエグゼキュータ同志での通信はない．ディスパッチャーは一度に一つの部分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部分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部

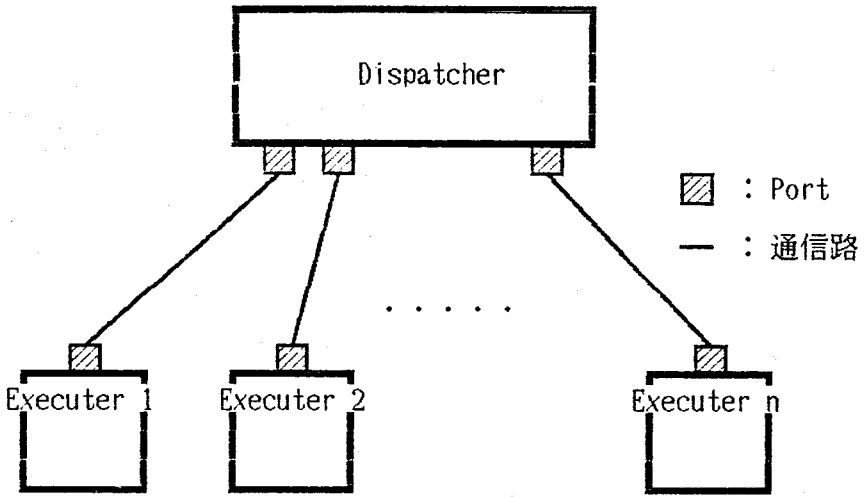


図5.12　マルチプロセッサ・システムの構成

分画面があれば、その中の一つを再びそのエグゼキュータに渡す．全ての部分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部分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．

このような処理を行うとき、一つの画面をいくつの部分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、多くの部分画面に分割すればするほど並列性は向上するが、部分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、少ない数の部分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dispatcher/executer問題（D/E問題）と呼び、ＳＥＤＳを用いて最適な分割数を求めてみる．

５．８．２　ＳＥＤＳによるD/E問題のモデル化

前節で説明したD/E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）ジョブの分割に要する時間は無視する．

２）分割された各ジョブの実行時間は平均１／μ_eの指数分布に従うとする．

３）各ジョブのディスパッチに要する時間は平均１／μ_dの指数分布に従うとする．

４）ディスパチャーとエグゼキュータの間の通信時間はディスパッチャの処理時間に含める．

表5.1　D/E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	$1/\mu_e=2.5$
ジョブ実行時間	$1/\mu_d=10000/N$
executer数	M＝4

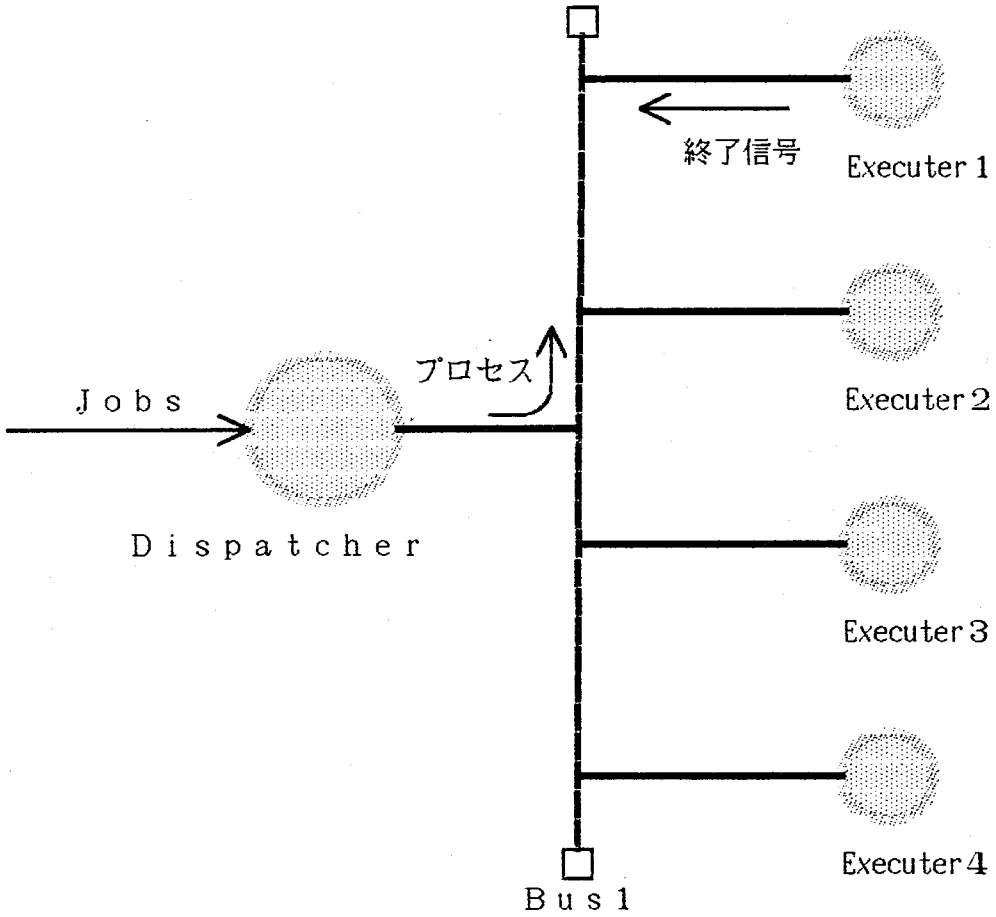


図5.13　ディスパッチャーのモデル

D/E問題のパラメータを表5.1にまとめる．ディスパッチャとエクゼキュータとの通信時間をディスパッチャーの処理時間に含めることにより、各エクゼキュータとディスパッチャの間に別々の通信路を持つのではなく、ディスパッチャと各エクゼキュータを伝搬遅延が0の単一共有バスで接続する．その結果、マルチプロセッサ・システムは図5.13の形態になる．

また、ディスパッチャーおよびエクゼキュータの処理は１つのアプリケーションプロセスとして表わし、それぞれ１つのPEに割り当てる．各PEには１つのアプリケーションプロセスが割り当てられ、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、フォームの形で表したものを付録2に示す．

５．８．３　シミュレーション結果

前節で説明したD/E問題について付録2のフォームに基づきシミュレーションを実行した．その結果を付録3に示す．

全ての処理が終了するまでの時間（ジョブ終了時間：Job Termination Time）とジョブの分割数との関係について注目し、最適の分割数がどのあたりに存在するかを考える．シミュレーション結果から得られたジョブ終了時間と分割数との関係を図5.14に示す．この図からジョブ終了時間を最小にする最適分割数はほぼ30～40に存在することがわかる．また、プロセスの分配のためのオーバーヘッド、つまりディスパッチャーの処理時間が減少すると最適分割数は大きくなる．図5.15には5.7で述べた並列度を表す尺度であるEffectivenessを示している．１に近いほど並列度が高いと考えられ、最適分割数近くで鋭いピークを持つことが分かる．

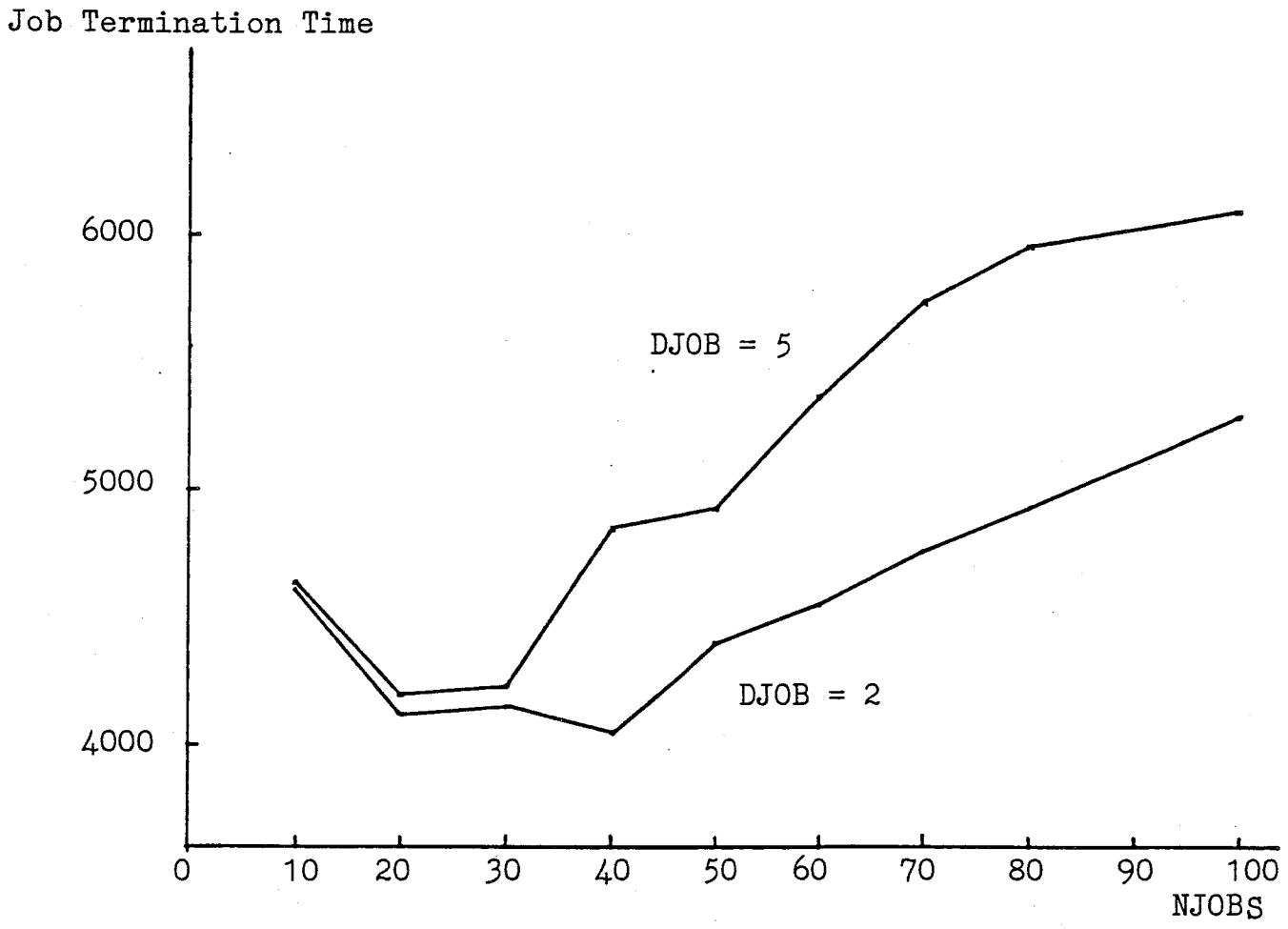


図5.14　D／E問題におけるジョブ処理時間

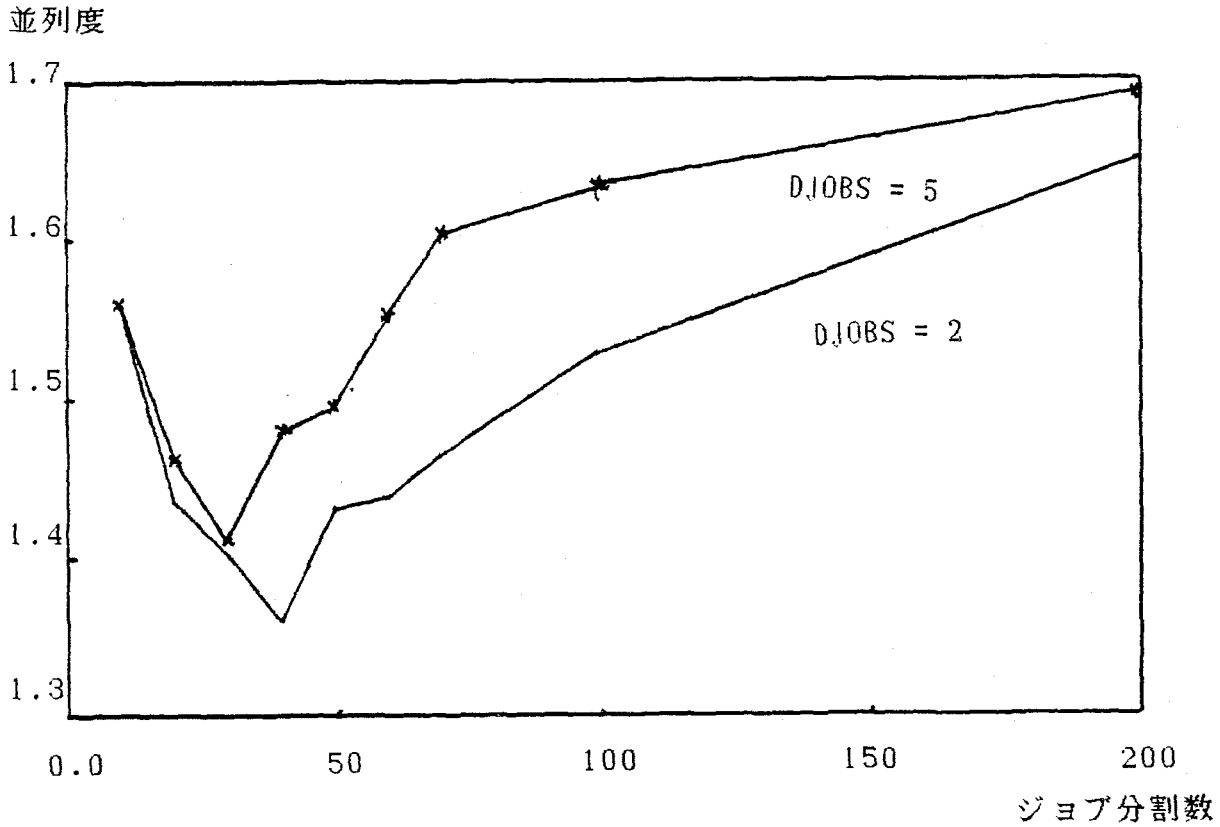


図5.15　D／E問題における並列度の変化

6. 結論

従来行われて来た分散処理システムにおける性能評価はいくつかの問題点を含んでいる．分散型アルゴリズムの評価はハードウェア要素を全くあるいはほとんど考慮せずに（無限台数のPE、完全結合、IPC競合の無視等）行われている．また、分散処理システムにおけるPE同志を結合する相互通信網のスループット・遅延などが評価されているが、これらはその上で実行されるソフトウェアの影響を考慮していない．つまり、これら分散処理システムにおける性能評価は個々のソフトウェア（アルゴリズム）やハードウェア（相互通信網など）単独の性能に対する洞察を与えてくれるが、それらを組み合わせた相互効果についてはあまり研究されていない．

本論文では分散処理システムにおいてハードウェア・ソフトウェアの両面を統合的に評価する手法としてプロセス間通信による通信競合を含めた分散処理システムの性能評価手法を提唱した．分散処理システム上で実行されるジョブの構成と分散処理システム形態とが互いにシステムの性能に与える効果をプロセス間通信による通信競合という形で取り込み、分散処理システムを待ち行列網理論を用いてモデル化することにより、平均ジョブ実行時間を求めた．これによりハードウェアである分散処理システム形態とソフトウェアである分散型アルゴリズムがシステムに与える影響を考慮に入れた総合的な性能評価を行うことができる．次に、分散処理システムにおける通信競合を含めたプロセス割り当て法すなわち、平均ジョブ実行時間を最小にする割り当てを求める方法を示した．

対象とする分散処理システムとしては、複数のPEが共通のCNに結合されている一般的なモデルを考えたが、割り当てられるプロセスの処理形態について二種類を考えた．すなわち、プロセス実行とIPCの同時処理を行わない（プロセスの実行においてあるプロセスがIPCを行っているときには同一PE内の他のプロセスの実行は行わない）モデルとプロセス実行とIPCの同時処理を行うモデルとである．さらに後者のモデルでは一つのジョブに属するプロセス間の同期を考慮に入れている．これ

ら二つのモデルを扱うことにより広範囲な分散処理システムを解析的に評価することが可能となった．

通常プロセス割り当て問題は整数計画問題として定式化され、本論文でも一般のプロセスに対する割り当て問題については、任意のプロセス割り当てに対する平均ジョブ実行時間を待ち行列網理論を用いて解析し、これを目的関数とする整数計画問題を構成している．しかし、整数計画問題はＮＰ完全な問題として知られており、また本論文で取り扱っている問題は平均ジョブ実行時間を最小化しようとするため、目的関数が複雑な非線型となり、さらに問題を難しくしている．このため、なんらかの発見的手法を導入して近似解を求めることが主眼となる．

本論文では均質なプロセスに対するプロセス割り当て問題に関しては近似を導入することにより、平均ジョブ実行時間を最小にするPE数を解析的に求めた．これにより、均質プロセスの割り当て問題をすべての割り当てを探索することなく容易に解くことができた．また、一般のプロセス割り当て問題に対しては、各PEにおけるプロセスの実行時間の総和を等しくする（ロードバランス）とともに割り当てられたプロセスによる各PE間の通信量がほぼ等しくなるような割り当てがよいという発見的手法を導入し、これに基づく近似解法を提唱した．特に４章では分枝限定法に上記の発見的手法を導入することにより、解の探索時間を大幅に改善することができた．また、数値計算の結果、厳密解と比較してもよい解が得られることが示された．

従来、分散処理システムにおいては、プロセッサの台数を増やしてもプロセッサ間の通信量の増加により、効率が上がらないことが簡単な解析に基づき指摘されて来た．本論文で提唱した解析手法により数値計算を行った結果、このことが具体的に平均ジョブ実行時間を評価測度として明らかになった．つまり、プロセスを割り当てるPE数が増加すると平均ジョブ実行時間は大幅に減少するが、最適PE数を越えると、それ以降平均ジョブ実行時間は徐々に増加するということが明らかになった．

また、分散処理システムにおいて効率が最大となる点を最適プロセッサ数あるいは最適割り当てとして近似的に求めることができた

上記のような解析的手法による分散処理システムの性能評価は厳密な論理に基づいた明確な関係が得られるという意味で非常に重要である．しかし、一方で解析的な手法で扱えるシステムには限界があり、複雑なシステムになると解くことが困難になる．そこで、分散処理システムにおいてソフトウェア・ハードウェアを統合的に評価できるようなシミュレータＳＥＤＳを開発した．ＳＥＤＳを用いて分散処理システムの評価を行うことにより解析的に解くことが困難である複雑な分散処理システムに対する性能評価を容易に行うことができた．

分散処理システムにおいて性能評価に用いられる手法は様々であり、ソフトウェア面での評価はグラフ理論、組み合わせ算法など、ハードウェア面では待ち行列網理論やマルコフ過程などが用いられる．したがって、分散処理システムをこれら両面を統合して評価するためには相異なる手法を効果的に結びつける必要があり、本論文では待ち行列網理論と組み合わせ手法を融合したアプローチをとった．待ち行列網理論を用いて分散処理システムを確率モデル化することによりシステムをマクロ的に捉えることができ、大きく複雑な分散処理システムに対する性能評価を容易に行い得ることを明らかにした．

本論文で扱っているプロセスはすべて静的なものとしている．つまり、ジョブを構成するプロセスは割り当て時に既に存在しており、ジョブの実行中に生成したり、消滅したりすることはないとしている．したがって、この場合、一旦プロセスを割り当てると実行中に変えることはない．しかし、並列言語の中には、アクターやSmalltalkなど動的にプロセスを生成・消滅させる機能を具備したものがあり、これらで記述されたジョブに対してはジョブの実行時に動的にプロセス割り当てを行う必要がある．つまり、プロセスが実行時に生成・消滅するため実行前に割り当てただけでは、プロセスを多数抱えているPEと少ししか抱えていないPEができ、その

間で動的に負荷分散をする必要が出て来る．これは動的プロセス・スケジュール問題として知られており、５．８で扱ったジョブ・ディスパッチ問題などもこれに属する．これについていくつかの方式が提案されている[39],[40]．しかし、ここでもハードウェアの評価しかなされておらず、ソフトウェアであるジョブのモデルはあまり明確ではない．したがって、本論文で述べたようなハードウェア／ソフトウェア両面からの統合的な評価が望まれる

謝辞

本研究をまとめるにあたり、本研究の機会を与えて頂き、直接ご指導頂いた大阪大学基礎工学部高島堅助教授に心から深謝致します。

学部、大学院を通じて御指導御教授頂いた大阪大学基礎工学部の藤沢俊男教授、都倉信樹教授、嵩忠雄教授、鳥居宏次教授、並びに大阪大学産業科学研究所の豊田順一教授に心から感謝致します。

また、研究の細部に渡り日頃御指導頂いた高島研究室の宮原秀夫助教授、西田竹志助手に心から感謝致します。

さらに、本研究を進めるに当たって御討論、御助言頂いた京都大学工学部数理工学科の長谷川利治教授、室章治郎助手、高橋豊助手、カリフォルニア大学アーバイン分校の須田達也助手に心からお礼申し上げます。

最後になりましたが、著者の在学中、御討論頂いた高島研究室の方々に心から感謝致します。

【参考文献】

1. Zary Segall, Ajay Singh, Richard T. Snodgrass, Anita K. Jones, and Daniel P. Siewiorek, "An Integrated Instrumentaion Environment for Multiprocessors," IEEE Trans. on Comput., vol. C-32, no. 1, pp. 4-13, JAN 1983.

2. Fromm Hansjorg, Uwe Hercksen, Ulrich Herzog, Karl-Heinz John, Rainer Klar, and Wolfgang Lleinoder, "Experiences with Performance Measurement and Modeling of a Processor Array," IEEE Trans. on Comput., vol. C-32, no. 1, pp. 15-31, JAN 1983.

3. Leonard S. Haynes, Richard L. Lau, Daniel P. Siewiorek, and David W. Mizell, "A Survey of Highly Parallel Computing," Computer magazine, vol. 15, no. 1, pp. 9-24, JAN 1982.

4. Hideyuki Tokuda, Eric G. Manning, "An Interprocess Communication Model for a Distributed Software Testbed," ACM, pp. 205-212, 1983.

5. R. C. Holt, Concurrent Euclid, The Unix System, and Tunis, Addison-Wesley Publishing Co., Mass., 1983.

6. 坂村、"アーキテクチャにおけるユニークさとは何か - オッカムとトランスピュータ -".bit. Vol. 16. No. 3. pp. 40-50. (1984.3).

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

8. Carl Hewitt, "Viewing Control Structures as Patterns of Passing Messages," Artificial Intelligence, no. 8, pp. 323-364, 1977.

9. Laxmi N. Bhuyan and Dharma P. Agrawal, "Performance Analysis of FFT Algorithms on Multiprocessor Systems," IEEE Trans on Soft. Eng., vol. SE-9, no. 4, pp. 512-521, JUN 1983.

10. Christopher P. Arnold, Michael I. Parr, and Michael B. Dewe, "An Efficient Parallel Algorithm for the Solution of Large Sparse Linear Matrix Equations," IEEE Trans. on Comput., vol. C-32, no. 3, pp. 265-273, MAR 1983.

11. Carla Savage and Joseph Ja'Ja, "Fast, Efficient Parallel Algorithms for Some Graph Problems," SIAM J. Comput., vol. 10, no. 4, pp. 682-691, Nov 1981.

12. C. D. Thompson and H. T. Kung, "Sorting on a Mesh-Connected Parallel Computer," C. ACM, vol. 20, no. 4,

pp. 263-271, APR 1977.

13. Allan Gottieb and J. T. Schwartz, "Networks and Algorithms for Very-Large-Scale Parallel Computation," Computer magazine, vol. 15, no. 1, pp. 27-36, JAN 1982.

14. Jiro Tanaka, Robert M. Keller and Frank C.H. Lin, "Rediflow Multiprocessing," Compcon '84 spring, pp. 410-417, FEB 1984.

15. Carl Hewitt and Henry Lieberman, "Design Issues in Parallel Architectures for Artificial Intelligence," Proc. on Compcon'84 spring, pp. 418-423, FEB 1984.

16. Flavia Rosemberg, "An Ada Oriented Protocol for Intertask Communication In Real Time Common Bus System," Proc. of Inter. Seminar of Comput. Network and Perf. Eval., pp. 11-4, SEP 1985.

17. K. M. Chandy and J. Misra, "Distributed Computation on Graphs: Shortest Path Algorithms," C.ACM, vol. 25, no. 11, pp. 833-837, NOV 1982.

18. John T. Robinson, "Some Analysis Techniques for Asynchronous Multiprocessor Algorithms," IEEE Trans. on Soft. Eng., vol. SE-5, no. 1, pp. 24-31, JAN 1979.

19. Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, and Francesco Gregoretti, "Modeling Bus Contention and Memory Interference in a Multiprocessor System," IEEE Trans. on Comput., vol. C-32, no. 1, pp. 60-71, JAN 1983.

20. Marco Ajmone Marsan, Gianfranco Balbo, and Gianni Conte, "Comparative Performance Analysis of Single Bus Multiprocessor Architectures," IEEE Trans. on Comput., vol. C-31, no. 12, pp. 1179-1191, DEC 1982.

21. Daniel A. Reed and Herbert D. Schwetman, "Cost-Performance Bounds for Multimicrocomputer Networks," IEEE Trans. on Comput., vol. C-32, no. 1, pp. 83-95, JAN 1983.

22. Janak H. Patel, "Performance of Processor-Memory Interconnections for Multiprocessors," IEEE Trans. on Comput., vol. C-30, no. 10, pp. 771-780, OCT 1980.

23. Wesley W. Chu, Leslie J. Holloway, Min-Tsung Lan, and Kemal Efe, "Task Allocation in Distributed Data Processing," IEEE Computer magazine, vol. 13, no. 11, pp. 57-69, NOV 1980.

24. 有吉、笠原、成田、"マルチプロセッサスケジューリング問題に対する最

適及び近似アルゴリズム(1),(2)",情処全大(昭和59年度 前期), No. 50-6, pp. 11-12, (1984.3).

25. C. V. Ramamoorthy, K. M. Chandy, and Mario J. Gonzalez, JR., "Optimal Scheduling Strategies in a Multiprocessor System," IEEE Trans. on Comput., vol. C-21, no. 2, pp. 137-146, FEB 1972.

26. M. A. Salichs, "Task Assignment Across Space and Time in a Distributed Computer System," Proc. on IFAC Distributed Computer Control Systems, pp. 131-141, 1983.

27. Chien-Chung Shen and Wen-Hsiang Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," IEEE Trans. on Comput., vol. C-34, no. 3, pp. 197-203, MAR 1983.

28. Jerome M. Kurtzberg, "On the Memory Conflict Problem in Multiprocessor Systems," IEEE Trans. on Comput., vol. C-23, no. 3, pp. 286-293, MAR 1974.

29. 茨城, "組合わせ最適化",産業図書, (1983).

30. K. Mani Chandy and Charles H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computing Systems," Computing Surveys, vol. 10, no. 3, pp. 281-317, SEP 1978.

31. Leonard Kleinrock, Queueing Systems Vol. 2, John Wiley & Sons, NY, 1976.

32. 下条真司, 西田竹志, 宮原秀夫, 高島堅助, "通信競合を含めたプロセス割り当て問題とその近似解法",信学技報 電子計算機研究会, Vol. EC84, No.47, pp. 25-35 (1984,12).

33. 下条, 宮原, 高島, "通信競合を含めたマルチプロセッサにおけるプロセス割り当て問題", 信学会 論文誌(D), Vol. J68-D, No. 5, pp. 1049-1056, (1985,5).

34. Kensuke Takashima, "Process Assignment on Distributed System with Communication Contentions," Proc. of Inter. Seminar of Comput. Network and Perf. Eval., pp. 11-5, SEP 1985.

35. 下条真司, 宮原秀夫, 高島堅助, "分散処理システムにおけるプロセスの割り当て法の比較 検討",信学技報 電子計算機研究会, Vol. EC85, No. 16, pp. 21-28,(1985.8).

36. Philip Heidelberger and Kishor S. Trivedi, "Analytic Queueing Models for Programs with Internal Concurrency," IEEE Trans. on Comput., vol. C-32, no. 1, pp. 73-82, JAN 1983.

37. 下条、坪郷、宮原、高島、"分散処理システム評価シミュレータの設計、開発"、
信学全大、No. 1715、pp. 7-21、(1985.3).

38. 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).

39. Andre M. Van Tilborg and Larry D. Wittie, "Wave Scheduling --Decentralized Scheduling of Task Forces in Multicomputers," IEEE Trans. on Comput., vol. C-33, no. 9, pp. 835-843, SEP 1984.

40. Daniel A. Reed, "The Performance of Multimicrocomputer Networks Supporting Dynamic Workloads," IEEE Trans. on Comput., vol. C-33, no. 11, pp. 1045-1048, NOV 1984.

【付録】

付録１：　平均外部通信時間の導出
付録２：　フォームによるＤ／Ｅ問題の記述

付録１：平均外部プロセス間通信時間Rc(Λ) の導出

あるプロセッサでは一時に高々一個のプロセスしか実行せず、したがってプロセッサにおける外部プロセス間通信要求も高々一つしか起こらないから、この各プロセッサにおける外部プロセス間通信要求はPE全体を無限サーバーと考えたときの客とみなすことができる．PE j からCNへのアクセス率は$\lambda^{PE}j$で与えられ、PE jにおける外部プロセス通信要求の発生間隔は平均$1/\lambda_j$の指数分布に従うとみなせる．このため無限サーバーにおける各客の処理時間は異なり、PE j における外部プロセス間通信要求をクラス j の客、その無限サーバーにおける処理時間を平均$1/\lambda_j$とする．またCNは全てのクラスで処理時間の平均が$1/\mu_c$の単一指数サーバーである．したがってマルチプロセッサ・システムは二つのサーバーを持つ待ち行列網となる．これはBCMPモデルになり、CNをノード0、PE全体をノード1とし各ノードの状態をその中の客数n_0 ($0 \leq n_0 \leq n_p$)、$\mathbf{n}_1$(n_1, n_2, . . . n_{n_p}) (n_j=0/1 (j=1..n_p)) で表わすとする．システム全体の状態をS＝(n_0, n_1) とすると、Sの定常分布確率p(S) はn_0、n_1別々に解いた時の定常分布確率の積で与えられる[31]．したがって、p(S) は次式で与えられる．

$$p(S) = G^{-1} f_0(n_0)\, f_1(n_1) \qquad (A.1)$$

ここでf_0、f_1はそれぞれ、

$$f_0(n_0) = (1/\mu_c)^{n_0}\, n_0! \qquad (A.2)$$

$$f_1(\mathbf{n}_1) = \prod_{r=1}^{n_p} (1/\lambda^{PE}{}_r)^{n_r} \qquad (A.3)$$

となるから、結局p(S) は、以下のようになる．

$$p(S) = G^{-1} (1/\mu_c)^{n_0} \prod_{r=1}^{n_p} (1/\lambda^{PE}{}_r)^{n_r} \qquad (A.4)$$

ただし、Gは正規化定数で、

$$G = \sum_{\text{for all } S} (1/\mu_c)^{n_0} \prod_{r=1}^{n_p} (1/\lambda^{PE_r})^{n_r} \qquad (A.5)$$

ここで、Sは

$$S = (n_0, n_1, \ldots n_{np})$$

$$1 \leqq n_0 \leqq n_p, \quad n_j = 0,1 \quad 1 \leqq j \leqq n_p \qquad (A.6)$$

を満たす.

ここで、ノード0のutilization factor U_0は

$$U_0 = p\,[n_0 \geqq 1] = 1 - G^{-1} \prod_{r=1}^{n_p} (1/\lambda^{PE_r})^{n_r} \qquad (A.7)$$

これからスループットλ_0は

$$\lambda_0 = U_0 \mu_c \qquad (A.8)$$

一方、平均システム在中数E［n_0］は

$$E\,[n_0] = \sum_{j=1}^{n_p} j\,p\,[n_0 = j] \qquad (A.9)$$

リトルの公式からCNでの平均システム在中時間Rc(Λ)は

$$Rc(\Lambda) = E\,[n_1] / \lambda_0 \qquad (A.10)$$

で与えられる.

付録２　フォームによる記述
５．７で説明したD/E問題のモデルをフォームを使用して記述する．

Simulatioan Form

```
/*  ジョブディスパッチ・モデルのシミュレーション  */
Simulation Form  JOB DISPATCHER
        Type of Simulation Run: set
                        Total Set: 40
                        Interval Sets: 40
```

PE Form

```
PE Form DISPATCHER      /*  ディスパッチャーの記述  */
        Local OS: Round_Robin
                parameters
                        slice_time=9999         /*  シングルタスクのため、
                                                    単位処理時間は無限大  */
        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  */
        Ports
                a: BUS1                         /*  メディア名  */

PE Form PE1
        Local OS: Round_Robin
                parameters
                        slice_time=9999
        Processes: executer
                parameters
                        NODENUM=1       /*  エグゼキュータ番号  */
                Ports
                        a: BUS1 /*  Media BUS1に関するパラメータ  */
                                parameters
                                        type=BLOCKED
                                        elasped_time=0
```

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．

Process Form

```
Process Form DISPATCHER          /* ディスパッチャー・プロセスの記述 */
        Parameters
           double TJOB, DJOB, NJOBS;  /* 開放するパラメータの宣言 */
        Message
                int     no;     /* パケットを送ったエクゼキュータの番号 */
                double  average;  /* 割り当てるジョブの処理時間の平均 */
        Program
dispatcher()
{   PROCESS pe1, pe2, pe3, pe4;
    MESSAGE mes;
    double job_time, exp_rand();
    int loop;

start;  /* アルゴリズムの記述 */

/* メッセージの送り先の情報の取り出し */
pe1 = find( "PE1" );    pe2 = find( "PE2" );
pe3 = find( "PE3" );    pe4 = find( "PE4" );

/* 1つめのジョブの割り付け */
mes->average = TJOB / NJOBS;
send( pe1, mes, BLOCKED);       send( pe2, mes, BLOCKED);
send( pe3, mes, BLOCKED);       send( pe4, mes, BLOCKED);

for(loop = 4; loop < NJOBS - 4 ; loop++){

        /* 処理の終了のパケットを各エクゼキュータから受ける. */
        recieve( ANY, mes, BLOCKED);

        /* ディスパッチの処理 */
        job_time = exp_rand( DJOB );
        exec( job_time );

        /* 次のジョブの割り付け */
        mes->average = TJOB / NJOBS;
        switch mes->no{
         case 1: send( pe1, mes, BLOCKED);
                 break;
         case 2: send(pe2,mes,BLOCKED);
                 break;
         case 3: send(pe3,mes,BLOCKED);
                 break;
         case 4: send(pe4,mes,BLOCKED);
                 break;
```

```
        }
}  /* for statement end */
/* 各エクゼキュータから最後の終了のパケットを受ける．  */
for(loop = 0; loop < 4 ;loop++) recieve( ANY, mes, BLOCKED);

/* ディスパッチの終了を各エクゼキュータに伝える．  */
mess->no = 999;          /* mess->noが999が最後を表わす．  */
send( pe1, mes, BLOCKED);        send( pe2, mes, BLOCKED);
send( pe3, mes, BLOCKED);        send( pe4, mes, BLOCKED);

endp;   /* 記述の終了 */
}


Process Form executer
        parameters
        int NODENUM;
        Program
executer(){
    PROCESS dispatcher;
    MESSAGE mes;
    double job_time, exp_rand();

    start;      /* アルゴリズムの記述 */

    /* メッセージの送り先の情報の引き出し */
    dispatcher = find( "DISPATCHER" );

    while( 1 ){
        /* ディスパッチャから割り当てられたジョブの情報を受ける．*/
        recieve( dispatcher, mes, BLOCKED);

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

        /* 割り当てられたジョブの処理 */
        job_time = exp_rand( mes->average );
        exec( job_time );

        /* 処理終了のパケットをディスパッチャに送る．  */
        mes->no = 1   /* 1 is process number.  */ （注）
        send( dispatcher, mes, BLOCKED);
    }  /*  while end  */
```

```
    endp;        /* 記述の終了 */
}
```

注）エクゼキュータ２～４の記述はこの文を２～４の代入文に変えるだけのものであるため省略する．

Media Form

```
Media Form BUS1          /*  単一競合バス  */
        parameters
                int type;        /* [BLOCKED/NBLOCKED]  */
                double elasped_time;    /*  通信時間  */
```

Local OS Form

```
Local OS: Round_Robin   /*  ラウンド・ロビン  */
        parameters
                int slice_time; /*  処理単位  */
```

注）ここで示した Media Form および Local OS Form は各Processor Form に共通