分散環境で動作する分散アルゴリズムシミュレータ

弘田暢幸[†] 梅本秀樹[†] 相原玲二[††]
山下雅史[†††] 阿江忠[†††]

分散アルゴリズムの振舞いは複雑でその正当性を示し複雑度を解析することは容易ではない．そこで分散アルゴリズムの動作を確認し，その平均的特性を調べるために分散アルゴリズムシミュレータを作成したので報告する．本シミュレータは大規模な分散システムのシミュレーションを効率良く行うことを目的として作成されており，イーサネットによって結合されている複数のワークステーション上にシミュレータを分散させてシミュレーションを行う．本シミュレータはシミュレータの使用を容易にし，シミュレートされているアルゴリズムの実行状況の可視化を助けるためのユーザインタフェースを持っている．

A Distributed Simulator for Distributed Algorithms

Nobuyuki Hirota,[†] Hideki Umemoto,[†] Reiji Aibara,[††]
Masafumi Yamashita[†††] and Tadashi Ae[†††]

Distributed algorithms are algorithms for processes participating in a distributed system, to control their behaviors so that they, in total, work as a consistent system. Although there have been proposed several formal methods for showing properties, e.g., correctness, termination, deadlock-freedom, desired for distributed algorithms, applying any of them to practical systems is by no means easy, and hence, simulation seems to be a practical and promising way for convincing their correctness. An issue we should overcome to design a practically usable simulator for distributed algorithms is that simulating a distributed system on a sole workstation has an apparent limit on the number n of processes in the simulated system, which is usually very low, since intuitively we need to simulate the behaviors of n workstations by one. In this paper, we design a simulator which can simulate large scale distributed systems by distributing the simulator to workstations connected with Ethernet. The simulator has a user interface which makes the use of simulator easy and helps to visualize behaviors of distributed algorithms.

1. はじめに

計算機と通信技術の進歩に伴い局所ネットワークや広域ネットワークが広く利用されるようになり，分散システムの普及が進んでいる．分散システムは通信リンクを介して互いに通信し，自律的に動作するプロセスの集合としてモデル化される．分散システムに属するプロセスは共有変数を持たず，通信リンクを介してメッセージを受信することにより互いに情報を交換する．このような分散システム上で与えられた問題を解くアルゴリズムを分散アルゴリズムという（分散システムおよび分散アルゴリズムについてのより詳細な説明は例えば文献4),7)を参照せよ）．

分散アルゴリズムは分散システムを構成するプロセス上で並行的に実行されることに加え，分散システムは通常非同期であるため，設計した分散アルゴリズムが分散システム上で正しく動作することを厳密に証明したり，その複雑度を解析的に調べることは容易ではない．そこで分散アルゴリズムシミュレータを用いてその動作を確認したり，その平均的特性を調べることが重要な検討手段となる．

このような観点から永松は *DAS* (Distributed Algorithm Simulator) と呼ばれる分散アルゴリズムシミュレータを設計し[8]，負荷分散アルゴリズムの評価を始め，いくつかの分散アルゴリズムの評価に用いてきた[5]．また，都倉は特にアルゴリズムの可視化を含むユーザインタフェースに焦点を当てて分散アルゴリ

† 広島大学大学院工学研究科情報工学専攻
Information Engineering Course, Graduate School, Hiroshima University
†† 広島大学集積化システム研究センター
Hiroshima University Research Center for Integrated Systems
††† 広島大学工学部第二類
Department of Electrical Engineering, Hiroshima University

ズムシミュレータを構成している[11]．しかし，これらのシミュレータはいずれも単体の計算機上で実行されるため，実行速度および利用できるメモリの制約から，シミュレートできる分散システムの大きさに制約が課せられていた．

そこで，イーサネットで結合された複数のワークステーションから構成される LAN 上に *DAS* を分散し，LAN 上で分散的に分散システムのシミュレーションを行う分散アルゴリズムシミュレータ D^2AS (Distributed Distributed Algorithm Simulator) を作成したので報告する．D^2AS の第一の目的はシミュレータを分散することにより大規模な分散システムのシミュレーションを効率良く行うことである．またユーザインタフェースを完備することによりユーザにかかる負担を軽減し，グラフィカルで対話的な環境で容易にシミュレーションが行えることを目指した．

2. 分散システムのモデル

D^2AS がシミュレートするのは，与えられた分散システム上で動作する与えられた分散アルリズムの動作である．分散システムとその上での分散アルゴリズムの振舞いをまず定義する（本章では，その概略を説明するに止める．厳密な定義に興味のある読者は文献4),7)を参照せよ)．

2.1 分散システム

分散システムは自律的に動作するプロセスの集合 V とプロセス間通信のための単方向の通信リンクの集合 E から構成される．分散システムのトポロジとして(有向グラフ $G=(V,E)$ として表現される）任意の有向グラフが許される．分散システムに属するプロセスは共有変数を持たず，プロセス間通信のみによって互いの情報を交換する．各プロセスは固有の識別子を持っている．各プロセスは，また，局所時計を持っているが，後で述べるように局所時計の正確さは分散システムの非同期さの度合により異なる．

プロセス間通信は点対点通信であり，プロセス P_1 から P_2 に向かう単方向通信リンク L が存在するときに限り P_1 は P_2 に直接メッセージを送信できる．より明確には，P_1 は L にメッセージを送出するための出力ポートを持ち，P_2 は L からのメッセージを受信のための入力ポートを持っている．受信ポートには FIFO キューが置かれており，P_1 からのメッセージはこのキューに格納される．基本通信命令は Send と ReceiveOR 命令である．Send が実行されると，指定された1つの出力ポートからメッセージを送信し，ただちにその実行を終了する．送信されたメッセージはある通信遅延を伴って受信側のキューに格納された後，受信可能になる．送信されたメッセージには，送信プロセスの識別子と送信時刻(局所時計)が時刻印として自動的に付加され，分散アルゴリズムはこれを利用できる．メッセージの受信には ReceiveOR 命令が用いられる．ReceiveOR 命令はプロセスに属する入力ポートの中から任意のポート群を受信の対象として指定できる．そして，指定されたポート群の受信キューの中で待っている受信可能なメッセージのうち一番小さな時刻印を持つメッセージを受信し，実行を終了する．もし指定されたポート群の中に受信可能なメッセージがない場合には，メッセージが到着するまでこの命令の実行は凍結される．ReceiveOR 命令による凍結を避けるために，受信キューにメッセージが入っているかどうかを確かめる Empty Queue 命令を利用できる．

つぎの3条件を満たす分散システムを同期システムと呼ぶ．(1)各プロセスの実行速度は等しい．(2)通信遅延はある定数である．(3)各プロセスの局所時計は同じ時刻を指している．同期システムでない分散システムは非同期システムである．したがって，さまざまな程度の非同期性がありうる．

2.2 分散アルゴリズム

分散アルゴリズムは，各プロセス P_i が実行するアルゴリズム A_i の集合 A であり，言語 C++ を用いて記述する（言語 C++ については文献9) を参照)．ただし C++ は，2.1節で定義した基本通信命令 Send と ReceiveOR を利用できるように拡張されている．

アルゴリズム A_i は初期情報として，以下の情報を利用できる．

- 分散システムに属するプロセスの総数
- 分散システムに属する通信リンクの総数
- P_i の識別子
- P_i に属する入出力ポート数
- P_i の隣接プロセスの識別子
- P_i の局所時計が指す時刻

分散システムにおける分散アルゴリズムの実行は，そのシステムに属するあるプロセス P_i が自発的にアルゴリズム A_i の実行を開始することで，開始される．その他のプロセス P_j は自発的にアルゴリズム A_j の実行を始めるか，あるいはすでに実行を開始してい

るプロセスからメッセージを受信することによって実行を開始する．分散アルゴリズムはすべてのプロセスがアルゴリズムの実行を終了したとき，終了する．

3. D^2AS の構成

分散アルゴリズム A と分散システム S が与えられると，D^2AS は A の S 上での動作をシミュレートする．このシミュレーションは，イーサネットで結合された複数のワークステーション上で分散的に行う．すなわち，S をいくつかの部分システム $S_1, \cdots, S_m$ に分割し，それぞれを別々のワークステーション $W_1, \cdots, W_m$ 上でシミュレートする．異なる部分システム S_i, S_j 間の通信は，W_i, W_j 上のシミュレータ間のイーサネットを介した通信によってシミュレートする．

D^2AS を用いて分散アルゴリズムのシミュレーションを行う際に，ユーザが行わなければならない仕事は以下の3つである．

(1) C++を用いた分散アルゴリズム A の記述．

(2) A が実行される分散システム S の記述．
有向グラフとして与える分散システムのトポロジの記述と，分散システムが非同期であるか同期であるかの選択，さらに命令実行時間，通信遅延，受信キューの長さなどのパラメータの設定に分類される．

(3) シミュレータ制御情報の記述．
シミュレータが分散されるワークステーション名 $W_1, \cdots, W_m$，最大シミュレーション時間，シミュレーションの中間/最終結果を採集するための周期などの設定．

D^2AS はマスタとスレーブから構成されている（**図1**参照）．マスタは起動されると上に述べた入力を記述した設定ファイルを読み込み，それに従ってスレーブをワークステーション $W_1, \cdots, W_m$ 上に起動し，分散システム S を $S_1, \cdots, S_m$ に分割して，$W_1, \cdots, W_m$ に割り当てる．S の $S_1, \cdots, S_m$ への分割方法はシミュレーション時間に大きく影響すると考えられ，本論文の考察事項の1つである（第8章参照）．以下では，ワークステーション W_i 上のスレーブを W_i と同一視し，スレーブ W_i と呼ぶ．

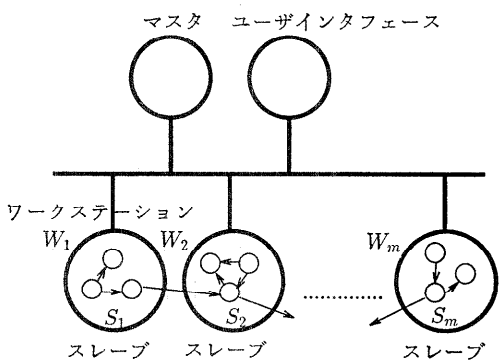


図1 D^2AS の構成
Fig. 1 Components of D^2AS.

スレーブ W_i は，コルーチンを用いて担当する部分システム S_i 上の分散アルゴリズム A の動作をシミュレートする．異なるスレーブが担当する部分システムの通信は，TCP/IP プロトコルのストリームソケットを利用して行う．同期システムをシミュレートする場合では，スレーブ間で同期をとる必要があるが，このための同期メッセージの送信はマスタの仕事である．

設定された周期に従って，各スレーブは分散アルゴリズム A の実行の中間情報，および最終結果を収集し，それをマスタに送信する．

4. シミュレーション方法

4.1 命令実行時間の設定

D^2AS は基本的には論理時間シミュレーションを行う．このため，1単位時間を決定し，各命令の実行時間を定める必要がある．これは Tick 命令によってつぎのように行う．ユーザは分散アルゴリズムの記述の中に Tick 命令を挿入しておく．同期システムをシミュレートする場合，D^2AS は2つの連続する Tick 命令にはさまれたアルゴリズム部分が1単位時間で実行されるものと考える．例えば，プロセス内の通信命令以外の局所計算時間を無視したい場合，Tick 命令は通信命令の直後だけに挿入すればよい．逆にプロセス内の局所計算時間が無視できない場合，Tick 命令を各命令の後に挿入する．

非同期システムでは命令の実行速度が動的に変化する可能性がある．これをシミュレートするために，乱数を用いて2つの Tick 間のアルゴリズムの部分の実行にかかる単位時間数を動的に変化させる☆（4.2節，非同期システムのシミュレーションの項を参照）．

4.2 シミュレーション

マスタの指示にしたがって，各スレーブ W_i は担当する部分システム S_i における（4.1 節で述べた）1単位時間分のシミュレーションを開始する．各スレーブは，このシミュレーションが終了すると，その

☆ 繁雑さを避けるために本論では言及しないが，D^2AS では，実行時間や通信遅延をいくつかの分布に従って発生させることもできる．

ことをマスタに報告する．この1単位時間分のシミュレーションをフェーズと呼ぶことにする．マスタはすべてのスレーブからの終了報告を確認した後，つぎのフェーズの開始を各スレーブに通知する．このように各スレーブがフェーズを同期的に繰り返すことで，シミュレーションは進行する．

D^2AS は同期および非同期システムをつぎのようにシミュレートする．

- 同期システム
  - (1) 各スレーブは担当する部分システムに属するすべてのプロセスを1単位時間分ずつシミュレートする．
  - (2) 送信されたメッセージはユーザが定めたある定数単位時間の後，受信側のキューに置かれ，受信可能な状態になる．
  - (3) 局所時計はフェーズ番号（実行中のものを含み，今までに実行されたフェーズ数）を示す．
- 非同期システム
  - (1) プロセスの実行速度の動的な変化は，あるフェーズでプロセスを実行した後，その実行を続く数フェーズの間凍結することによって実現する．凍結する時間は乱数によって決定する．
  - (2) 通信遅延の動的な変化は，ユーザが設定する通信遅延値に乱数を加えた値を通信遅延とすることにより実現する．FIFO 型システムをシミュレートする場合には，メッセージ順序が変化しないという条件の下で，通信遅延を変化させる．
  - (3) 多くの分散アルゴリズムでは，分散システムが論理時計[6]をサポートしていると仮定しているので，論理時計を局所時計として用いる．

4.3 高速シミュレーション

4.2 節で述べたシミュレーションは，フェーズごとにスレーブ間で同期をとるため，スレーブ数やそれらが実行されているワークステーションの負荷がシミュレーション結果に影響を及ぼさないという利点はある．しかし，実行速度が最も遅いスレーブに合わせてシミュレーションが進行することになり，また，スレーブ間同期のためのオーバヘッドが問題となる．そこで，D^2AS では非同期システムをシミュレートする場合にはつぎのようなスレーブ間同期を省略する方法を選択でき，これを高速シミュレーションと呼んでいる．

マスタの指示に従って，各スレーブは担当する部分システムのシミュレーションを開始する．このシミュレーションは，4.2 節の場合と同様に行う．唯一の相違は，1つのフェーズが終了したときにスレーブの同期を省略して，つぎのフェーズに進むことである．その結果，フェーズの進行速度は各スレーブでまちまちである．

上で述べたように，部分システム S_i のシミュレーションにおいて，メッセージ M の送信にかかる遅延が d であると（乱数を用いて）決定されたとする．この送信が S_i 内のプロセス間での通信ならば，スレーブ W_i が d フェーズだけシミュレーションを進めた後，M は対応するキューに格納される．一方，この通信が S_i のあるプロセスから S_j のあるプロセスへの送信ならば，スレーブ W_i は d フェーズだけシミュレーションを進めた後，M を W_j に送信する．M を受信した W_j はただちに対応するキューに M を格納する．W_i と W_j のシミュレーションがほぼ並行に進んでいるときには，M はおおよそ d フェーズ遅れて対応するキューに格納されるが，d と M を送信した W_i のフェーズ番号と M を受信した W_j のフェーズ番号の差には，関連が一般にはない．

高速シミュレーションでは，シミュレーション結果がシミュレーションが進行している時点の LAN の状況に依存するので，高速シミュレーションを繰り返すことによって，ある固定された分散システム S 上でのアルゴリズム A の動作の統計的性質を導くことはできない．しかし，高速シミュレーションは S 上の A の動作の例を正しくシミュレートしているので☆，分散システム上でのアルゴリズムの動作の観察という目的にはかなっている．

5. シミュレーション結果の出力

分散アルゴリズムを評価する場合，一般的に評価基準として考えられるものに，メッセージ複雑度と理想時間複雑度がある☆☆．メッセージ複雑度はアルゴリズムの実行が終了するまでにプロセス間で交換されるメッセージの総数である．理想時間複雑度は局所計算時

☆ このシミュレーションが，時間的前後関係 (happend-before relation)[5]に矛盾しないように進められているという事実からわかる．

☆☆ 時間計算量と呼ばれる尺度もある[7]．D^2AS では時間計算量も測定できるが，ユーザの工夫が多少必要である．

間を無視し，アルゴリズムが完全同期システム上で実行されているとみなしたときに，アルゴリズムが停止するまでに必要な論理時間数，すなわち，ラウンド数である．D^2AS では，メッセージ数とラウンド数以外にもシミュレーションから得られるさまざまな情報を得ることができる．

D^2AS にはシミュレーションの中間または終了時に各プロセスの持つ情報を採集する機能があり，この機能をプローブ (probe) と呼んでいる．プローブはプロセスの状態に重要な変化が起きたとき，つまり，受信命令の実行が凍結され待ち状態になった場合，待ち状態から抜け出した場合，そしてプロセスが終了した場合に自動的に実行される．また，ユーザが記述するアルゴリズムの中に Probe 命令を挿入することによって Probe 命令が実行される直前の状況を採集できる．

プローブが採集する情報には，プロセスの識別子，局所時計，プロセスの状態，ユーザが指定した変数の値が含まれる．プロセスの状態は，実行中，メッセージ待ち状態，実行終了状態のいずれかである．プローブ情報の対象となる変数は，C++ の関数として記述されたアルゴリズムにおいて，一番外側のブロック（'{'と'}'で囲まれた部分）で宣言された変数群である．

プローブ機能によりアルゴリズムの中間的なシミュレーション結果を得ることができ，ユーザインタフェースを介してアルゴリズムの動作状況の確認を行うことができる．

大規模なシミュレーションを行う際，すべてのプロセスでプローブを実行すると，通信にかかる負荷のためにシミュレーションの効率が悪くなる．そこでそれぞれのプロセスについてプローブを実行するかしないかをユーザが指示することにしている．また，プローブの実行を開始する時刻と，実行する周期をユーザが指定することで，重要なプロセスの情報だけを収集し，無駄を省くことができる．ただし，シミュレーションの終了時には無条件にすべてのプロセスでプローブが実行される．

6. デッドロックの検出

分散アルゴリズムの正当性の証明は困難であり，したがって，シミュレートしている分散アルゴリズムがデッドロックすることも十分予想される．このような状況を放置すれば，実質的にシミュレーションは進行せず，資源の浪費となる．そこで，この章では，D^2AS のデッドロック検出機能について説明する．デッドロックとは，あるプロセスが決して満たされることのない要求を待ち続ける現象を指す．ここでの要求はメッセージの受信である．

要求待ちモデルはいくつか提案されているが，D^2AS では ReceiveOR を基本受信命令としているので OR モデル[1]を用いる．プロセスの待ち状態が待機グラフ WFG (Wait-For Graph)[1]で表されているとする．WFG はプロセスを表す点とプロセス間の待ちの依存関係を表す有向辺から構成されるグラフで，あるプロセス v がプロセス w からのメッセージを待っているとき，かつそのときに限り v から w に有向辺が張られる．有向グラフの強連結部分グラフ K でかつ，K に属するすべての頂点について，それらから到達可能なすべての頂点がまた K に属するような性質を持つものを結び (knot) と呼ぶ．OR モデルでは WFG 上に結びが存在するとき，かつそのときに限りデッドロックが存在する．デッドロック状態に対応する WFG の例を**図 2** に示す．

デッドロックの検出は以下のように進められる．各スレーブ W_i は，部分システム S_i に関する待ち状況を（局所的）WFG として維持しており，あるプロセス v がプロセス w からのメッセージをある一定時間以上待っているとき，v から w に辺が張られる．v が w からのメッセージを受信すればただちにその辺は除去される．

スレーブ W_i は新しく辺が張られるたびに，S_i に局所的に発見できるドッドロックが起こっていないか（すなわち，この局所的 WFG に結びが存在するかどうか）調べる．局所 WFG に結びがある場合には，デッドロックが検出されたことになり，そのことをマスタに報告し，マスタはユーザインタフェースを介してユーザにそのことを報告してシミュレーションを中断し，ユーザに指示を仰ぐ．

結びの候補に含まれるあるプロセスが，他の部分システムに属するプロセスを待っている場合，すなわ

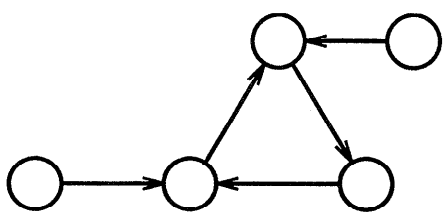


図 2 WFG 上のデッドロックの例
Fig. 2 A WFG showing a deadlock.

ち，全システム S の大域的 WFG を解析することが必要な場合には，局所 WFG をマスタに送信し，マスタによるデッドロック検出アルゴリズムの起動を促す．マスタは，スレーブから送られてくる局所 WFG を基に大域的 WFG を構成し，この大域的 WFG に結びがあるかどうかをチェックし，結びがあるようならばデッドロックが検出されたことになる．

このデッドロック検出を正しく行うためには，各スレーブから集めた局所的 WFG が無矛盾である必要がある．無矛盾性の厳密な定義は文献2)に譲るが，直観的には，集められた局所的 WFG が同時に存在しうるようなものであることを意味する．このためにいわゆる，大域スナップショットをとる[2]．

D²AS では，以下のような方法で大域スナップショットをとる．まず，マスタはすべてのスレーブの実行を一時停止させる．つぎに，マスタに各スレーブの局所的 WFG と通信リンクの状態を集める．マスタは，すべての情報が集まったことを確認した後，シミュレーションの再開を各スレーブに指示する．マスタは集められた情報から大域的 WFG を構成する．このようにして構成された WFG に結びがあれば，分散システムはデッドロックしており，誤って錯覚デッドロックを検出してしまうことはない．なぜなら，マスタは各スレーブの実行中断を確認してから大域スナップショットを要請し，大域スナップショットの終了を確認してから，シミュレーションの再開を指示しているので，大域スナップショットに対応するカットは無矛盾であり，したがって，(シミュレートされているシステムが非同期であるか同期であるかにかかわらず) 得られる大域スナップショットは無矛盾となるからである．

7. ユーザインタフェース

D²AS を用いてシミュレーションを行うために，ユーザは通常以下のことをしなければならない．

(1) シミュレートする分散アルゴリズムを C++ で記述してコンパイルし，*D²AS* のスレーブのライブラリとリンクしてスレーブプログラムを作成する．

(2) 対象となる分散システムの定義（ネットワークの形状や通信リンクの通信遅延，受信キューのサイズなど）を記述した設定ファイルを作成する．

(3) シミュレーションを行う環境（スレーブ数やそれを実行するワークステーションの指定など）を記述した設定ファイルを作成する．

D²AS は(1)のために，アルゴリズムを記述した C++ プログラムファイルと完成したスレーブのプログラム名を指定すれば自動的にスレーブプログラムを作成するユーティリティを用意している．

(2)と(3)の設定ファイルの作成は，あらかじめ決められた書式で記述しなければならない．この設定ファイルは分散システムの性格やシミュレーション条件を変更して *D²AS* を繰り返し実行する場合，毎回書き換えなければならないので，設定ファイルの修正や作成の際にユーザにかかる負担は大きい．そこでスレーブの作成，設定ファイルの編集，シミュレーションの実行を統一した環境で行えるユーザインタフェースを作成した．このユーザインタフェースは X-Window 上で動作し，グラフィカルで対話的な環境をユーザに提供する．数値の入力を除けばほとんどの操作がマウスでボタンやメニューを選択することで実行できる．

ユーザインタフェースはスレーブの作成，分散システムの定義および *D²AS* の動作環境の設定，シミュレーションの実行を行う部分から構成されている．

7.1 スレーブの作成

アルゴリズムを記述するファイルとスレーブプログラム名を指定してウィンドウ上のテキストエディタを用いてシミュレートするアルゴリズムを記述しコンパ

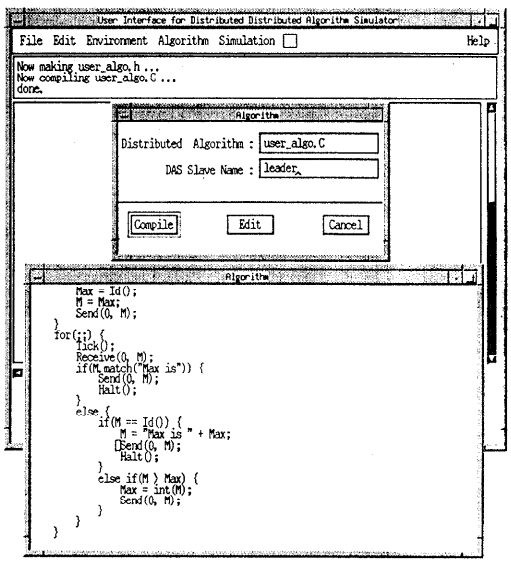


図 3 スレーブの作成
Fig. 3 Making slaves.

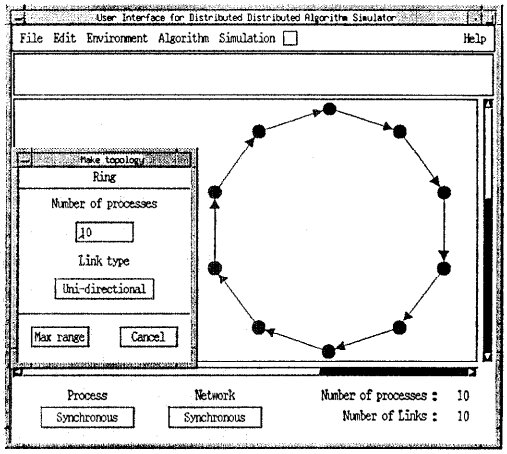


図 4 分散システムの記述
Fig. 4 Specifying a distributed system.

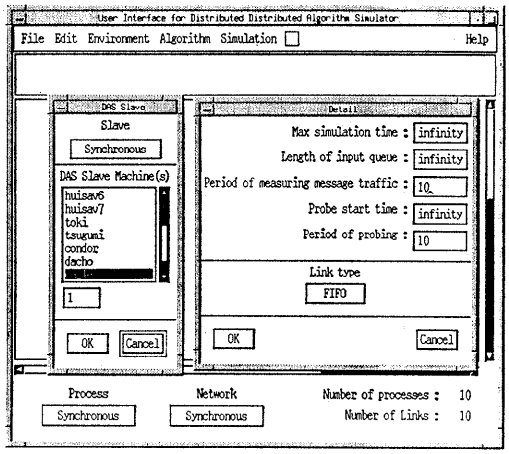


図 5 動作環境の設定
Fig. 5 Setting the environment of *D²AS*.

イルする．コンパイル時のエラーメッセージはメッセージウィンドウに表示される（**図 3** 参照）．

7.2 分散システムの記述と動作環境の設定

分散システムのトポロジを記述するためのグラフィックエディタが用意されており，ユーザはこれを用いてプロセスと通信リンクを描いて分散システムを構成する．また画面上のプロセスや通信リンクを指定してプローブを集めるプロセスや通信遅延などを設定する．このグラフィックエディタでは特定の形状（リング，完全結合，メッシュ，線状，星状）はプロセス数と通信リンクの単方向，双方向を指定することにより自動的に作成できる（**図 4** 参照）．双方向の通信リンクは内部では 2 本の単方向リンクとして記憶される．

分散システムのトポロジ記述以外にシミュレーションの条件として分散システムの性格を定めるパラメータ群（同期/非同期の設定，受信キュー長等々），*D²AS* の動作環境であるマスタ/スレーブを実行するワークステーション名，最大シミュレーション時間，メッセージ数を調べる周期などを設定する（**図 5** 参照）．

7.3 *D²AS* の実行

シミュレーションを行う場合，アルゴリズムの動作状況を確かめながら実行したい場合と結果だけが得られれば良い場合がある．本ユーザインタフェースでは両方の場合に対応する実行をサポートする．

前者の場合，シミュレーションを開始するとユーザインタフェースは *D²AS* のマスタと通信してプローブ情報や通信リンク上のメッセージ数を設定ファイルで指定した時間間隔で表示する．また，*D²AS* の実行を一時停止したりステップ実行することも可能である．シミュレーションが終了すると全プロセスの最終的な状態と，送受信されたメッセージ総数が出力され，これらの結果はファイルに保存される（**図 6** 参照）．

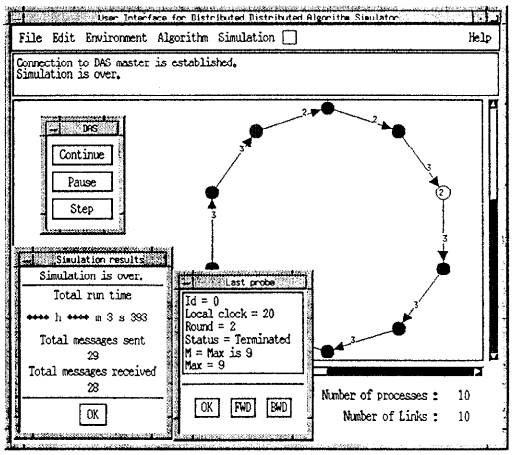


図 6 実行画面
Fig. 6 Executing *D²AS*.

後者は大規模なシミュレーションを行うときに使われる．この場合 *D²AS* はバックグラウンドで実行され結果はファイルに保存される．

8. シミュレーションの分散化の効果

8.1 実験方法と結果

シミュレーションを分散して行うことによる効果を確かめるために実験を行った．利用するワークステー

ション数を変化させて，非同期単方向リング上での最大値発見問題に対する，Chang と Roberts のアルゴリズム[3]のシミュレーションを行い，それらの実行にかかる時間を計測した．Chang と Roberts のアルゴリズムのメッセージ複雑度は，最悪の場合 $n(n+1)/2+n$（ただし，n はプロセス数）である．今回の実験では，メッセージ数が常に最悪になる場合について実験を行った．

シミュレートする分散システムは非同期とし，(1)スレーブ間で同期をとる場合と，(2)同期をとらない場合，つまり高速シミュレーションの場合について調べた．また，異なるスレーブに属するプロセス間の通信がシミュレーション時間に与える影響を調べるため，上記の(1)，(2)のそれぞれに対して，(a)隣合うプロセスが，できるだけ同じスレーブに属するようなリングの分割と，(b)隣合うプロセスが，できるだけ異なるスレーブに属するようなリングの分割に対する実行時間を調べた．

実験はシミュレートする分散システムのプロセス数を256とし，スレーブを1～6台のワークステーションに分散させてそれぞれ10回ずつ D^2AS を実行し，シミュレーションの実行時間の平均をとった．使用したワークステーションは，データゼネラル社の AV 300 (DG/UX 4.32) である．結果を**図7**に示す．

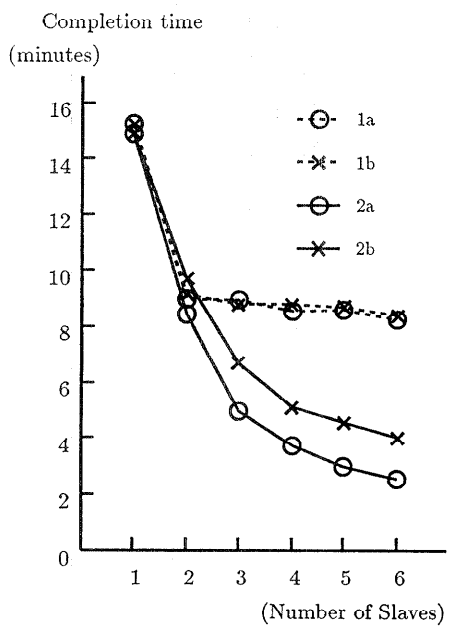


図7 スレーブ数と D^2AS の実行時間の関係
Fig. 7 The effect of slave distribution.

8.2 考　察

スレーブ間で同期をとった場合（図7，1a と 1b 参照），スレーブ数を3以上にすると，ほとんど分散の効果が現れない．1フェーズの実行時間はスレーブ数が n のときに $1/n$ になると期待されるが，このとき，同期のための通信時間は n 倍になると予想され，AV 300 間での通信遅延が2～3ミリ秒であるから，実行時間の改善が通信遅延に埋没したと考えるのが普通であろう．しかし，この説明では，スレーブ数が増加したときに，シミュレーション時間が逆に増えるはずであり，結果と矛盾する．そこで，同期の結果，フェーズの進行速度はもっとも遅れているフェーズに合わされるので，フェーズの実行時間における分散の効果や，通信遅延における分散の代償は，一番遅いフェーズを待ち合わす時間に埋没してしまった，と考えるのが適切であろう．グラフ 1a と 1b に差が出ないことも，この説を支持している．グラフ 1a と 1b の差は AV 300 内通信と AV 300 間通信の差があり，本来，グラフ 2a と 2b の差程度が予想される．しかし，同期のための AV 300 間の通信遅延が，遅いスレーブの待ち合わせ時間に埋没したものと考えられる．

一方，高速シミュレーションの場合（図7，2a と 2b 参照）には，分散するスレーブの数が増えるに従って実行時間は短縮され，分散した効果が順調に現れている．特に，隣合うプロセスができるだけ同じスレーブに属するような配置の場合（グラフ 2a），実行時間は分散させたスレーブの数にほぼ反比例し，並列化がほぼ完全に行われたことをうかがわせる．事実，この場合，各スレーブは各フェーズに最大1個のメッセージを他のスレーブに送信するだけであり，それ以外の処理はすべてスレーブ内で済まされるので，並列化の効果がそのまま現れる．隣合うプロセスをできるだけ異なるスレーブに配置した場合（グラフ 2b）は，隣合うプロセスができるだけ同じプロセスに配置された場合（グラフ 2a）と比較すると，常にほぼ一定の割合で多くの実行時間を必要とし，これは AV 300 内通信と AV 300 間の通信の遅延の差と場合 2a と 2b での AV 300 間通信数の差との積となっていると考えられる．（通信遅延の差を3ミリ秒と仮定すると，この積は約 98 秒となり，この考えの正しさがうらづけられる．）このことから，（当然のことではあるが）システムを分割するときには，できるだけスレーブ間の通信が少なくなるように分割するのが得策

であることがわかる．

9. おわりに

今回作成した D^2AS は，現在 SUN 3, SUN 4, SONY NEWS (CISC), AV 300 で利用可能であり，プログラムは C++ で約 8,000 行である．

D^2AS は大規模な分散システム上での分散アルゴリズムのシミュレーションを目的として開発された分散型のシミュレータである．われわれは2種類のシミュレーション方法を D^2AS に実現し，検討した．

第1の方法では，複数のシミュレータ（スレーブ）がフェーズごとに同期をとりながら担当する部分システムのシミュレーションを進めていくものであり，シミュレータの素直な分散になっている．しかしながら，この方法では，同期に時間がかかりすぎるために，2台以上への分散は無意味であり，したがって，従来に比べ格段に大きいシステムをシミュレートすることには（シミュレーション時間の節約の点から）無理がある．この方法の改良はこの研究最大の今後に残された課題である．

第2の方法では，複数のシミュレータは互いに同期をとらずに，担当部分のシミュレーションを進める．この場合には，シミュレーションの結果に，D^2AS を実行する LAN の状態が反映されてしまうという欠点を持つが，分散化の効果が約 100% 現れるので，スレーブ数を増やせば，十分に大きなサイズの分散システムもシミュレート可能であると考えられる．

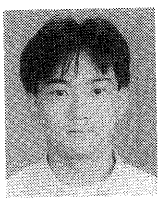
D^2AS のデッドロック検出方法については，第6章に述べた．単体の計算機上の分散アルゴリズムシミュレータは，デッドロックを容易に検知できるのに対し，分散型のシミュレータでは，さまざまな問題をはらんでいる．問題の安全な解決を目指したので，D^2AS ではすべてのスレーブでのシミュレーションをいったん中断し，大域スナップショットをとり，大域 WFG を作製した．これは通常の分散システムでは許されないことであり，シミュレータの場合でも，できればシミュレーションを行いながら，大域 WFG を作製したい．なお，デッドロック検出方式の評価は，対象となる分散システムや分散アルゴリズムはもちろんのこと，デッドロック検出に乗り出すタイミングや頻度などにも依存する複雑な問題であり，改めて報告する予定である．その内容は一部文献10)に報告されているので，興味のある方は，そちらを参照願いたい．

謝辞 本論文を丁寧にご査読いただいた査読者の方々に感謝いたします．その結果，本論文を多くの点で改善する機会を持つことができました．なお，本研究は一部電気通信普及財団の援助を受けて行われた．

参考文献

1) Bracha, G. and Toueg, S.: Distributed Deadlock Detection, *Distributed Computing*, Vol. 2, pp. 127-138 (1987).
2) Chandy, M. and Lamport, L.: Distributed Snapshot: Determing Global States of Distributed Systems, *ACM Trans. Prog. Lang. Syst.*, Vol. 3, No. 1, pp. 63-75 (1985).
3) Chang, E. and Roberts, R.: An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes, *Comm. ACM*, Vol. 22, No. 5, pp. 281-283 (1979).
4) 萩原，増沢：分散アルゴリズム，情報処理，Vol. 31, No. 9, pp. 1245-1256 (1990).
5) 池川，山下，阿江：リングネットワークにおけるリーダ選挙アルゴリズムの実験的評価，電子情報通信学会論文誌，Vol. J 73-D-I, No. 3, pp. 261-268 (1990).
6) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol. 21, No. 7, pp. 558-565 (1978).
7) Lamport, L. and Lynch, N. A.: Distributed Computing: Models and Methods, *Hand Book of Theoretical Computer Science, Volume B: Formal Models and Semantics*, van Leeuwen, J. (ed.), pp. 1157-1200, The MIT Press, Cambridge, Massachusetts (1990).
8) 永松：木構造ネットワークの負荷分散アルゴリズムについて，広島大学卒業論文 (1988).
9) Stephen, C. D. and Kathy, T. S.（小山裕司監訳）：C++言語入門，アスキー出版 (1990),
10) 谷江：分散デッドロック検知アルゴリズムの実験的評価，広島大学卒業論文 (1993).
11) 都倉：分散アルゴリズム設計支援ツール，情報処理，Vol. 30, No. 4, pp. 380-386 (1989).

（平成5年1月18日受付）
（平成5年9月 8 日採録）

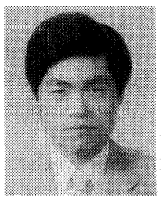


弘田 暢幸

昭和43年生．平成3年広島大学工学部第二類（電気系）卒業．平成5年広島大学大学院博士課程前期修了．同年横河・ヒューレット・パッカード(株)入社．

梅本　秀樹

昭和43年生．平成4年広島大学工学部第二類（電気系）卒業．現在，同大大学院博士課程前期在学中．ユーザインタフェース，計算幾何学等に興味を持つ．

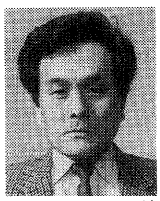


相原　玲二（正会員）

昭和33年生．昭和56年広島大学工学部第二類（電気系）卒業．昭和61年同大大学院博士課程修了．工学博士．同大助手を経て，現在，同大集積化システム研究センター助教授．マルチプロセッサシステムの設計，製作，ニューロコンピュータ設計，コンピュータネットワークの研究に従事．電子情報通信学会，IEEE-CS 各会員．

山下　雅史（正会員）

昭和 49 年京都大学工学部情報卒業．昭和 52 年同大大学院修士課程修了．昭和 55 年名古屋大学大学院博士課程修了．工学博士．豊橋技術科学大学助手，広島大学工学部第二類助教授を経て，平成4年教授となり現在に至る．この間，昭和61年より約1年間，カナダサイモンフレーザー大学客員研究員．分散／並列のアルゴリズム／システムに興味を持つ．電子情報通信学会，日本応用数理学会，ACM 各会員．



阿江　　忠（正会員）

昭和 39 年東北大学工学部通信卒業．昭和 44 年同大大学院博士課程修了．工学博士．東北大学助手，広島大学助教授を経て，昭和 57 年広島大学教授（工学部第二類計算機工学教育科目担当）となり現在に至る．この間，昭和49年より1年間，仏グルノーブル大学客員研究員．現在，主として，分散処理システム（マルチプロセッサおよび LAN）の設計，制作ならびに性能評価の研究に従事．電子情報通信学会，ACM，IEEE 各会員．